

Graphics Library Help

Manual for Microchip Graphics Library

Made with **Doc-O-Matic**.

Table of Contents

Introduction	1
Release Notes	2
Getting Started	23
Demo Projects	31
Demo Summary	31
Microchip Application Library Abbreviations	32
Demo Compatibility Matrix	32
Library Architecture	33
Graphics Object Layer	33
Object Rendering	34
Graphics Primitive Layer	35
Display Device Driver Layer	36
Library API	37
Graphics Library Configuration	37
Graphics Object Layer Configuration	37
Input Device Selection	37
USE_KEYBOARD Macro	38
USE_TOUCHSCREEN Macro	38
Focus Support Selection	38
USE_FOCUS Macro	38
Graphics Object Selection	39
USE_ANALOGCLOCK Macro	39
USE_BUTTON Macro	40
USE_BUTTON_MULTI_LINE Macro	40
USE_CHECKBOX Macro	40
USE_DIGITALMETER Macro	40
USE_EDITBOX Macro	40
USE_GROUPBOX Macro	41
USE_LISTBOX Macro	41
USE_METER Macro	41

USE_PICTURE Macro	41
USE_PROGRESSBAR Macro	41
USE_RADIOBUTTON Macro	42
USE_ROUNDDIAL Macro	42
USE_SLIDER Macro	42
USE_STATICTEXT Macro	42
USE_WINDOW Macro	42
USE_CUSTOM Macro	43
USE_GOL Macro	43
USE_TEXTENTRY Macro	43
Graphics Primitive Layer Configuration	43
Image Compression Option	43
USE_COMP_IPU Macro	44
USE_COMP_RLE Macro	44
Font Type Selection	44
USE_MULTIBYTECHAR Macro	45
USE_UNSIGNED_XCHAR Macro	45
Advanced Font Features Selection	45
USE_ANTIALIASED_FONTS Macro	46
Gradient Bar Rendering	46
USE_GRADIENT Macro	46
Transparent Color Feature in PutImage()	46
USE_TRANSPARENT_COLOR Macro	46
Alpha Blend Option	47
USE_ALPHABLEND_LITE Macro	47
External Memory Buffer	47
Display Device Driver Layer Configuration	47
USE_ALPHABLEND Macro	48
USE_DOUBLE_BUFFERING Macro	48
GFX_LCD_TYPE Macro	48
GFX_LCD_CSTN Macro	49
GFX_LCD_MSTN Macro	49
GFX_LCD_OFF Macro	49
GFX_LCD_TFT Macro	50
STN_DISPLAY_WIDTH Macro	50
STN_DISPLAY_WIDTH_16 Macro	50
STN_DISPLAY_WIDTH_4 Macro	50
STN_DISPLAY_WIDTH_8 Macro	51
Application Configuration	51
Configuration Setting	51
USE_NONBLOCKING_CONFIG Macro	51
Font Source Selection	52

USE_FONT_FLASH Macro	52
USE_FONT_EXTERNAL Macro	52
USE_GFX_FONT_IN_PROGRAM_SECTION Macro	53
Image Source Selection	53
USE_BITMAP_FLASH Macro	53
USE_BITMAP_EXTERNAL Macro	54
Miscellaneous	54
USE_BITMAP_NO_PADDING_LINE Macro	55
USE_PALETTE_EXTERNAL Macro	55
USE_PALETTE Macro	55
COLOR_DEPTH Macro	55
GFX_free Macro	55
GFX_malloc Macro	56
GraphicsConfig.h Example	56
Hardware Profile	56
PMP Interface	57
USE_8BIT_PMP Macro	57
USE_16BIT_PMP Macro	57
Development Platform Used	58
EXPLORER_16 Macro	58
PIC24FJ256DA210_DEV_BOARD Macro	59
MEB_BOARD Macro	59
PIC_SK Macro	59
Graphics PICTail Used	60
GFX_PICTAIL_LCC Macro	60
GFX_PICTAIL_V3 Macro	61
GFX_PICTAIL_V3E Macro	61
Display Controller Used	61
GFX_USE_DISPLAY_CONTROLLER_DMA Macro	62
GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 Macro	63
GFX_USE_DISPLAY_CONTROLLER_S1D13517 Macro	63
GFX_USE_DISPLAY_CONTROLLER_SSD1926 Macro	63
Display Panel Used	64
GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q Macro	65
GFX_USE_DISPLAY_PANEL_TFT_640480_8_E Macro	65
GFX_USE_DISPLAY_PANEL_TFT_800480_33_E Macro	65
GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E Macro	66
Device Driver Options	66
DISP_DATA_WIDTH Macro	68
DISP_ORIENTATION Macro	68
DISP_HOR_RESOLUTION Macro	68
DISP_VER_RESOLUTION Macro	69

DISP_HOR_FRONT_PORCH Macro	69
DISP_HOR_BACK_PORCH Macro	69
DISP_VER_FRONT_PORCH Macro	69
DISP_VER_BACK_PORCH Macro	70
DISP_HOR_PULSE_WIDTH Macro	70
DISP_VER_PULSE_WIDTH Macro	70
DISP_INV_LSHIFT Macro	70
HardwareProfile.h Example	71
Graphics Object Layer API	72
GOL Objects	72
GOL_OBJ_TYPE Enumeration	75
OBJ_HEADER Structure	76
DRAW_FUNC Type	76
FREE_FUNC Type	77
MSG_DEFAULT_FUNC Type	77
MSG_FUNC Type	77
Analog Clock	77
Analog Clock States	78
AcCreate Function	80
AcDraw Function	81
AcSetHour Function	83
AcSetMinute Function	83
AcSetSecond Function	84
ANALOGCLOCK Structure	84
Button	85
Button States	87
BtnCreate Function	90
BtnDraw Function	92
BtnGetText Macro	93
BtnSetText Function	93
BtnGetBitmap Macro	94
BtnSetBitmap Macro	95
BtnMsgDefault Function	95
BtnTranslateMsg Function	96
BUTTON Structure	98
Chart	99
Chart States	101
Data Series Status Settings	105
Chart Examples	105
ChCreate Function	106
ChDraw Function	108
ChAddDataSeries Function	109

ChRemoveDataSeries Function	110
ChShowSeries Macro	111
ChHideSeries Macro	111
ChGetShowSeriesCount Macro	112
ChGetShowSeriesStatus Macro	113
ChSetValueLabel Macro	113
ChGetValueLabel Macro	114
ChGetValueMax Macro	114
ChGetValueMin Macro	115
ChSetValueRange Function	115
ChGetValueRange Macro	116
ChSetSampleLabel Macro	117
ChGetSampleLabel Macro	117
ChGetSampleStart Macro	118
ChGetSampleEnd Macro	118
ChSetPercentRange Function	119
ChGetPercentRange Macro	119
ChSetSampleRange Function	120
ChGetSampleRange Macro	121
ChGetPercentMax Macro	121
ChGetPercentMin Macro	122
ChSetColorTable Macro	123
ChGetColorTable Macro	123
ChSetTitle Macro	124
ChGetTitle Macro	124
ChSetTitleFont Macro	125
ChGetTitleFont Macro	125
ChGetAxisLabelFont Macro	126
ChSetAxisLabelFont Macro	127
ChGetGridLabelFont Macro	127
ChSetGridLabelFont Macro	128
ChFreeDataSeries Function	128
ChTranslateMsg Function	129
CHART Structure	130
DATASERIES Structure	130
CHARTPARAM Structure	131
Color Table	132
Checkbox	135
Check Box States	137
CbCreate Function	139
CbDraw Function	140
CbGetText Macro	141

CbSetText Function	141
CbMsgDefault Function	142
CbTranslateMsg Function	143
CHECKBOX Structure	144
Round Dial	144
Dial States	145
RdiaCreate Function	147
RdiaDraw Function	148
RdialIncVal Macro	148
RdiaDecVal Macro	149
RdiaGetVal Macro	150
RdiaSetVal Macro	150
RdiaMsgDefault Function	151
RdiaTranslateMsg Function	152
ROUNDDIAL Structure	153
Digital Meter	154
Digital Meter States	155
DmCreate Function	157
DmDraw Function	158
DmGetValue Macro	159
DmSetValue Function	159
DmDecVal Macro	160
DmIncVal Macro	160
DmTranslateMsg Function	161
DIGITALMETER Structure	162
Edit Box	162
Edit Box States	164
EbCreate Function	166
EbDraw Function	167
EbGetText Macro	167
EbSetText Function	168
EbAddChar Function	169
EbDeleteChar Function	169
EbMsgDefault Function	170
EbTranslateMsg Function	171
EDITBOX Structure	171
Grid	172
Grid States	173
Grid Item States	175
GridCreate Function	177
GridDraw Function	178
GridClearCellState Function	179

GridGetFocusX Macro	180
GridGetFocusY Macro	180
GRID_OUT_OF_BOUNDS Macro	181
GRID_SUCCESS Macro	181
GridFreeItems Function	181
GridGetCell Function	182
GridSetCell Function	183
GridSetCellState Function	183
GridSetFocus Function	184
GridMsgDefault Function	185
GridTranslateMsg Function	186
GRID Structure	187
GRIDITEM Structure	188
Group Box	188
Group Box States	189
GbCreate Function	190
GbDraw Function	192
GbGetText Macro	192
GbSetText Function	193
GbTranslateMsg Function	194
GROUPBOX Structure	194
List Box	195
List Box States	197
List Item Status	199
LbCreate Function	199
LbDraw Function	201
LbGetItemList Macro	202
LbAddItem Function	202
LbDeleteItem Function	204
LbChangeSel Function	204
LbSetSel Macro	205
LbGetSel Function	205
LbGetFocusedItem Function	206
LbSetFocusedItem Function	207
LbGetCount Macro	207
LbGetVisibleCount Macro	208
LbSetBitmap Macro	208
LbGetBitmap Macro	209
LbDeleteItemsList Function	210
LbMsgDefault Function	210
LbTranslateMsg Function	211
LISTBOX Structure	212

LISTITEM Structure	213
Meter	213
Meter States	215
MtrCreate Function	216
MtrDraw Function	218
MtrSetVal Function	219
MtrGetVal Macro	219
MtrDecVal Macro	220
MtrIncVal Macro	220
MtrSetScaleColors Macro	221
MtrSetTitleFont Macro	222
MtrSetValueFont Macro	222
METER_TYPE Macro	223
MTR_ACCURACY Macro	223
MtrMsgDefault Function	224
MtrTranslateMsg Function	224
METER Structure	225
Picture Control	226
Picture States	227
PictCreate Function	228
PictDraw Function	229
PictSetBitmap Macro	230
PictGetBitmap Macro	230
PictGetScale Macro	231
PictSetScale Macro	232
PictTranslateMsg Function	232
PICTURE Structure	233
Progress Bar	233
Progress Bar States	235
PbCreate Function	236
PbDraw Function	237
PbSetRange Function	238
PbGetRange Macro	238
PbSetPos Function	239
PbGetPos Macro	240
PbTranslateMsg Function	240
PROGRESSBAR Structure	241
RadioButton	242
RadioButton States	243
RbCreate Function	245
RbDraw Function	246
RbGetCheck Function	247

RbSetCheck Function	248
RbGetText Macro	249
RbSetText Function	249
RbMsgDefault Function	250
RbTranslateMsg Function	251
RADIOBUTTON Structure	252
Slider/Scroll Bar	252
Slider States	254
SldCreate Function	256
SldDraw Function	258
SldSetPage Function	259
SldGetPage Macro	259
SldSetPos Function	260
SldGetPos Macro	261
SldSetRange Function	262
SldGetRange Macro	262
SldIncPos Macro	263
SldDecPos Macro	264
SldMsgDefault Function	265
SldTranslateMsg Function	266
SLIDER Structure	267
Static Text	267
Static Text States	269
StCreate Function	270
StDraw Function	271
StGetText Macro	272
StSetText Function	273
StTranslateMsg Function	273
STATICTEXT Structure	274
Text Entry	275
TextEntry States	277
Key Command Types	278
TeCreate Function	279
TeDraw Function	280
TeGetBuffer Macro	281
TeSetBuffer Function	281
TeClearBuffer Function	282
TeGetKeyCommand Function	283
TeSetKeyCommand Function	283
TeCreateKeyMembers Function	284
TeAddChar Function	285
TelsKeyPressed Function	285

TeSpaceChar Function	286
TeDelKeyMembers Function	286
TeSetKeyText Function	287
TeMsgDefault Function	288
TeTranslateMsg Function	289
TEXTENTRY Structure	290
KEYMEMBER Structure	291
Window	292
Window States	293
WndCreate Function	295
WndDraw Function	296
WndGetText Macro	297
WndSetText Function	297
WndTranslateMsg Function	298
WINDOW Structure	299
Object States	299
Common Object States	300
FOCUSSED Macro	300
DISABLED Macro	301
HIDE Macro	301
DRAW Macro	301
DRAW_FOCUS Macro	301
DRAW_UPDATE Macro	301
GetState Macro	302
ClrState Macro	302
SetState Macro	303
Object Management	304
GOLAddObject Function	305
GOLFFindObject Function	306
GOLRedraw Macro	307
GOLRedrawRec Function	308
GOLDraw Function	308
GOLDrawComplete Macro	309
GOLDrawCallback Function	310
GOLFree Function	311
GetObjType Macro	312
GetObjID Macro	312
GetObjNext Macro	313
GOLDeleteObject Function	314
GOLDeleteObjectByID Function	314
GOLNewList Macro	315
GOLGetList Macro	315

GOLSetList Macro	316
GOLSetFocus Function	317
IsObjUpdated Macro	317
GOLInit Function	318
GOLGetFocus Macro	319
GOLCanBeFocused Function	319
GOLGetFocusNext Function	320
GOLGetFocusPrev Function	320
GOLPanelDraw Macro	320
GOLPanelDrawTsk Function	321
GOLTTwoTonePanelDrawTsk Function	322
GOL Messages	322
GOLMsg Function	325
GOLMsgCallback Function	326
GOL_MSG Structure	327
TRANS_MSG Enumeration	328
INPUT_DEVICE_EVENT Enumeration	330
INPUT_DEVICE_TYPE Enumeration	330
Scan Key Codes	331
SCAN_BS_PRESSED Macro	332
SCAN_BS_RELEASED Macro	332
SCAN_CR_PRESSED Macro	332
SCAN_CR_RELEASED Macro	332
SCAN_DEL_PRESSED Macro	332
SCAN_DEL_RELEASED Macro	333
SCAN_DOWN_PRESSED Macro	333
SCAN_DOWN_RELEASED Macro	333
SCAN_END_PRESSED Macro	333
SCAN_END_RELEASED Macro	333
SCAN_HOME_PRESSED Macro	334
SCAN_HOME_RELEASED Macro	334
SCAN_LEFT_PRESSED Macro	334
SCAN_LEFT_RELEASED Macro	334
SCAN_PGDOWN_PRESSED Macro	334
SCAN_PGDOWN_RELEASED Macro	335
SCAN_PGUP_PRESSED Macro	335
SCAN_PGUP_RELEASED Macro	335
SCAN_RIGHT_PRESSED Macro	335
SCAN_RIGHT_RELEASED Macro	335
SCAN_SPACE_PRESSED Macro	336
SCAN_SPACE_RELEASED Macro	336
SCAN_TAB_PRESSED Macro	336

SCAN_TAB_RELEASED Macro	336
SCAN_UP_PRESSED Macro	336
SCAN_UP_RELEASED Macro	337
Style Scheme	337
GOLCreateScheme Function	339
GOLSetScheme Macro	339
GOLGetScheme Macro	340
GOLGetSchemeDefault Macro	341
GOL_SCHEME Structure	341
Default Style Scheme Settings	342
FONTDEFAULT Variable	342
GOLFondDefault Variable	342
GOL_EMBOSS_SIZE Macro	342
GOLSchemeDefault Variable	343
RGBConvert Macro	343
GOL Global Variables	344
_pDefaultGolScheme Variable	344
_pGolObjects Variable	344
_pObjectFocused Variable	345
Graphics Primitive Layer API	345
Text Functions	345
FONT_HEADER Structure	346
FONT_FLASH Structure	347
FONT_EXTERNAL Type	348
SetFont Function	348
GetFontOrientation Macro	348
SetFontOrientation Macro	349
GFX_Font_GetAntiAliasType Macro	350
GFX_Font_SetAntiAliasType Macro	350
OutChar Function	351
OutText Function	351
OutTextXY Function	352
GetTextHeight Function	353
GetTextWidth Function	354
XCHAR Macro	354
Anti-Alias Type	355
ANTIALIAS_OPAQUE Macro	355
ANTIALIAS_TRANSLUCENT Macro	355
Gradient	355
BarGradient Function	356
BevelGradient Function	358
GFX_GRADIENT_TYPE Enumeration	359

GFX_GRADIENT_STYLE Structure	359
Line Functions	360
Line Function	360
LineRel Macro	361
LineTo Macro	361
SetLineThickness Macro	362
SetLineType Macro	362
Line Types	363
SOLID_LINE Macro	363
DASHED_LINE Macro	363
DOTTED_LINE Macro	364
Line Size	364
NORMAL_LINE Macro	364
THICK_LINE Macro	364
Rectangle Functions	365
Bar Function	365
Rectangle Macro	366
DrawPoly Function	366
Circle Functions	367
Circle Macro	368
FillCircle Macro	368
Arc Function	369
DrawArc Function	371
Bevel Function	371
FillBevel Function	373
SetBevelDrawType Macro	374
Graphic Cursor	375
GetX Macro	375
GetY Macro	375
MoveRel Macro	376
MoveTo Macro	376
Alpha Blending Functions	377
SetAlpha Macro	378
GetAlpha Macro	378
AlphaBlendWindow Function	379
Bitmap Functions	380
PutImage Macro	381
PutImagePartial Function	381
GetImageHeight Function	383
GetImageWidth Function	384
BITMAP_HEADER Structure	384
Bitmap Settings	385

IMAGE_NORMAL Macro	385
IMAGE_X2 Macro	385
Bitmap Source	385
External Memory	385
ExternalMemoryCallback Function	386
EXTERNAL_FONT_BUFFER_SIZE Macro	387
Memory Type	388
Set Up Functions	388
ClearDevice Function	388
InitGraph Function	389
GFX_RESOURCE Enumeration	389
GFX_IMAGE_HEADER Structure	390
IMAGE_FLASH Structure	391
IMAGE_RAM Structure	391
GFX_EXDATA Structure	391
Display Device Driver Layer API	392
Display Device Driver Level Primitives	392
GetPixel Function	393
PutPixel Function	394
GetColor Macro	394
SetColor Macro	395
GetMaxX Macro	395
GetMaxY Macro	396
SetClip Function	396
SetClipRgn Function	397
GetClipBottom Macro	398
GetClipLeft Macro	398
GetClipRight Macro	398
GetClipTop Macro	399
CLIP_DISABLE Macro	399
CLIP_ENABLE Macro	399
TransparentColorEnable Function	400
TransparentColorDisable Macro	401
GetTransparentColorStatus Macro	401
GetTransparentColor Macro	402
TRANSPARENT_COLOR_DISABLE Macro	402
TRANSPARENT_COLOR_ENABLE Macro	402
DisplayBrightness Function	402
GetPageAddress Macro	403
CopyBlock Function	404
CopyPageWindow Function	404
CopyWindow Function	405

SetActivePage Function	406
SetVisualPage Function	407
Color Definition	407
BLACK Macro	408
BLUE Macro	408
BRIGHTBLUE Macro	408
BRIGHTCYAN Macro	408
BRIGHTGREEN Macro	409
BRIGHTMAGENTA Macro	409
BRIGHTRED Macro	409
BRIGHTYELLOW Macro	409
BROWN Macro	409
CYAN Macro	410
DARKGRAY Macro	410
GRAY0 Macro	410
GRAY1 Macro	410
GRAY2 Macro	410
GRAY3 Macro	411
GRAY4 Macro	411
GRAY5 Macro	411
GRAY6 Macro	411
GREEN Macro	411
LIGHTBLUE Macro	412
LIGHTCYAN Macro	412
LIGHTGRAY Macro	412
LIGHTGREEN Macro	412
LIGHTMAGENTA Macro	412
LIGHTRED Macro	413
MAGENTA Macro	413
RED Macro	413
WHITE Macro	413
YELLOW Macro	413
Display Device Driver Control	414
IsDeviceBusy Function	414
ResetDevice Function	414
Advanced Display Driver Features	415
Alpha Blending	415
GFXGetPageOriginAddress Function	416
GFXGetPageXYAddress Function	417
Transitions	418
GFXTransition Function	419
GFXSetupTransition Function	419

GFXExecutePendingTransition Function	420
GFXIsTransitionPending Function	421
GFX_TRANSITION_DIRECTION Enumeration	421
GFX_TRANSITION_TYPE Enumeration	422
Double Buffering	422
SwitchOffDoubleBuffering Function	424
SwitchOnDoubleBuffering Function	425
InvalidateRectangle Function	425
RequestDisplayUpdate Function	426
UpdateDisplayNow Function	426
Microchip Graphics Controller	427
Rectangle Copy Operations	428
Decompressing DEFLATED data	442
Palette Mode	443
Set Up Display Interface	455
External or Internal Memory and Palettes	465

Image Decoders 468

Image Decoders API 472

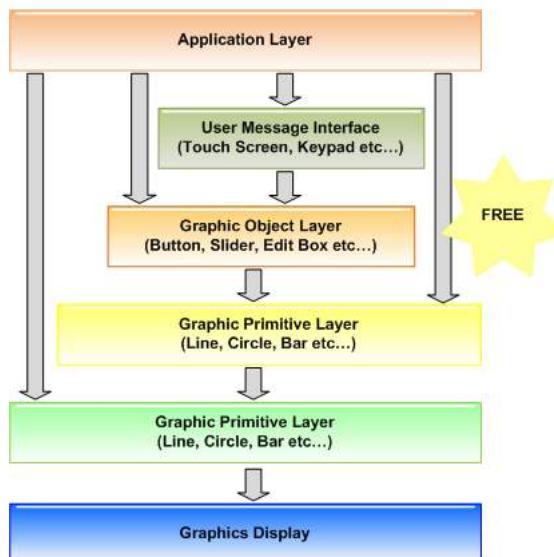
Image Decoder Configuration	472
IMG_SUPPORT_BMP Macro	473
IMG_SUPPORT_GIF Macro	473
IMG_SUPPORT_JPEG Macro	473
IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK Macro	474
IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT Macro	474
IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT Macro	474
ImageDecode Function	475
ImageDecoderInit Function	476
ImageLoopCallbackRegister Function	476
ImageDecodeTask Function	477
ImageFullScreenDecode Macro	477
ImageAbort Macro	478
_BMPDECODER Structure	479
_GIFDECODER Structure	479
_JPEGDECODER Structure	480

Miscellaneous Topics 483

Starting a New Project	483
Changing the default Font	489
Advanced Font Features	490
Using Primitive Rendering Functions in Blocking and Non-Blocking Modes	491
Using Microchip Graphics Module Color Look Up Table in Applications	492
Converting Images to Use a Common Palette in GIMP	497
How to Define Colors in your Applications	501
Connecting RGB data bus	502
Adding New Device Driver	503
References	505
Index	a

1 Introduction

This document explains how to get started with Microchip Graphics Library solution. It presents the available resources and where to obtain these resources. It also includes the Application Programming Interface (API) of the Microchip Graphics Library.



The Microchip Graphics Library is highly modular and is optimized for Microchip's 16-bit microcontrollers. Additionally, the library is free for Microchip customers, easy to use, and has an open documented interface for new driver support, which requires creation of only one C file.

The Microchip Graphics Library Software Version 3.06.00 supports the following features:

- Configurable graphic resolution
- Up to 16-bit or 65K colors
- 2D objects such as line, circle, text, rectangle, polygon, bar
- 3D objects such as buttons, meter, dial, list box, edit box, check box, radio buttons, window, group box, slider.
- Image, animation
- Variety of user input device such as touch screen, keypad etc...
- Multiple fonts
- Unicode fonts
- PIC24 support, PIC32 support

2 Release Notes

Microchip Graphics Library Release Notes

Version 3.06.00 (2012-07-18)

This library provides support for the display modules with built in graphics controller and displays connected to external graphics controller. Documentation for the Microchip Graphics Library can be found in this file:

- **Graphics Library Help.chm**

Version Log

New:

- Partial rendering of images now supported (see PutImagePartial (█)()).
- Double Buffering is now supported in Microchip Low-Cost Controllerless (LCC) Graphics Display Driver.
- Added dsPIC33EPXXX device family support.
- Added S1D13522 EPD Controller Driver.
- Added E-Paper Epson Demo.
- Added Alpha-Blend support for Bar (█)() function.
- Graphics Resource Converter (GRC) now allows for padding and non-padding bitmap images. Bitmap images are padded which means that each horizontal line will start on a byte boundary. The option has been added to allow for conversion of bitmap resources to be non-padded which allows the least resource space and controllers with windowing that auto increments to use them.
- For XC16 or C30 builds, internal fonts can now be placed program memory. If the font data or a combination of font data resources exceed the 32 Kbyte limit of the data memory space, a define, USE_GFX_FONT_IN_PROGRAM_SECTION (█), should be defined in graphics configuration header (GraphicsConfig.h). This will place the font resource data in program memory space.

Changes:

- Graphics Object Demo now uses images that are RLE compressed.
- Address range check for GFX_EPMP_CS1_MEMORY_SIZE (█) and GFX_EPMP_CS2_MEMORY_SIZE (█) in Graphics Module in PIC24FJ256DA210 Display Driver file (mchpGfxDrv.c) is modified for strict checks of allocated address pins for the EPMP. See "Migration Changes" below for the address lines needed to be allocated.
- Swapped the bit orientation for the 1 BPP bitmap images. The previous versions of the library expects the left most pixel at the MSBit. This has been changed so the the left most pixel is located at the LSBit. The change was made to accommodate controllers that have windowing. This also makes the pixel orientation consistent with the pixel orientation of 4bpp images.

Fixes:

- Fix FillBevel (█)() & FillCircle (█)() to avoid rendering lines more than once.

Deprecated Items:

The following Resistive Touch Screen macro names are replaced for readability and flexibility if use:

- TRIS_XPOS - replaced by ResistiveTouchScreen_XPlus_Config_As_Input()
- TRIS_YPOS - replaced by ResistiveTouchScreen_YPlus_Config_As_Input()
- TRIS_XNEG - replaced by ResistiveTouchScreen_XMinus_Config_As_Input()
- TRIS_YNEG - replaced by ResistiveTouchScreen_YMinus_Config_As_Output()
- LAT_XPOS - replaced by ResistiveTouchScreen_XPlus_Drive_High()
- LATS_YPOS - replaced by ResistiveTouchScreen_YPlus_Drive_High()
- LAT_XNEG - replaced by ResistiveTouchScreen_XMinus_Drive_Low()

- LAT_YNEG - replaced by ResistiveTouchScreen_YMinus_Drive_Low()

Migration Changes:

- To use the new Resistive Touchscreen macros, replace the TouchScreenResistive.c file with the version in this release. Then replace the hardware profile macros to use the new macro names. Existing hardware profile can still be used but build warnings will appear.
- If custom display driver is used and the PutImage (🔗) functions are implemented for faster rendering, the new partial image rendering feature requires these PutImage (🔗) functions to be modified. See PutImagePartial (🔗) API description and implementation in Primitive.c for details.
- Address range check for GFX_EPMP_CS1_MEMORY_SIZE (🔗) and GFX_EPMP_CS2_MEMORY_SIZE (🔗) that are defined in hardware profile are modified when using the Graphics Module in PIC24FJ256DA210 Device and using external memory for display buffer, the driver file (mchpGfxDrv.c). This check allocates address pins for the EPMP. Modify the GFX_EPMP_CS1_MEMORY_SIZE (🔗) and GFX_EPMP_CS2_MEMORY_SIZE (🔗) values set in the hardware profile to match the table shown below.
- GFX_EPMP_CSx_MEMORY_SIZE <= 0x20000 (bytes) - Use PMA[15:0]
- 0x20000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x40000 (bytes) - Use PMA[16:0]
- 0x40000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x80000 (bytes) - Use PMA[17:0]
- 0x80000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x100000 (bytes) - Use PMA[18:0]
- 0x100000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x200000 (bytes) - Use PMA[19:0]
- 0x200000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x400000 (bytes) - Use PMA[20:0]
- 0x400000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x800000 (bytes) - Use PMA[21:0]
- 0x800000 (bytes) < GFX_EPMP_CSx_MEMORY_SIZE <= 0x1000000 (bytes) - Use PMA[22:0]
- 1BPP images needs to be regenerated using the "Graphics Resource Converter" since the bit orientation is swapped. When rendering a 1 BPP image, the PutImage (🔗) function will expect the left most pixel to be located in the LSBit of each word.
- To utilize the new feature where the fonts can be placed in the program memory, to remove the 32 KByte limit for data space in XC16 or C30 builds, fonts must be regenerated using the "Graphics Resource Converter" and then add #define USE_GFX_FONT_IN_PROGRAM_SECTION (🔗) in GraphicsConfig.h.

Known Issues:

- Extended glyph for certain font (such as Thai) when used with Static text widget is clipped. Future version will add additional vertical text alignment to the static text widget
- Anti-aliasing and extended glyph features are not supported when using PIC24FJ256DA210 CHRGPU.
- SPI flash programming on the Epson S1D13517 PICtail board Rev 1.1 is not always reliable, the S1D13517 demo no longer uses external memory flash in the example.
- When using PIC24FJ256GB210 PIM with Explorer 16 board that has a 5v Lumex LCD display, the S1D13517 demo does not run correctly.
- When using XC16 Compiler V1.00, add "-fno-ivopts" compile option. This is a known issue V1.00 of XC16.

Application Notes

- [AN1136](#) How to Use Widgets in Microchip Graphics Library
- [AN1182](#) Fonts in the Microchip Graphics Library
- [AN1227](#) Using a Keyboard with the Microchip Graphics Library
- [AN1246](#) How to Create Widgets in Microchip Graphics Library
- [AN1368](#) Developing Graphics Applications using PIC MCU with Integrated Graphics Controller

PIC Family

This version of the library supports PIC24, dsPIC and PIC32 Family.

Development Tools

This graphics library release (v3.06) was tested with IDEs MPLAB v8.85 and MPLAB X 1.20; compilers XC16 v1.00 and XC32 v1.00.

There is a known compatibility issue between the graphics library and C30 v3.25. using C30 3.25 is not recommended for

graphics library development.

Documentation of Resources and Utilities

- The Graphics Library Help File and API document is located in <install_dir>/Microchip/Help.
- Graphics Library.pdf
- "Graphics Resource Converter" documentation is located in <install_dir>/Microchip/Graphics/bin/grc.
- Graphics Resource Converter Help.pdf
- External Memory Programmer documentation is located in <install_dir>/Microchip/Graphics/bin/memory_programmer.
- External Memory Programmer Help.pdf

where: "install_dir" - is the location of the Microchip Application Library installation.

Display Modules

The display driver layer of the library is organized to easily switch from one display driver to another. Use of customized display driver is allowed and in the Display Device Driver Layer section. Following graphics controllers are supported in this version:

Display Module	Interface	File Names
Microchip Graphics Display Driver - customizable driver for RGB Glass. Currently used in PIC24FJ256DA210 device family.	RGB	mchpGfxDrv.c, mchpGfxDrv.h
Microchip Low-Cost Controllerless (LCC) Graphics Display Driver - customizable driver for RGB Glass. Currently used for selected PIC32MX device families.	RGB	mchpGfxLCC.c, mchpGfxLCC.h
Samsung S6D0129/S6D0139	8/16 bit PMP	drvTFT001.c, drvTFT001.h
LG LGDP4531	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Renesas R61505U/R61580	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Orise Technology SPDF5408	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Epson S1D13517	8/16 bit PMP	S1D13517.c, S1D13517.h
Epson S1D13522	8/16 bit PMP, SPI	S1D13522.c, S1D13522.h
Solomon Systech SSD1926	8/16 bit PMP	SSD1926.c, SSD1926.h
Solomon Systech SSD1289	8/16 bit PMP	drvTFT002.c, drvTFT002.h
Solomon Systech SSD1339 for OLED displays	8 bit PMP	SSD1339.c, SSD1339.h
Solomon Systech SSD1303 for OLED displays	8 bit PMP	SH1101A_SSD1303.c, SH1101A_SSD1303.h
Solomon Systech SSD1305 for OLED displays	8 bit PMP	SSD1305.c, SSD1305.h
Solomon Systech SSD2119	8/16 bit PMP	drvTFT002.c, drvTFT002.h
Sino Wealth SH1101A for OLED displays	8 bit PMP	SH1101A_SSD1303.c, SH1101A_SSD1303.h
Sitronix ST7529	8 bit PMP	ST7529.c, ST7529.h
Hitech Electronics HIT1270	8 bit PMP	HIT1270.c, HIT1270.h
Ilitek ILI9320	8/16 bit PMP	drvTFT001.c, drvTFT001.h
Himax HX8347	8/16 bit PMP	HX8347.c, HX8347.h
UltraChip UC1610	8 bit PMP	UC1610.c, UC1610.h

Please refer to Adding New Device Driver (§) section to get more information on adding support for another LCD controller.

Widgets

In this version the following widgets are implemented:

- Analog Clock (§)

- Button (█)
- Chart (█)
- Checkbox (█)
- Dial (█)
- Digital Meter (█)
- Edit Box (█)
- Grid (█)
- Group Box (█)
- List Box (█)
- Meter (█)
- Picture Control (█)
- Progress Bar (█)
- Radio Button (█)
- Slider (█) / Scroll Bar (█)
- Static Text (█)
- Text Entry (█)
- Window (█)

This version of the library supports touch screen, side buttons and a variety of key pad configurations as a user input device.

Required Resources

The library utilizes the following estimated MCU resources (in # of bytes):

Module	Heap for PIC24F	Heap for PIC32	RAM for PIC24F	RAM for PIC32	ROM for PIC24F	ROM for PIC32
Primitives Layer	0	0	68	53	3375	8868
GOL	20 (per style scheme)	24 (per style scheme)	32	28	2076	5400
Button (█)	28 (per instance)	44 (per instance)	8	12	1002	2748
Chart (█)	48 (per instance)	76 (per instance)	94	104	11427	26364
Check Box (█)	22 (per instance)	36 (per instance)	2	4	894	2320
Dial (█)	30 (per instance)	40 (per instance)	8	12	1065	3228
Digital Meter (█)	28 (per instance)	56 (per instance)	2	4	1125	2202
Edit Box (█)	26 (per instance)	40 (per instance)	2	4	822	2332
Group Box (█)	24 (per instance)	36 (per instance)	8	12	903	2164
List Box (█)	28 (per instance), 12 (per item)	44 (per instance)	6	12	1809	2580
Meter (█)	52 (per instance)	68 (per instance)	36	40	2778	6788
Picture Control (█)	22 (per instance)	36 (per instance)	10	12	645	1512
Progress Bar (█)	24 (per instance)	36 (per instance)	12	16	1050	2452
Radio Button (█)	26 (per instance)	44 (per instance)	14	20	993	2632
Slider (█) / Scroll Bar (█)	32 (per instance)	44 (per instance)	20	24	2094	5720
Static Text (█)	22 (per instance)	36 (per instance)	8	12	747	1884
Text Entry (█)	34 (per instance), (24 per key)	52 (per instance)	22	28	2484	6376

Window (■)	24 (per instance)	40 (per instance)	2	4	804	1996
------------	-------------------	-------------------	---	---	-----	------

The heap is a dynamic memory requirement. It is released when screen is destroyed. Please consider screen with maximum number of objects to calculate worse case RAM requirement. If for worse case you have 6 buttons, 2 sliders and 2 edit box on screen that utilizes three style schemes then worse case dynamic memory requirement will be $6*28 + 2*32 + 2*26 + 3*20$; 344 bytes. The RAM requirement in column 4 doesn't change based on number of instances. If object is included in code then it will take fix amount of RAM irrespective of its usage.

Each font may require 7 – 10KB of program memory. This is for English fonts. This requirement may change for other languages with additional characters.

Images require additional memory. The memory requirement for images depends on color depth and size.

The fonts and images can be stored in internal memory or external memory. The external memory can be anything serial EEPROM, parallel Flash, SD card etc. The application provides physical interface code for these devices.

Previous Versions Log

v3.04.01 (2012-04-03)

New:

- No new items on this release.

Changes:

- Structure of this help file is modified. Section names that list APIs now has the API in the name.

Fixes:

- Fix BarGradient (■)() and BevelGradient (■)() when USE_NONBLOCKING_CONFIG (■) config is enabled.
- Added missing GbSetText (■)() function in GroupBox widget.
- Fix SetPalette (■)() to work with palette stored in external memory.

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE (■)
- EXTDATA - replaced by GFX_EXTDATA (■)
- BITMAP_FLASH - replaced by IMAGE_FLASH (■)
- BITMAP_RAM - replaced by IMAGE_RAM (■)
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA (■)
- EEPROM.h and EEPROM.c are replaced by the following files:
 - MCHP25LC256.c - source code
 - MCHP25LC256.h - header file
- in the HardwareProfile.h add #define USE_MCHP25LC256 to use the new driver.

Migration Changes:

- none

Known Issues:

- Extended glyph for certain font (such as Thai) when used with Static text widget is clipped. Future version will add additional vertical text alignment to the static text widget
- Anti-aliasing and extended glyph features are not supported when using PIC24FJ256DA210 CHRGPU.
- SPI flash programming on the Epson S1D13517 PICtail board Rev 1.1 is not always reliable, the S1D13517 demo no longer uses external memory flash in the example.
- When using PIC24FJ256GB210 PIM with Explorer 15 board with a 5v Lumex LCD display, the S1D13517 demo does not run correctly.

v3.04 (2012-02-15)

New:

- Font Table is updated to version 2 to accommodate anti-alias font, and extended glyph.
- Added anti-aliasing support for fonts (2bpp). See Primitive demo for an example.
- Added extended-glyph support for fonts. See demo in the AppNote demo - AN1182, This demo now includes Hindi and Thai font.
- Added 32-bit CRC code for resources to be placed in external memory. This CRC value can be used to verify if the data in external memory is valid or not. Refer to Graphics Resource Converter release notes for details. Demos ([\[1\]](#)) that uses external memory as a resource are now checking the CRC value, and if invalid, automatically requests for external memory resource programming.
- Added new demos specific to PIC24FJ256DA210:
 - Elevator Demo - This demo shows an elevator status monitor that indicates the current location of the elevator car.
 - RCCGPU-IPU Demo - formerly PIC24F_DA Demo. This demo shows how RCCGPU and IPU modules are used.
 - Color Depth Demo - This demo shows how 1bpp, 4bpp and 8 bpp color depths applications are implemented.
 - Remote Control Demo - This demo shows how a universal remote control can be implemented using RF4CE communication protocol. This demo also integrates the MRF24J40MA (a certified 2.4 GHz IEEE 802.15.4 radio transceiver module) for sending RF4CE messages to the paired device.
- Added driver for Solomon Systech 132x64 OLED/PLED Display Controller SSD1305

Changes:

- Modified Resistive Touch Screen calibration. Now the calibration stores the 8 touch points to support large touch panels.
- Modified 4-wire Resistive Touch Screen driver to support build time setting of single samples and auto-sampling of resistive touch inputs.
- Naming of internal and external resource files (files that defined fonts and images) in most demos are now standardized to use the same naming convention.
- Support for Graphics PICTail v2 (AC164127) is now discontinued.
- Merged "JPEG" demo and "Image Decoders" demo to "Image Decoder" demo.
- Graphics Resource Converter upgrade (Version 3.17.47) - refer to "Graphics Resource Converter Help.pdf" located in **<Install Directory>\Microchip Solutions\Microchip\Graphics\bin\grc** for details.
- External Memory Programmer upgrade (Version 1.00.01) - refer to "External Memory Programmer Help.pdf" located in **<Install Directory>\Microchip Solutions\Microchip\Graphics\bin\memory_programmer** for details.

Fixes:

- Fix PushRectangle() issue where one line of pixel is not being updated.
- Fix TextEntry widget issue where the string is not displayed when the allocated string buffer length is equal to the maximum string length set in the widget.
- Fonts maximum character height is now set to 2^16.

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE ([\[2\]](#))
- EXTDATA - replaced by GFX_EXTDATA ([\[3\]](#))
- BITMAP_FLASH - replaced by IMAGE_FLASH ([\[4\]](#))
- BITMAP_RAM - replaced by IMAGE_RAM ([\[5\]](#))
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA ([\[6\]](#))
- EEPROM.h and EEPROM.c are replaced by the following files:
 - MCHP25LC256.c - source code
 - MCHP25LC256.h - header file
- in the HardwareProfile.h add #define USE_MCHP25LC256 to use the new driver.

Migration Changes:

- For existing code that wants to use the new anti-aliased fonts or extended glyph features: regenerate the font tables using the "Graphics Resource Converter" with the check box for the required feature set to be enabled. For anti-aliased fonts,

add the macro #define USE_ANTIALIASED_FONTS ([🔗](#)) in the GraphicsConfig.h

Known Issues:

- Extended glyph for certain font (such as Thai) when used with Static text widget is clipped. Future version will add additional vertical text alignment to the static text widget
- Anti-aliasing and extended glyph features are not supported when using PIC24FJ256DA210 CHRGPU.
- SPI flash programming on the Epson S1D13517 PICtail board Rev 1.1 is not always reliable, the S1D13517 demo no longer uses external memory flash in the example.
- When using PIC24FJ256GB210 PIM with Explorer 15 board with a 5v Lumex LCD display, the S1D13517 demo does not run correctly.

v3.03 (v3.02)

New:

- Added custom video playback from SD Card in SSD1926 Demo. Video frames are formatted to RGB565 format.
- Added support for 1bpp, 4bpp and 8 bpp color depth on Chart ([🔗](#)) widget.
- Added support for Display Boards from Semitron
- Seiko 35QVW1T
- Seiko 43WVW1T

Changes:

- Maximum font height is now 256 pixels.
- Modified EditBox behavior
- Caret is now by default enabled.
- Caret can now be shown even if USE_FOCUS ([🔗](#)) is disabled.
- Applications can now respond to touchscreen event on EditBoxes when USE_FOCUS ([🔗](#)) is disabled.
- Modified resistive touchscreen calibration sequence.
- Graphics Resource Converter upgrade (Version 3.8.21) - refer to "Graphics Resource Converter Help.pdf" located in <Install Directory>\Microchip Solutions\Microchip\Graphics\bin\grc for details.
- External Memory Programmer upgrade (Version 1.00.01) - refer to "External Memory Programmer Help.pdf" located in <Install Directory>\Microchip Solutions\Microchip\Graphics\bin\memory_programmer for details.

Fixes:

- Fix Low Cost Controller display driver issue when run with Resistive Touch Screen driver that uses single samples.
- Fix issue on PIC24FJ256DA210 display driver PutImage ([🔗](#)())'s issue when using palette on 4 bpp and 1 bpp images.
- Fix issue on PIC24FJ256DA210 display driver PutImage ([🔗](#)())'s missing lines when the image last pixel row or column falls on the edge of the screen.
- Fix Resistive Touch Screen driver issue on rotated screens.
- Fix GetImageHeight ([🔗](#)()) GetImageWidth ([🔗](#)()) issues for images that are RLE compressed.
- EPMP module is now disabled when memory range defined for display buffer is located in internal memory.
- Add default color definitions in gfxcolors.h for 1bpp, 4bpp 8bpp and 16 bpp. Added back legacy colors.
- Fix HX8347 driver WritePixel() macro when using 16bit PMP mode.
- Fix PIC24FJ256DA210 display driver issue on source data (continuous and discontinuous data) when doing block copies of memory using RCCGPU.

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE ([🔗](#))
- EXTDATA - replaced by GFX_EXTDATA ([🔗](#))
- BITMAP_FLASH - replaced by IMAGE_FLASH ([🔗](#))
- BITMAP_RAM - replaced by IMAGE_RAM ([🔗](#))
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA ([🔗](#))

- EEPROM.h and EEPROM.c are replaced by the following files:
 - MCHP25LC256.c - source code
 - MCHP25LC256.h - header file
- in the HardwareProfile.h add #define USE_MCHP25LC256 to use the new driver.

Migration Changes:

- EditBox widget's caret behavior is now by default enabled when USE_FOCUS (✉) is set. To disable, ignore all messages for the edit box by returning a zero when in GOLMsgCallback (✉)().

Known Issues:

- PutImage (✉)() does not work when using PIC24FJ256DA210 and look up table is used on images located at EDS memory with color depth less than 8bpp.
- External Memory Programmer utility does not work with Graphics PICTail v2 (AC164127)
- When using PIC24FJ256GB210 PIM with Explorer 15 board with a 5v Lumex LCD display, the S1D13517 demo does not run correctly.
- Font tables are limited to 256 pixel character height.

v3.02

New:

- Added custom video playback from SD Card in SSD1926 Demo. Video frames are formatted to RGB565 format.
- Added support for 1bpp, 4bpp and 8 bpp color depth on Chart (✉) widget.
- Added support for Display Boards from Semitron
- Seiko 35QVW1T
- Seiko 43WVW1T

Changes:

- Maximum font height is now 256 pixels.
- Modified EditBox behavior
- Caret is now by default enabled.
- Caret can now be shown even if USE_FOCUS (✉) is disabled.
- Applications can now respond to touchscreen event on EditBoxes when USE_FOCUS (✉) is disabled.
- Modified resistive touchscreen calibration sequence.
- Graphics Resource Converter upgrade (Version 3.8.21) - refer to "Graphics Resource Converter Help.pdf" located in <Install Directory>\Microchip Solutions\Microchip\Graphics\bin\grc for details.
- External Memory Programmer upgrade (Version 1.00.01) - refer to "External Memory Programmer Help.pdf" located in <Install Directory>\Microchip Solutions\Microchip\Graphics\bin\memory_programmer for details.

Fixes:

- Fix issue on PIC24FJ256DA210 display driver PutImage (✉)'s missing lines when the image last pixel row or column falls on the edge of the screen.
- Fix Resistive Touch Screen driver issue on rotated screens.
- Fix GetImageHeight (✉)() GetImageWidth (✉)() issues for images that are RLE compressed.
- EPMP module is now disabled when memory range defined for display buffer is located in internal memory.
- Add default color definitions in gfxcolors.h for 1bpp, 4bpp 8bpp and 16 bpp. Added back legacy colors.
- Fix HX8347 driver WritePixel() macro when using 16bit PMP mode.
- Fix PIC24FJ256DA210 display driver issue on source data (continuous and discontinuous data) when doing block copies of memory using RCCGPU.

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE (✉)
- EXTDATA - replaced by GFX_EXTDATA (✉)

- BITMAP_FLASH - replaced by IMAGE_FLASH (█)
- BITMAP_RAM - replaced by IMAGE_RAM (█)
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA (█)
- EEPROM.h and EEPROM.c are replaced by the following files:
 - MCHP25LC256.c - source code
 - MCHP25LC256.h - header file
- in the HardwareProfile.h add #define USE_MCHP25LC256 to use the new driver.

Migration Changes:

- EditBox widget's caret behavior is now by default enabled when USE_FOCUS (█) is set. To disable, ignore all messages for the edit box by returning a zero when in GOLMsgCallback (█)().

Known Issues:

- PutImage (█)() does not work when using PIC24FJ256DA210 and look up table is used on images located at EDS memory with color depth less than 8bpp.
- External Memory Programmer utility does not work with Graphics PICTail v2 (AC164127)
- When using PIC24FJ256GB210 PIM with Explorer 15 board with a 5v Lumex LCD display, the S1D13517 demo does not run correctly.
- Font tables are limited to 256 pixel character height.

v3.01 (v3.00)

New:

- Graphics External Memory Programmer ported to java version.
- Two options to program boards:
 - UART option if the board supports UART interface
 - USB option if the board support USB device interface
- when installing the USB drivers for the programmer utility Use the drivers located in "<install directory>\Microchip\Utilities\USB Drivers\MPLABComm"
- For detailed usage, please refer to the External Programmer help file
- Added Analog Clock (█) widget.
- Added new driver Epson S1D13517 display driver with additional driver features:
 - Gradient
 - Alpha Blending
 - 16/24 bits per pixel (bpp)
- Added new Graphics PICTail Plus Epson S1D13517 Board (AC164127-7)
- Added new Graphics Display Truly 5.7" 640x480 Board (AC164127-8)
- Added new Graphics Display Truly 7" 800x480 Board (AC164127-9)
- Added 24bpp support.
- Added a specific PIC24FJ256DA210 demo, PIC24F_DA
- Graphics Resource Converter - refer to the Graphics Resource Converter help file for release note information.
- New PIC32MX Low-Cost Controllerless Graphics PICTail Board (AC164144)
- Added Run Length Encoding (RLE) compression for bitmap images.
- RLE4 - compression for 4-bit palette (16 color) images
- RLE8 - compression for 8-bit palette (256 color) images
- Added Transparency feature for PutImage (█)() functions in Primitive Layer. For Driver Layer, this feature is enabled in the following drivers:
- mchpGfxDrv - Microchip Graphics Controller Driver

- SSD1926 - Solomon Systech Display Controller Driver
- Added new demo for Graphics PICtail Plus Epson S1D13517 Board (S1D13517 Demo)
- Added AR1020 Resistive Touch Screen Controller beta support.
- Added support for MikroElektronika "mikroMMB for PIC24" board.
- Added DisplayBrightness (■)() function for display drivers that have an option to control the display back light.
- MPLAB X demo project support (BETA)
- Tested with MPLAB X Beta 6
- Each demo project contains multiple configuration schemes
- The graphics object layer uses a default scheme structure. If the application wishes to use a different default scheme, the application will need to define GFX_SCHEMEDEFAULT in GraphicsConfig header file.

Changes:

- Relocated all Graphics Demo projects under Graphics directory (<install directory>/Graphics).
- Works with Graphics Display Designer version 2.1
- Works with Graphics Resource Converter version 3.3
- Removed IPU decoding from the Primitive demo
- Removed ImageFileConverter application from Image Decoder demo
- Use Graphics Resource Converter to generate output for the demo.
- Shorten file name by using abbreviated names
- Refer to abbreviations.htm in the Microchip help directory for details
- Change the "Alternative Configurations" directory in the demos to "Configs"
- Changed the "Precompiled Demos (■)" directory in the demos to "Precompiled Hex"
- Combined all application note demos (AN1136, AN1182, AN1227 and AN1246) into one demo project (AppNotes).
- Modified External Memory and JPEG demos to include USB device mode to program external flash memory.
- Moved the location of the COLOR_DEPTH (■) setting from the HardwareProfile.h to the GraphicsConfig.h
- Removed USE_DRV_FUNCTIONNAME (example USE_DRV_LINE to implement the Line() function in the driver) option to implement Primitive Layer functions in the Driver Layer. The Primitive Layer functions are now modified to have weak attributes, so when the driver layer implements the same function, the one in the driver will be used at build time.
- Modified HardwareProfile.h for Graphics demo boards and development Platforms.
- When using Resistive Touch Screen: add macro USE_TOUCHSCREEN_RESISTIVE
- When using AR1020 as the touch screen controller: add macro USE_TOUCHSCREEN_AR1020
- When using SPI Flash Memory (SST25VF016) in Graphics Development Boards: add macro USE_SST25VF016
- When using Parallel Flash Memory (SST39LF400) in PIC24FJ256DA210 Development Board: add macro USE_SST39LF400
- When using Parallel Flash Memory (SST39LF400) in PIC24FJ256DA210 Development Board: add macro USE_SST39LF400
- Added function pointers for Non-Volatile Memories when used in the Demos (■).
- NVM_SECTORERASE_FUNC - function pointer to sector erase function.
- NVM_WRITE_FUNC - function pointer to write function.
- NVM_READ_FUNC - function pointer to read function.
- Display Driver Layer architecture is changed. Refer to Adding New Device Driver (■) for new requirements.
- Modified Resistive Touch Driver calibration
- In the PIC24FJ256DA210 driver, CopyWindow (■)() is modified to CopyBlock (■)().

Fixes:

- Fixed issue on vertical Progress Bar (■) rendering.

- Updated demos Google map and JPEG to use the proper GFX_RESOURCE (🔗) identifiers for JPEG resources
- HX8347 driver now compiles and works with 'mikroMMB for PIC24'
- Bug fixes in the digital meet and cross hair widgets
- Removed references to the PIC24 configuration bit COE_OFF.
- Fixed issue on PutImage (🔗)() when using PIC24FJ256DA210 and look up table is used on images located at internal or external SPI flash with color depth less than 8bpp.
- Fixed issue on Line() in the SSD1926 and mchpGfxDrv driver files. The stored coordinates after a successful rendering of a line will be at the end point (x2,y2).

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE (🔗)
- EXTDATA - replaced by GFX_EXTDATA (🔗)
- BITMAP_FLASH - replaced by IMAGE_FLASH (🔗)
- BITMAP_RAM - replaced by IMAGE_RAM (🔗)
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA (🔗)

Migration Changes:

- DisplayDriver.c is not used to select the display driver (the file is not a part of the graphics library release package).
- The application should include the driver(s) in the project. Multiple driver files can be included in one project because the hardware profile will define the driver and only that driver's source code will be used.
- For example, a project may be designed to use the Microchip's PIC24FJ256DA210 graphics controller and the SSD1926 depending on the hardware profile used. The project will include the following source files, SSD1926.c and mchpGfxDrv.c, among the graphics source files. The hardware profile will contain the following macros, GFX_USE_DISPLAY_CONTROLLER_SSD1926 (🔗) and GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (🔗), to select the SSD1926 and Microchip's PIC24FJ256DA210 graphics controller, respectively.
- When using PIC24FJ256DA210, the driver files are now changed to:
 - mchpGfxDrv.c - source code
 - mchpGfxDrv.h - header file
 - mchpGfxDrvBuffer.c - source code that declares display buffer, work areas and cache areas for IPU operations in EDS.
- The touch screen drivers in the "Board Support Package" directory are renamed to:
 - TouchScreenResistive.c - internal resistive touch source code
 - TouchScreenResistive.h - internal resistive touch header file
 - TouchScreenAR1020.c - external AR1020 touch source code
 - TouchScreenAR1020.h - external AR1020 touch header file
- The two original files (TouchScreen.c and TouchScreen.h) are still needed since they are defining common interfaces to resistive touch drivers.
- When using the internal resistive touch module, the project will contain TouchScreenResistive.c and TouchScreen.c. All modules that reference touch screen APIs will include TouchScreen.h header file.
- When using the external AR1020 touch module, the project will contain TouchScreenAR1020.c and TouchScreen.c. All modules that reference touch screen APIs will include TouchScreen.h header file.
- The touch screen initialization routine API has changed. The new TouchInit API has four parameters passed. Please refer to the API definition for more details.
- When using the Potentiometer on the graphics development boards, include in your project the following files found in the "Board Support Package" directory :
 - TouchScreenResistive.c - source code
 - TouchScreenResistive.h - header file
 - Potentiometer.h - contains the APIs for the A/D interface.
- EEPROM.h and EEPROM.c are going to be deprecated. Use the two new files:
- MCHP25LC256.c - source code

- MCHP25LC256.h - header file
- in the HardwareProfile.h add #define USE_MCHP25LC256 to use the new driver.
- A SPI driver has been created to support projects with multiple devices on one SPI channel.
- Projects will need to include the source file drv_spi.c in projects that use devices on a SPI channel.
- SPI Flash initialization routines will need to pass a DRV_SPI_INIT_DATA structure. This structure defines the SPI control and bit rate used by the SPI Flash module.
- The COLOR_DEPTH (■) macro has been moved from the hardware profile header file to the GraphicsConfig.h header file.
- For project migration please refer the graphics demos for examples.

Known Issues:

- PutImage (■)() does not work when using PIC24FJ256DA210 and look up table is used on images located at EDS memory with color depth less than 8bpp.
- PutImage (■)() of when using PIC24FJ256DA210 for 8bpp images is missing the last row and last column of the bitmap when the image is from external memory, look up table is used and the screen is rotated 90 degrees.
- When compiling the Analog Clock source code with C30 v3.24 the optimization setting must be set to 0 (none).
- External Memory Programmer utility does not work with Graphics PICTail v2 (AC164127)
- When using PIC24FJ256GB210 PIM with Explorer 15 board with a 5v Lumex LCD display, the S1D13517 demo does not run correctly.
- Font tables are limited to 256 pixel character height. For fonts generated for external memory, the maximum height limitation is 128 pixels.

v2.11

New:

- Graphics Resource Converter (GRC) ported to java version.
- Added support for Inflate Processing Unit (IPU) and Character Processing Unit (CHRGPU) of the Microchip Graphics Module implemented in PIC24FJ256DA210.
- Added new "Google Map Demo" for PIC32MX795F512L and PIC24FJ256DA210 device.
- Added SST39LF400 Parallel Flash Memory driver in "Board Support Package". This is the driver for the parallel flash on the "PIC24FJ256DA210 Development Board".
- Added demo support for PIC32 MultiMedia Expansion Board (DM320005).
- Added GFX_IMAGE_HEADER (■) structure. This structure defines how the image(s) are accessed and processed by the Graphics Library.
- Added a third option (#define XCHAR (■) unsigned char) on the XCHAR (■) usage. This is provided as an option to use characters with IDs above 127 and below 256. With this option, European fonts that uses characters with character IDs above 127 and below 256 can now be generated and used in the library.
- Added a scheme to replace the default font GOLFFontDefault (■) in the library with any user defined fonts. Refer to "Changing the default Font (■)" for details.

Changes:

- Added compile switches to all drivers in "Board Support Package" for options to compile out when not used in specific projects.
- Replaced TYPE_MEMORY with GFX_RESOURCE (■) type enumeration and expanded the enumeration for graphics resources (such as images and fonts). GFX_RESOURCE (■) type will determine the source and the data format of the resource (compressed or uncompressed).
- Changes on the macros used on the "Graphics JPEG Demo":

```
// Valid values of the first field for JPEG_FLASH and JPEG_EXTERNAL structures
#define FILE_JPEG_FLASH      2    // the JPEG file is located in internal flash
#define FILE_JPEG_EXTERNAL   3    // the JPEG file is located in external memory
to

// Valid values of the first field for JPEG_FLASH and JPEG_EXTERNAL structures
#define FILE_JPEG_FLASH      0    // the JPEG file is located in internal flash
```

```
#define FILE_JPEG_EXTERNAL 1 // the JPEG file is located in external memory
```

- Added function pointers to GOL structure OBJ_HEADER (§). These function pointers makes it easier to add user created objects in the Graphics Library.
- DRAW_FUNC (§) - function pointer to object drawing function.
- FREE_FUNC (§) - function pointer to object free function. Only for objects that needs free routines to effectively free up memory used by the object create function.
- MSG_FUNC (§) - function pointer to object message function.
- MSG_DEFAULT_FUNC (§) - function pointer to object default message function.
- Merged "Graphics External Memory Demo" and "Graphics External Memory Programmer" into one demo "Graphics External Memory Programmer and Demo".

Fixes:

- TouchScreen driver now checks display orientation and adjusts the touch to be aligned to the display orientation.
- Fixed GOLFocusNext() issue on list that does not contain an object that can be focused.
- Removed redundant code in GOLRedrawRec (§)().
- Added an option in XCHAR (§) to use unsigned char.

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE (§)
- EXTDATA - replaced by GFX_EXTDATA (§)
- BITMAP_FLASH - replaced by IMAGE_FLASH (§)
- BITMAP_RAM - replaced by IMAGE_RAM (§)
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA (§)

Migration Changes:

- To use drivers located in "Board Support Package" directory, add the USE_DRIVERNAME macro in application code (in the demos these are added in the HardwareProfile.h) to include the drivers. Refer to the specific driver code for the correct USE_DRIVERNAME macro name.
- The new version of the Graphics Resource Converter generates graphics application resources (fonts and images) using the new GFX_IMAGE_HEADER (§) structure for images and new GFX_RESOURCE (§) type defines to specify location of the resources. Because of this, some structures are deprecated and replaced with more appropriate structures. To remove the deprecation warnings, regenerate the fonts and images files using the new Graphics Resource Converter.

Known Issues:

- Graphics SSD1926 JPEG and SD Card Demo does not support Graphics Display Powertip 4.3" 480x272 Board (PH480272T_005_I11Q). As is, there's not enough spare memory space to carry out the hardware JPEG decoding operation by the SSD1926. A potential work around is to reduce the active display area size to reserve more memory space for the JPEG decoding operation.
- SSD1926 hardware acceleration for eclipse is disabled due to missing pixels at Angle 0.
- PIC32MX460 PIM (not Starter Kit) does not support 16-bit PMP mode with Graphics PICtail™ Plus Board Version 3 (SSD1926) Board. It only supports 8-bit PMP mode. This is due to pin mapping conflicts on the boards.
- This version of Graphics Library is not compatible with Graphics Display Designer v2.0.0.9c

v2.10

New:

- Added new demo "Graphics Object Layer Palette Demo" for PIC24FJ256DA210 device.
- Added support for PIC32MX795F512L device.
- Added documentation for the Grid (§) object.
- Added Vertical Mode to Progress Bar (§).
- Added MicrochipGraphicsModule display driver.
- Added "Board Support Package" directory. This contains common hardware drivers for Microchip demo boards.
- Added OBJ_MSG_PASSIVE as a translated message on the slider to detect a touch screen release message. This has

no effect on the state of the slider. Applications that does not qualify for touch press and touch move event must now qualify the messages for the slider object to avoid processing messages for touch release.

Changes:

- To improve speed modified gfpmp.c and gfpmp.c to be inline functions in gfpmp.h and gfpmp.h respectively.
- Changed HX8347A.c to HX8347.c (both the D and A driver version is implemented in the new file). To select set the DISPLAY_CONTROLLER to be HX8347A or HX8347D in the hardware profile.
- Modified malloc() and free() to be defined by macros in GraphicsConfig.h file. For applications using Operating System, the macros can be redefined to match the OS malloc and free functions. The default settings are:
- #define GFX_malloc (■)(size) malloc(size)
- #define GFX_free (■)(ptr) free(ptr)
- Merged GOL Demo English and Chinese demo into one demo.
- Removed the macro "GRAPHICS_HARDWARE_PLATFORM". This is specific to Microchip demo boards.
- Abstracted the timer from the touch screen driver.
- Moved the following hardware drivers to the "Board Support Package" directory
- Touch screen driver: TouchScreen.c and TouchScreen.h files.
- SPI Flash driver: SST25VF016.c and SST25VF016.h files.
- Graphics PICtail Version 2 Parallel Flash driver: SST39VF040.c and SST39VF040.h files.
- Explorer 16 SPI EEPROM Flash driver: EEPROM.c and EEPROM.h files.
- Graphics PICtail Version 2 Beeper driver: Beep.c and Beep.h files.
- Revised the Seiko 3.5" 320x240 display panel schematic to revision B. Corrected the pin numbering on the hirose connector. See "Schematic for Graphics Display Seiko 3.5in 320x240 Board Rev B.pdf" file on the \Microchip\Solutions\Microchip\Graphics\Documents\Schematics directory.

Fixes:

- Fixed TextEntry object issue on output string maximum length.
- Fixed Slider (■) increment/decrement issue on Keyboard messages.
- Fixed GOLGetFocusNext (■) bug when none of the objects in the list can be focused.

Migration Changes:

- pmp interface files are converted to header files and functions are now inline functions to speed up pmp operations. Projects must be modified to:
 - include gfpmp.h and gfpmp.h source files in the project.
 - gfpmp.c file is retained but will only contain the definition of the EPMP pmp_data.
- Converted the macro: #define GRAPHICS_HARDWARE_PLATFORM HARDWARE_PLATFORM where HARDWARE_PLATFORM is one of the supported hardware platforms defined in the section Graphics Hardware Platform to just simply #define HARDWARE_PLATFORM.
- Since the timer module is abstracted from the touch screen driver in the "Board Support Package", the timer or the module that calls for the sampling of the touch screen must be implemented in the application code. Call the function TouchProcessTouch() to sample the touch screen driver if the user has touched the touch screen or not.

Example:

```
// to indicate the hardware platform used is the
// Graphics PICtail™ Plus Board Version 3
#define GFX_PICTAIL_V3
```

- Projects which uses the following hardware drivers will need to use the latest version of the drivers located in the "Board Support Package" directory.
- Touch screen driver: TouchScreen.c and TouchScreen.h files.
- SPI Flash driver: SST25VF016.c and SST25VF016.h files.
- Graphics PICtail Version 2 Parallel Flash driver: SST39VF040.c and SST39VF040.h files.
- Explorer 16 SPI EEPROM Flash driver: EEPROM.c and EEPROM.h files.

- Graphics PICtail Version 2 Beeper driver: Beep.c and Beep.h files.
- In the TouchScreen driver, the timer initialization and timer interrupt sub-routine (ISR) are abstracted out of the driver. The initialization and the ISR should be defined in the application code. the TouchProcessTouch() function in the driver should be called in the ISR to process the touch.

Known Issues:

- Graphics SSD1926 JPEG and SD Card Demo does not support Graphics Display Powertip 4.3" 480x272 Board (PH480272T_005_I11Q). As is, there's not enough spare memory space to carry out the hardware JPEG decoding operation by the SSD1926. A potential work around is to reduce the active display area size to reserve more memory space for the JPEG decoding operation.
- SSD1926 hardware acceleration for eclipse is disabled due to missing pixels at Angle 0.
- PIC32MX460 PIM (not Starter Kit) does not support 16-bit PMP mode with Graphics PICtail™ Plus Board Version 3 (SSD1926) Board. It only supports 8-bit PMP mode. This is due to pin mapping conflicts on the boards.
- This version of Graphics Library is not compatible with Graphics Display Designer v2.0.0.9c

v2.01

Changes:

- Modified drivers for abstraction of pmp and epmp interfaces. they have a common header file DisplayDriverInterface.h.
- DisplayDriverInterface.h is added to the Graphics.h file.
- DelayMs() API is abstracted from the driver files. TimeDelay.c and TimeDelay.h is added.

Fixes:

- Fixed background color bug in StaticText and Digital Meter (█) object.
- Fixed ListBox LbSetFocusedItem (█)() bug on empty lists.
- Graphics SSD1926 JPEG and SD Card Demo is fixed to support SD card of size 2GB or bigger

Migration Changes:

- pmp interface is abstracted from the driver. Projects must be modified to:
- include gfxpmp.c and gfaxpmp.c source files in the project.
- DelayMs() is abstracted from the drivers.
- Add TimeDelay.c source file in the project.
- Add TimeDelay.h header file in the project.

v2.00

Changes:

- "Graphics PICtail Board Memory Programmer" has been renamed to "Graphics External Memory Programmer".
- "Bitmap & Font Converter" utility has been renamed to "Graphics Resource Converter".
- Font format has changed. The bit order has been reversed. Necessary for cross compatibility.
- Added 2 new directories in each demo
- Precompiled Demos (█) - this directory contains all pre-compiled demos for all hardware and PIC devices supported by the demo.
- Alternative Configurations - this directory contains all the Hardware Profiles for all hardware and PIC devices supported by the demo.
- Moved all hardware and display parameters from GraphicsConfig.h to HardwareProfile.h. HardwareProfile.h references a hardware profile file in "Alternative Configurations" directory based on the PIC device selected.

Fixes:

- Fixed BtnSetText (█)() bug when using Multi-Line Text in Buttons.
- Fixed SDSectorWrite() function in SSD1926_SDCard.c in the "Graphics SSD1926 JPEG and SD Card Demo".

Migration Changes:

- Move all hardware and display parameters from GraphicsConfig.h to HardwareProfile.h
- panel type, display controller, vertical and horizontal resolution, front and back porches, synchronization signal timing and

polarity settings etc.

- GRAPHICS_PICTAIL_VERSION,1,2,250,3 options are now deprecated, new usages are:
- #define GRAPHICS_HARDWARE_PLATFORM GFX_PICTAIL_V1
- #define GRAPHICS_HARDWARE_PLATFORM GFX_PICTAIL_V2
- #define GRAPHICS_HARDWARE_PLATFORM GFX_PICTAIL_V3 (█)
- The font format has changed, run Graphics Resource Converter to regenerate font files, the bit order is reversed. No legacy support is provided. Primitive/Driver layers now expects the new format.
- The initialization sequence of GOLInit (█)() relative to the flash memory initialization is sensitive due to the sharing of hardware resources, i.e. SPI or PMP. Care should be taken to make sure the peripheral and I/O port settings are correct when accessing different devices.
- A number of configuration options have been moved from GraphicsConfig.h to HardwareProfile.h, this is required to maintain a more logical flow.
- HardwareProfile.h now points to one of many Alternative Configuration files, each one specific to a certain hardware board setup.
- DelayMs routine in SH1101A-SSD1303.c/h is now a private function, no public API is exposed. In future releases, DelayMS will be removed from all drivers and be replaced by an independent module.
- GenericTypeDefs.h has been updated with new definitions, this should not impact any legacy codes.

v1.75b

Changes:

- None.

Fixes:

- Fixed Line2D() bug in SSD1926.c.
- Fixed remainder error in JPEG decoding in JpegDecoder.c.
- Fixed pinout labels for reference design schematic "Schematic for Graphics Display Powertip 4.3in 480x272 Board Rev 2.pdf".

Migration Changes:

- None.

v1.75 Release (July 10, 2009)

Changes:

- Added 2D acceleration support for controllers with accelerated primitive drawing functions.
- Added Digital Meter (█) Widget for fast display refresh using fixed width fonts.
- Added support for selected PIC24H Family of devices.
- Added support for selected dsPIC33 Family of devices.
- Updated all primitive functions to return success or fail status when executed. This is used to evaluate if the primitive function has finished rendering.
- Updated Solomon Systech SSD1926 driver to use 2D accelerated functions.
- New Display Controller Driver supported:
 - Ilitek ILI9320
 - Solomon Systech SSD1289
 - Himax HX8347
 - Renesas R61580
- New demos are added:
 - PIC24F Starter Kit Demo
 - PIC24H Starter Kit Demo 1
 - Graphics JPEG Demo using internal and external flash memory for image storage

- Graphics SSD1926 JPEG and SD Card Demo using SD Card for image storage
- Added JPEG support to "Font and Bitmap Converter Utility".
- Modified Button (■) Widget for new options:
- Use multi-line text (set USE_BUTTON_MULTI_LINE (■))
- Detect continuous touch screen press event using messaging
- Added Touch Screen event EVENT_STILLPRESS to support continuous press detection.
- New reference design schematics added:
- Schematic for Graphics Display DisplayTech 3.2in 240x320 Board.pdf
- Schematic for Graphics Display Newhaven 2.2in 240x320 with HX8347.pdf
- Schematic for Graphics Display Seiko 3.5in 320x240 Board.pdf
- Schematic for Graphics Displays DisplayTech and Truly 3.2in 240x320 with SSD1289.pdf
- Schematic for ILI9320.pdf

Fixes:

- Fixed dimension calculation for quarter sized meter.

Migration Changes:

- When using accelerated primitive functions while USE_NONBLOCKING_CONFIG (■) is enabled, the accelerated primitive must be checked if it successfully rendered. Refer to the coding example (■) for details.
- Replaced LGDP4531_R61505_S6D0129_S6D0139_SPFD5408.c and LGDP4531_R61505_S6D0129_S6D0139_SPFD5408.h files with drvTFT001.c and drvTFT001.h respectively.

v1.65 Release (March 13, 2009)

Changes:

- Added support for the new Graphics PICtail™ Plus Daughter Board (AC164127-3). This new board comes in two components: the controller board and the display board. The display board uses RGB type displays driven by the controller board. This configuration allows easy replacement of the display glass.
- Added application note AN1246 "How to Create Widgets".
- New Display Controller Driver supported
- UltraChip UC1610
- New demos are added
- Graphics AN1246 Demo showing the TextEntry Widget.
- Graphics Multi-App Demo showing USB HID, USB MSD and SD MSD demos using the Microchip Graphics Library.
- Modified Meter (■) Widget for new options:
- Set Title and Value Font at creation time
- Added GOLGetFocusPrev (■)() for focus control on GOL Objects.
- Added work spaces to demo releases.
- Graphics PICtail™ Plus Board 1 is obsolete. All references to this board is removed from documentation.
- New reference design schematics added:
- Schematic for Graphics Display Ample 5.7in 320x240 Board Rev A.pdf
- Schematic for Graphics Display Powertip 3.5in 320x240 Board Rev B.pdf
- Schematic for Graphics Display Powertip 4.3in 480x272 Board Rev B.pdf
- Schematic for Graphics Display Truly 3.5in 320x240 Board Rev A.pdf
- Schematic for Truly TOD9M0043.pdf

Fixes:

- Fixed drawing bug on TextEntry Widget
- Added missing documentation on Chart (■) and Text Entry (■) Widgets

Migration Changes:

- none

v1.60 Release (December 3, 2008)

Changes:

- Added TextEntry Widget.
- Modified Meter (■) Widget for new options:
 - Define different fonts for value and title displayed.
 - Add option for resolution of one decimal point when displaying values.
 - Add color options to all six arcs of the Meter (■).
- Meter (■) range is not defined by a minimum value and a maximum value.
- Added feature to a the Button (■) Widget to allow cancelling of press by moving away the touch and releasing from the Button (■)'s face.
- Added font sizes options of 3,4,5,6 & 7 in Font & Bitmap Converter Utility when converting fonts from TTF files.
- Enhanced the architecture of the Display Device Driver Layer.

Fixes:

- Fixed Font & Bitmap Converter Utility generation of reference strings to be set to const section.
- Fixed panel rendering to always draw the panel face color even if bitmaps are present.

Migration Changes:

- Added the following files in the Display Device Driver Layer to easily switch from one display driver to another.
 - DisplayDriver.h
 - DisplayDriver.c
- Modified implementation of GraphicsConfig.h file to support new Display Device Driver Layer architecture.
- Moved the definitions of pins used in device drivers implemented in all the demos to HardwareProfile.h file.
- EEPROM Driver
- Touch Screen Driver
- Beeper Driver
- Flash Memory Driver
- Display Drivers
- Modified GOL.c and GOL.h to include processing of TextEntry object when enabled by application.

v1.52 Release (August 29, 2008)

Changes:

- Added Chart (■) Widget.
- Added Property State for Window (■) Widget. Text in Title Area can now be centered.
- Added Supplementary Library for Image Decoders.
- Added documentation of Default Actions on widgets.
- Replaced USE_MONOCHROME compile switch with COLOR_DEPTH (■) to define color depth of the display.
- Added GOL_EMBOSS_SIZE (■) to be user defined in GraphicsConfig.h.
- Simplified initialization code for SSD1906 driver.

Fixes:

- Fixed touch screen algorithm.
- Fixed file path error in Font & Bitmap Converter Utility.

Migration Changes:

- USE_GOL (■) must be defined when using any Widgets.

- New include directory paths are added to demo projects:
- ..\..\..\Your Project Directory Name
- ..\..\Include
- Moved all driver files to new directory
- C files from ..\Microchip\Graphics to ..\Microchip\Graphics\Drivers
- GOL_EMBOSS_SIZE (█) can now be defined by the user in GraphicsConfig.h. If user does not define this in GraphicsConfig.h the default value in GOL.h is used.

v1.4 Release (April 18, 2008)

Changes:

- Added full support for PIC32 families.
- Added Application Note demo on fonts.
- Added images for end designs using PIC devices.

Fixes:

- Fixed GetPixel (█) error in SSD1906 Driver.
- Fixed SST39VF040 parallel flash driver reading instability.
- Fixed error in 64Kbytes rollover in utility conversion of bitmaps and SST39VF040 parallel flash driver.
- Fixed milli-second delay on PIC32.
- Fixed compile time option errors for PIC32.
- Fixed Graphics Object Layer Demo error in PIC32 when time and date are set.
- Fixed Picture (█) widget bug in detecting touchscreen in translating messages.

v1.3 Release (March 07, 2008)

Changes:

- none

Fixes:

- Fixed an inaccurate ADC reading problem with some Explorer 16 Development Boards that uses 5V LCD display.
- Editbox allocation of memory for text is corrected.
- Fix slider SetRange() bug.
- Set PIC32 configuration bits related to PLL to correct values.
- Touch screen portrait mode bug is fixed.

v1.2 Release (February 15, 2008)

Changes:

- Added support for Graphics PICtail Plus Board Version 2
- Added support for foreign language fonts
- Version 1.2 of the font and bitmap utility
- Support for multi-language scripts
- Support for generating font table from installed fonts
- Support to reduce font table size by removing unused characters
- Support to select between C30 and C32 compiler when generating bitmaps and font tables.
- Added Chinese version of Graphics Object Layer Demo.
- New Display Controller Drivers supported
- Solomon Systech SSD1906
- Orise Technology SPDF5408
- Replaced USE_UNICODE compile switch to USE_MULTIBYTECHAR (█) compile switch to define XCHAR (█) as 2-byte

character.

- Added compile switches
- GRAPHICS_PICTAIL_VERSION - sets the PICtail board version being used.
- USE_MONOCHROME – to enable monochrome mode.
- USE_PORTRAIT - to enable the portrait mode of the display without changing the display driver files.
- Added beta support for PIC32 device

Fixes:

- Specification changes to List Box widget. Bitmap is added to List Box items.
- Fixed List Box LbDellItemsList () error in not resetting item pointer to NULL when items are removed.
- Editbox allocation of memory for text is corrected.
- Static Text multi-byte character failure is fixed.
- Bar () function erroneous call to MoveTo () is removed since it causes the drawing cursor to be displaced.
- Removed SCREEN_HOR_SIZE and SCREEN_VER_SIZE macros from documentation. Maximum X and Y sizes are to be obtained by GetMaxX () and GetMaxY () macros.

v1.0 Release (November 1, 2007)

Changes:

- Edit Box, List Box, Meter (), Dial () widgets are added.
- Button () is modified. New options for the object are added.
- Modified OBJ_REMOVE definition to OBJ_HIDE (example BTN_REMOVE to BTN_HIDE ()).
- Modified GOLPanelDraw () function to include rounded panels.
- External memory support for the fonts and bitmaps is implemented.
- SSD1339 and LGDP4531 controllers support is added.
- Bevel (), FillBevel () and Arc () functions are added.
- Modified Graphics Object Layer Demo.
- Added Graphics External Memory Demo & Graphics PICtailTM Board Memory Programmer Demo.
- Added Graphics Application Note (AN1136- How to use widgets.) Demo.

Fixes: none

v0.93 Beta release (August 29, 2007)

Changes: none

Fixes:

- In demo code the bitmap images couldn't be compiled without optimization. bmp2c.exe utility output is changed to fix this bug.
- In demo code "volatile" is added for global variables used in ISRs.

v0.92 Beta release (July 25, 2007)

Changes:

- Keyboard and side buttons support is added.
- Keyboard focus support is added.
- PutImage () parameters are changed. Instead of pointer to the bitmap image the pointer to BITMAP_FLASH structure must be passed.
- GOLSuspend() and GOLResume() are removed.
- GOLMsg () doesn't check object drawing status. It should be called if GOL drawing is completed.
- GOLStartNewList() is replaced with GOLNewList ().
- Line() function calls are replaced with Bar () function calls for vertical and horizontal lines.

- Parameter “change” is removed for SldIncVal() and SldDecVal() .
- Some optimization and cleanup.
- Slider (■) API is changed:
- SldSetVal() is changed to SldSetPos (■)()
- SldGetVal() is changed to SldGetPos (■)()
- SldIncVal() is changed to SldIncPos (■)()
- SldDecVal() is changed to SldDecPos (■)()
- SldCreate (■)() input parameter are changed:
- “delta” changed to “res”

Fixes:

- PutImage (■)().
- Line().
- FillCircle (■)().
- For vertical slider the relation between thumb location and slider position is changed. For position = 0 thumb will be located at the bottom. For position = range it will be at the top.

v0.9 Beta release (July 06, 2007)

Changes:

- Background color support is removed.
- Non-blocking configuration for graphics primitives is added.
- GetImageWidth (■)(), GetImageHeight (■)() are added.
- OutText (■)(), OutTextXY (■)() and GetTextWidth (■)() functions are terminated by control characters (< 32).
- Graphics Objects Layer (GOL) is added.
- Button (■), Slider (■), Checkbox (■), Radio Button (■), Static Text, Picture (■) control, Progress Bar (■), Window (■), Group Box are implemented.
- Touch screen support is added.

Fixes:

- ReadFlashByte().
- GetTextWidth (■)().

v0.1 (June 05, 2007)

Changes:

- Initial release includes driver and graphic primitive layers only.
- Only driver for Samsung S6D0129 controller is available.

Fixes:

- None.

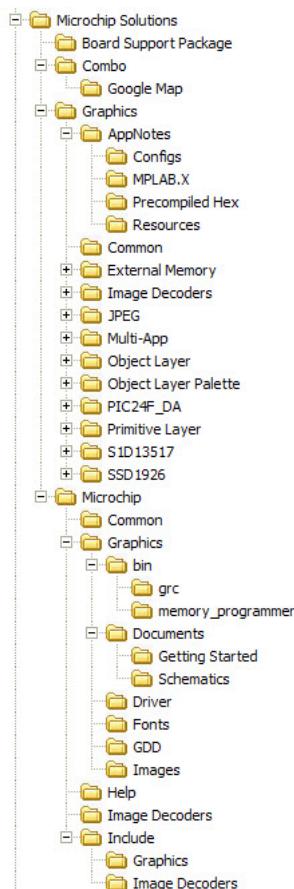
Known Issues

None

3 Getting Started

Directory Structure

The Microchip Graphics Library installation follows the standard directory structure for all Microchip library solutions. Installing the library will give the following structure:



One of the demo subdirectories (example: Graphics or Combo) may become "*Your Applications Directory*" that will contain your application source code. You can add code and modules here that will use and interact with the library. The library specific folders are the following:

- The **Microchip** folder will contain the library components.
- The **Help** sub-folder under **Microchip** folder will contain this document (**Graphics Library Help.chm** file).
- The **Graphics** sub-folder under the **Microchip** folder is where the C files, documentation and utilities are located.
- Inside this **Graphics** sub-folder are the directories for the **Drivers**, **Documents**, **GDD**, **Images** and **bin** directories. It will also contain the directory for the **Image Decoders** source files.
- The **GDD** (Graphics Display Designer) directory contains the GDD project template. Use this to start projects using the Graphics Display Designer.
- The **bin** directory contains the Graphics Resource Converter utility and External Memory Programmer both implemented in java.
- The **Include** sub-folder under the **Microchip** folder will contain common header files to all Microchip library solutions.
- Another **Graphics** directory is included in the **Include** sub folder. This will hold the Graphics Library header files as well as the header files for the **Image Decoders**.
- The **Board Support Package** folder will contain hardware specific drivers that are common to the Microchip Demo

Boards (such as Explorer 16, display panels or PICtail™ Plus Daughter Boards).

All subdirectories and files under the **Microchip** directory should not be modified. In case your project will use more than one Microchip library solution, this directory will contain all the library files you install. Thus, it is important to maintain the files in this directory. The **Microchip Solutions** directory may become your "MyProjects" directory that will contain all your projects using the different Microchip solutions.

How to Get Started

There are various ways to get started with Microchip Graphics Library:

1. Obtain Development Boards from the "Getting Started" section of the Microchip graphics website (www.microchip.com/graphics):
 1. Explorer 16 Starter Kit (DV164003) with any of the Graphics PICtail™ Plus Daughter Boards.
 2. PIC24FJ256DA210 Development Board (DV164039) with any of the individual Graphics Display Boards.
 3. A PIC32 Starter Kit and Graphics LCD Controller PICtail™ Plus SSD1926 Board (AC164127-5) with any of the individual Graphics Display Boards.
 4. A PIC32 Starter Kit and Graphics PICtail Plus Epson S1D13517 Board (AC164127-7) with any of the individual Graphics Display Boards.
 5. A PIC32 Starter Kit and Multi-Media Expansion Board (DM320005).
2. Graphics PICtail™ Plus Daughter Board available:
 - AC164127-3 - Graphics PICtail Plus Daughter Board with Truly 3.2" Display Kit



- AC164127-5 - Graphics LCD Controller PICtail Plus SSD1926 Board. This board is the same board used in AC164127-3 PICtail Plus and Display Panel combo shown above.



- AC164127-7 - Graphics PICtail Plus Epson S1D13517 Board.



- AC164144 - Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board.



- 3. Graphics Display Boards available:

- AC164127-4 - Graphics Display Truly 3.2" 240x320 Board



- AC164127-6 - Graphics Display Powertip 4.3" 480x272 Board



- AC164127-8 - Graphics Display Truly 5.7" 640x480 Board



- AC164127-9 - Graphics Display Truly 7" 800x480 Board



- AC164139 - Graphics Display Prototype Board



4. Refer to Web Seminar 4 on "Microchip Graphics Library Architecture" from the "Training and Support" section for an overview of the structure and the different layers of the library. It also gives a brief information on how to use the library.
5. Refer to Microchip's Regional Training Center class on Graphics Library:
 - [HIF 2131 – Designing with Microchip Graphics Library](#)
6. For a much detailed look on the usage, you can refer to the following application notes from the "Training and Support" section.
 - [AN1136 How to Use Widgets in Microchip Graphics Library](#). This application note introduces the basic functions needed to create and manage Widgets.
 - [AN1182 Fonts in the Microchip Graphics Library](#). This application note describes the format of the Microchip Graphics Library's font image. It also tells how to reduce the number of characters in a font and automate the creation of the character arrays referring to an application's strings.
 - [AN1227 Using a Keyboard with the Microchip Graphics Library](#). This application note describes how to implement a keyboard-based GUI.
 - [AN1246 How to Create Widgets in Microchip Graphics Library](#). This application note serves as a useful guide in creating customized Widgets. The essential components of a Widget are enumerated and described in this document. This application note also outlines the process of integrating the new Widget into the Graphics Library in order to utilize the already implemented routines for processing messages and rendering Widgets.
 - [AN1368 Developing Graphics Applications using PIC MCUs with Integrated Graphics Controller](#). This application note is

intended for engineers who are designing their first graphic application. It describes the basic definitions and jargons of graphics applications and it helps the engineer to understand the theory, necessary decision factors, hardware considerations, available microcontrollers and development tools.

- Finally, you can obtain the free licensed Microchip Graphics library also from the "Getting Started" section.



How to Build Projects for the PIC24FJ256DA210 Development Board

- In the application specific HardwareProfile.h file of your project set the hardware platform to PIC24FJ256DA210 Development Board:

```
#define PIC24FJ256DA210_DEV_BOARD
```

- In the same application specific HardwareProfile.h file of your project, set the correct display controller and the display panel combination. Selecting the correct display panel will choose the correct parameter settings for the display. Examples (■) of these parameters are horizontal and vertical resolution, display orientation, vertical and horizontal pulse width, and front and back porch settings.

- When using the Truly 3.2" display on AC164127-4 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210
// set the display panel
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
```

- When using the Powertip 4.3" display on AC164127-6 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210
// set the display panel
#define GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q
```

- In the same application specific HardwareProfile.h file of your project, set following (Refer to each demo hardware profiles for examples):

```
// set the PMP interface
#define USE_16BIT_PMP
// set the Graphics Clock Divider that generates the Pixel clock.
// Refer to display data sheet for pixel clock frequency requirement
// and Family Reference Manual - Oscillator for details on GCLK divider (GCLKDIV).
#define GFX_GCLK_DIVIDER 38
// set the display buffer start address.
#define GFX_DISPLAY_BUFFER_START_ADDRESS 0x00020000ul
// set the EPMP CS1 base address if using external memory on EPMP CS 1 space and its size
#define GFX_EPMP_CS1_BASE_ADDRESS 0x00020000ul
#define GFX_EPMP_CS1_MEMORY_SIZE 0x40000ul
// set the EPMP CS2 base address if using external memory on EPMP CS 1 space and its size
#define GFX_EPMP_CS2_BASE_ADDRESS 0x00080000ul
#define GFX_EPMP_CS2_MEMORY_SIZE 0x80000ul
```

- In the project's GraphicsConfig.h set the color depth to the desired bpp value (Refer to each demo hardware profiles for examples):

```
// set the color depth used
#define COLOR_DEPTH 16
```



How to Build Projects for Graphics PICtail™ Plus Board Version 3:

1. In the application specific HardwareProfile.h file of your project set the hardware platform to Graphics PICtail™ Plus Board Version 3:

```
#define GFX_PICTAIL_V3
```

2. In the same application specific HardwareProfile.h file of your project, set the correct display controller and the display panel combination. Selecting the correct display panel will choose the correct parameter settings for the display. Examples (■) of these parameters are horizontal and vertical resolution, display orientation, vertical and horizontal pulse width, and front and back porch settings.

- When using the Truly 3.2" display on AC164127-4 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_SSD1926
// set the display panel
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
```

- When using the Powertip 4.3" display on AC164127-6 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_SSD1926
// set the display panel
#define GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q
```

3. In the same application specific HardwareProfile.h file of your project, set following (Refer to each demo hardware profiles for examples):

```
// set the hardware platform
#define EXPLORER_16
// set the PMP interface
#define USE_8BIT_PMP
```

4. In the project's GraphicsConfig.h set the color depth to the desired bpp value (Refer to each demo hardware profiles for examples):

```
// set the color depth used
#define COLOR_DEPTH 16
```



How to Build Projects for Graphics PICtail™ Plus Epson S1D13517 Board

1. In the application specific HardwareProfile.h file of your project set the hardware platform to Graphics PICtail™ Plus Epson S1D13517 Board:

```
#define GFX_PICTAIL_V3E
```

2. In the same application specific HardwareProfile.h file of your project, set the correct display controller and the display panel combination. Selecting the correct display panel will choose the correct parameter settings for the display. Examples (■) of these parameters are horizontal and vertical resolution, display orientation, vertical and horizontal pulse width, and front and back porch settings.

- When using the Truly 3.2" display on AC164127-4 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_S1D13517
// set the display panel
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
```

- When using the Powertip 4.3" display on AC164127-6 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_S1D13517
// set the display panel
#define GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q
```

3. In the same application specific HardwareProfile.h file of your project, set following (Refer to each demo hardware profiles for examples):

```
// set the hardware platform
#define EXPLORER_16
// set the PMP interface
#define USE_8BIT_PMP
// set the display panel
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
```

- In the project's GraphicsConfig.h set the color depth to the desired bpp value (Refer to each demo hardware profiles for examples):

```
// set the color depth used
#define COLOR_DEPTH 16
```



Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board:

- In the application specific HardwareProfile.h file of your project set the hardware platform to Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board:

```
#define GFX_PICTAIL_LCC
```

- In the same application specific HardwareProfile.h file of your project, set the correct display controller and the display panel combination. Selecting the correct display panel will choose the correct parameter settings for the display. Examples (■) of these parameters are horizontal and vertical resolution, display orientation, vertical and horizontal pulse width, and front and back porch settings.

- When using the Truly 3.2" display on AC164127-4 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_DMA
// set the display panel
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
```

- When using the Powertip 4.3" display on AC164127-6 board

```
// set the display controller
#define GFX_USE_DISPLAY_CONTROLLER_DMA
// set the display panel
#define GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q
```

- In the same application specific HardwareProfile.h file of your project, set following (Refer to each demo hardware profiles for examples):

```
// set the hardware platform
#define EXPLORER_16 // or you can use PIC_SK if using Starter Kits
// set the PMP interface
#define USE_8BIT_PMP
```

- In the project's GraphicsConfig.h set the color depth to the desired bpp value (Refer to each demo hardware profiles for examples):

```
// set the color depth used
#define COLOR_DEPTH 16
```



How to Build Projects for the Multimedia Expansion Board

- In the application specific HardwareProfile.h file of your project set the hardware platform to Graphics PICtail™ Plus Epson S1D13517 Board:

```
#define MEB_BOARD
```

2. In the application specific HardwareProfile.h file of your project, set the correct display controller.

```
#define GFX_USE_DISPLAY_CONTROLLER_SSD1926
```

3. In the the same application specific HardwareProfile.h file of your project, set following (Refer to each demo hardware profiles for examples):

```
// set the PMP interface
#define USE_8BIT_PMP
// set the Starter Kit used
#define PIC32_GP_SK // use generic PIC32 Starter Kit
or
#define PIC32_USB_SK // use PIC32 USB Starter Kit
or
#define PIC32_ETH_SK // use PIC32 Ethernet Starter Kit
```

4. In the the project's GraphicsConfig.h set the color depth to the desired bpp value (Refer to each demo hardware profiles for examples):

```
// set the color depth used
#define COLOR_DEPTH 16
```

Demo Projects

The Microchip Graphics Library documentation has several components that covers installation, customization and usage of the library. Several demo projects are included in the installation to help you get started. Detailed information on each demo project is available from the "Getting Started" help file located in each of the demo folders.

Schematics

The library installation also includes schematics of currently supported controllers and glass. These can be found in the `../<install directory>/Microchip/Graphics/Documents/Schematics` directory.

- Schematic for Graphics Display Ampire 5.7in 320x240 Board Rev A.pdf
- Schematic for Graphics Display Powertip 3.5in 320x240 Board Rev B.pdf
- Schematic for Graphics Display Powertip 4.3in 480x272 Board Rev B.pdf
- Schematic for Graphics Display Truly 3.2in 240x320 Board Rev 4.pdf
- Schematic for Graphics Display Truly 3.5in 320x240 Board Rev A.pdf
- Schematic for Graphics LCD Controller PICtail SSD1926 Board Rev 2.pdf
- Schematic for Solomon Systech SSD1906.pdf
- Schematic for Truly GG1N1291UTSW-W-TP-E.pdf
- Schematic for Truly TFT-G240320UTSW-92W-TP.pdf
- Schematic for Truly TOD9M0043.pdf
- Schematic for Microtips MTF-T022BHNLP.pdf
- Schematic for Densitron TSR67802.pdf
- Schematic for Graphics Display DisplayTech 3.2in 240x320 Board.pdf
- Schematic for Graphics Display Newhaven 2.2in 240x320 with HX8347.pdf
- Schematic for Graphics Display Seiko 3.5in 320x240 Board.pdf
- Schematic for Graphics Displays DisplayTech and Truly 3.2in 240x320 with SSD1289.pdf
- Schematic for ILI9320.pdf
- Schematic for Graphics Display Prototype Board Rev 1.pdf
- Schematic for Graphics Display Truly 5.7in 640x480 Board Rev 2.pdf
- Schematic for Graphics Display Truly 7in 800x480 Board Rev 2.pdf

- Schematic for Graphics LCD Controller PICTail Plus S1D13517 Rev 1.1.pdf
- Schematic for Low-Cost Controllerless (LCC) Graphics Board Rev 1.pdf
- Schematic for PIC24FJ256DA210 Development Board Rev 1.1.pdf

Images

The library allows displaying 1bpp, 4bpp, 8bpp, 16bpp and 24bpp images. They can be located in program flash space or external memory. To convert the bitmap format file (BMP extension) or JPEG format file into source C file containing data array for internal memory or Intel hex file for external memory the “**Graphics Resource Converter**” included in the library installation can be used. Refer to the utility help file for details on usage.

Fonts

The library operates with 8-bit character encoded strings. It covers languages defined in ISO 8859 standards. East Asian and any other languages support is available for UNICODE encoded fonts. Font can be stored in internal flash as an array in const section (this limits font image size by 32Kbytes) or can be located in external memory. To convert the font file into source C file containing data array for internal memory or Intel hex file for external memory the “**Graphics Resource Converter**” utility can be used. The utility allows importing raster font files (FNT extension) or true font files (TTF extension). Refer to the help built in the utility for details. Raster font files can be extracted from MS Windows bitmap font package file (FNT extension) or converted from true type font file (TTF extension) with a third party font editor. One such freeware editor Fony is available at <http://hukka.furtopia.org/>. Another example is <http://fontforge.sourceforge.net>.

The utility also allows reducing the generated fonts to include only the characters that the application will use. This can be done by using font filtering. Please refer to application note "AN1182: Fonts in the Microchip Graphics Library" from the "Training and Support" section of the Microchip graphics website for details of implementing reduced fonts.

How to use the API Documentation

This help file includes the API description of the library. The way the API is structured is similar to the library layers.

1. The Device Driver Layer presents all the API included to initialize and use the display controller and the glass. This section also contains information on how to add new Device Driver.
2. The Graphics Primitive Layer is a hardware independent layer that contains the API for basic rendering functions. Use this section to render basic shapes like lines, rectangles, filled circle etc.
3. The Graphics Object Layer contains the API specific to each Widget type. Use this section to create and manage Widgets (■) as well as pages or screens of different Widgets (■). Messaging and rendering of Widgets (■) are also included in this section.

Updates and News

Refer to the Microchip graphics website www.microchip.com/graphics for the latest version of the Microchip Graphics Library, webinars, application notes, FAQs and latest news and updates.

4 Demo Projects

Summary of demo projects that comes with the installation of the Microchip Graphics Library.

4.1 Demo Summary

This is the current list of demo projects released with the Graphics Library.

Description

Demo Name	Description (see Note)
Primitive Layer	Shows how primitive functions are used.
Object Layer	Shows how objects are used with user messages interacting with objects.
Object Layer Palette	The same as the Object Layer demo but uses the Color Look Up Table of the Microchip Graphics Module.
External Memory	A simple demo showing how images and fonts from external memory are used in a graphics application.
Multi-App	A demo showing USB Framework, Memory Disk Drive, Image Decoders and Graphics libraries and stacks are integrated into one application.
PIC32 LCC	A demo showing how the Graphics LCD Controller PICtail™ Plus LCC Board (AC164144) can be connected to either the Explorer 16 board (with a PIC32 PIM) or connected to a PIC32 Starter Kits.
SSD1926	A demo showing the Solomon Systech Controller JPEG decoder module on the Graphics LCD Controller PICtail™ Plus SSD1926 Board (AC164127-5) and Multi-Media Expansion Board (DM320005) displaying JPEG images from an SD Card.
S1D13517	A demo showing the different features of the Epson S1D13517 Controller in the Graphics PICtail Plus Epson S1D13517 Board (AC164127-7)
E-Paper Epson	A demo showing how the E-paper Display PICtail™ Plus Board with Epson Controller (Epson P/N S5U13522C100S00) can be used with the Microchip Graphics Library.
Color Depth	Demo showing how the graphics module in PIC24FJ256DA210 is used in applications using color depths of 1, 4 and 8 BPP. (Location: <install_dir>/Graphics/PIC24F DA/Color Depth)
Elevator	A mock up of Elevator Monitor showing the location of the elevator car and the direction it is moving. (Location: <install_dir>/Graphics/PIC24F DA/Elevator)
RCCGPU-IPU	A demo showing how Rectangle (▣) Copy Graphical Processing Unit (RCCGPU) and Inflate Processing Unit (IPU) of the Graphics Module in PIC24FJ256DA10 is used. (Location: <install_dir>/Graphics/PIC24F DA/RCCGPU-IPU)
Remote Control	This is a demo showing how to create a universal remote control device with Graphical Interface using RF4CE protocol. (Location: <install_dir>/Combo/Remote Control)

AppNotes	A collection of application notes demo - Demo for Application Note AN1136 - Demo for Application Note AN1182 - Demo for Application Note AN1227 - Demo for Application Note AN1246
Image Decoders	A demo showing the Image Decoder library rendering bitmaps and jpeg images.
PIC24F Starter Kit	Demo for the PIC24F Starter Kit. (Location: <install_dir>/PIC24F Starter Kit)
PIC24H Starter Kit	Demo for the PIC24H Starter Kit. (Location: <install_dir>/PIC24H Starter Kit)
Google Map	A demo showing the TCPIP Stack and Graphics Library combined in an application. (Location: <install_dir>/Combo/Google Map)

Note:

Unless otherwise specified, the demos are located in <install_dir>/Graphics.

where: install_dir - is the directory location of the MLA installation.

4.2 Microchip Application Library Abbreviations

Summary of Microchip Applications Library Abbreviations used.

Description

Microchip Application Library Configuration File and Project Name Abbreviations. A summary of the abbreviations used can be found at <Install Directory>/Microchip/Help/Abbreviations.htm

4.3 Demo Compatibility Matrix

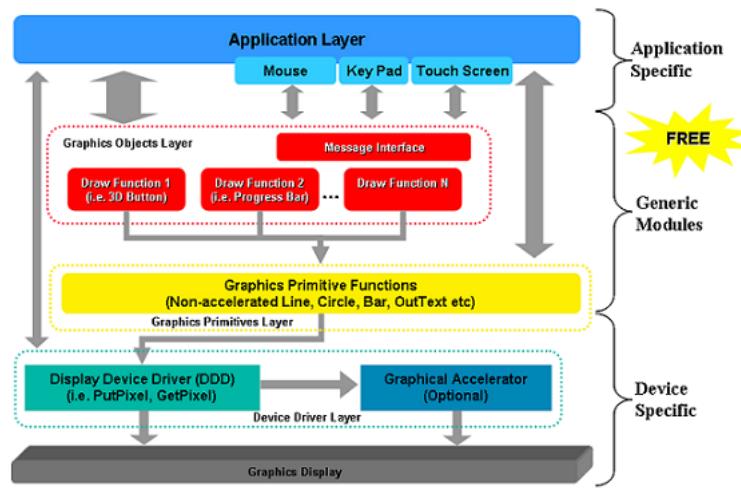
Refer to the Demo Compatibility matrix located in the <install directory>/Microchip/Graphics/Documents/Getting Started/Getting Started - Demo Compatibility Matrix.htm for details.

Description

Refer to the Demo Compatibility matrix located in the <install directory>/Microchip/Graphics/Documents/Getting Started/Getting Started - Demo Compatibility Matrix.htm for details.

5 Library Architecture

The Microchip Graphics Library structure is shown in the following figure.



Microchip Graphics Library Architecture

1. Application Layer – This is the program that utilizes the Graphics Library.
2. User Message Interface- This layer should be implemented by user to provide messages for the library.
3. Graphics Object Layer – This layer renders the widgets controls such as button, slider, window and so on.
4. Graphics Primitives Layer – This layer implements the primitive drawing functions.
5. Device Display Driver – This layer is dependent on the display device being used.
6. Graphics Display Module – This is the display device being used.

The library provides two configurations (Blocking and Non-Blocking).

For Blocking configuration, all draw functions are blocking calls that delay the execution of program until rendering is done.
 For Non-Blocking configuration, draw functions do not wait for the drawing completion and release control to the program.
 In this configuration, a draw function should be called repeatedly until the rendering of that particular draw function is complete. This allows efficient use of microcontroller CPU time since it can perform other tasks if the rendering is not yet done.

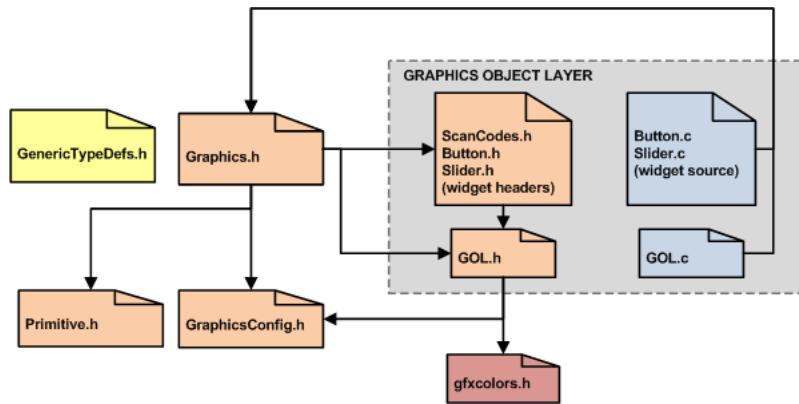
5.1 Graphics Object Layer

Describes the Graphics Object Layer (GOL) structure and its components.

Description

The Graphics Object Layer (GOL) implements the Widgets and the Graphics Library managed messaging and rendering. All Widget drawing are based on the Primitive Layer rendering functions.

The Graphics Object Layer organization is shown on the figure below:

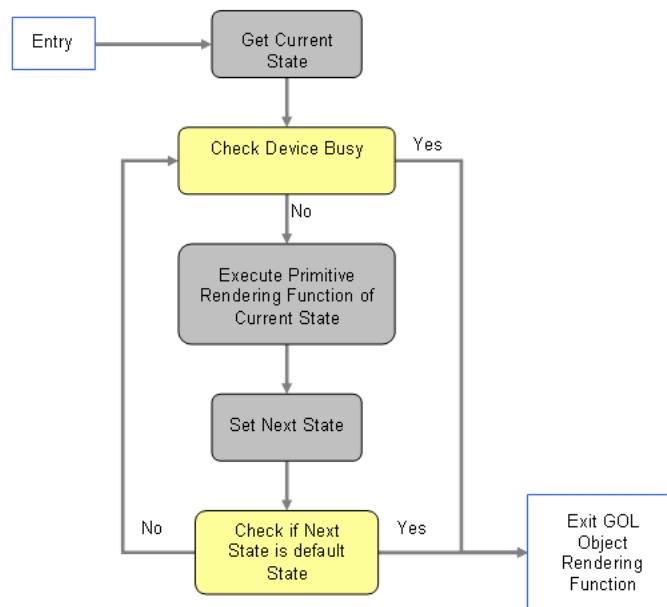


5.1.1 Object Rendering

Describes the difference between the Blocking or Non-Blocking configuration when rendering Objects.

Description

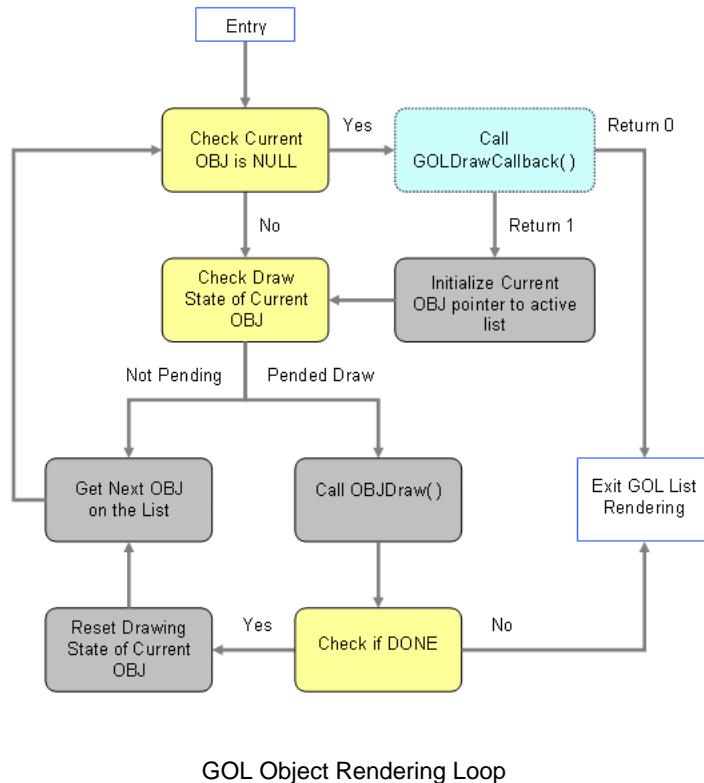
The library can render objects in a Blocking or a Non-Blocking manner. The Non-Blocking configuration is implemented by the use of drawing state machine. Each drawing functions groups the rendering steps into states. Every time a rendering step is executed, the drawing state is updated. Before each step is executed, the display device is checked if it is still busy with the previous rendering operation. If it is busy it returns a non-zero value. This indicates that the draw function must be called again to complete the rendering. The drawing function can be called several times until rendering is completed.



State Machine Controlled Rendering

For Blocking configuration linear flow of rendering is executed. Display device always return a non-busy status.

The GOL level uses the active object linked list for drawing of objects. Each object's state in the list is parsed to determine if the object needs to be redrawn or not. The drawing order is from the head to the tail of the list. This sequence is executed by `GOLDraw` (█)() function. The figure below explains the rendering loop of the `GOLDraw` (█)() function.



GOL Object Rendering Loop

The loop shows two exit points in the sequence. First is when the end of the list is reached and the second is when an `OBJDraw()` returns a NOT DONE status. Reaching the end of the list is a normal exit. This means that all the state machines of the draw functions of each object have reset to default. Exiting with a NOT DONE status means that the latest executed draw function was pended and the object is not yet fully rendered. To complete the rendering, `GOLDraw()` function should be called again. The next call to `GOLDraw()` will pickup the rendering on the last object that returned a not DONE status. This operation makes the rendering functions non-blocking and gives opportunity to release control to program without waiting for the rendering completion.

When all objects in the active object linked list are drawn `GOLDraw()` calls user defined `GOLDrawCallback()` function. User drawing can be done in this callback function. If the function returns a zero, drawing of GOL objects in the active list is suspended. In this case color, clipping region, line type and graphic cursor will not be modified by GOL. If it returns a 1 drawing control is returned to GOL. `GOLDraw()` resume rendering of objects in the current active list. Inside the `GOLDrawCallback()` function, the active object list is not used by `GOLDraw()`. It is safe to perform modification of the list. Please refer to Configuration Settings to set Blocking or Non-Blocking configuration.

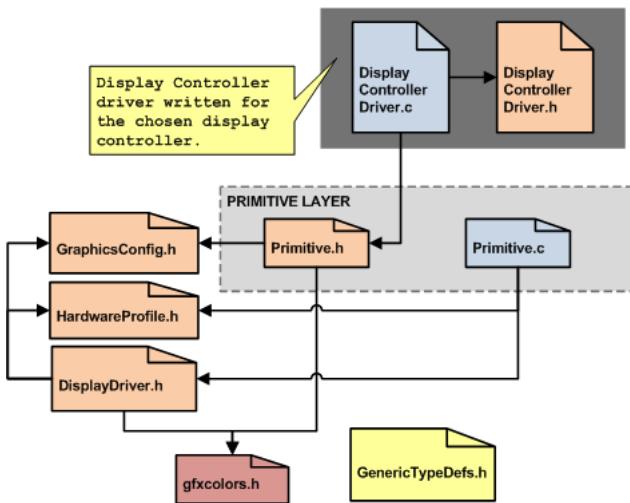
5.2 Graphics Primitive Layer

Describes the Graphics Primitive Layer structure and its components.

Description

This is a hardware independent layer that contains basic rendering functions. These functions can be implemented in the device driver layer if the display device supports hardware acceleration of the function.

The Primitive Layer organization is shown on the figure below:



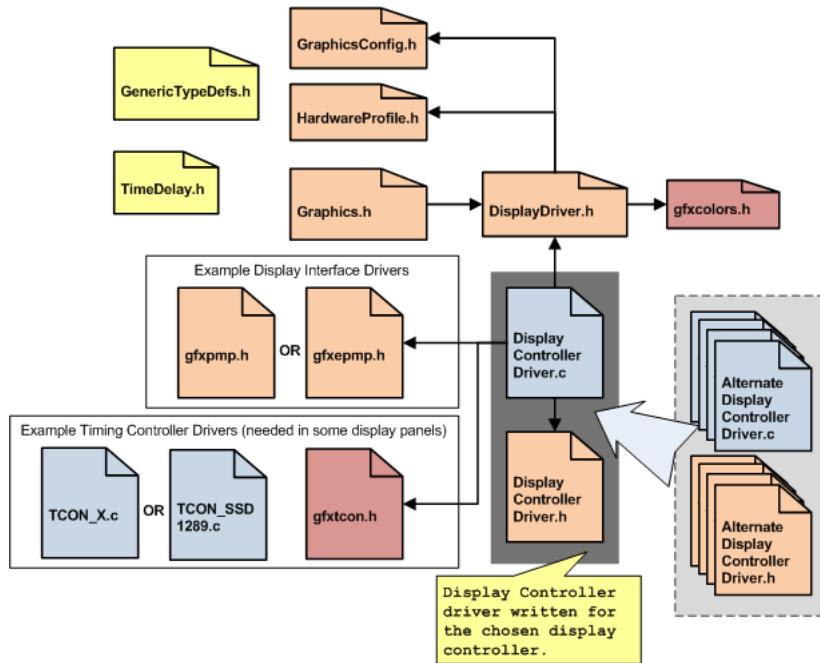
5.3 Display Device Driver Layer

Describes the Display Driver Layer structure and its components.

Description

The Device Driver Layer is the layer that comprises the selection of the display driver file based on the settings specified in the `HardwareProfile.h` file implemented on the application layer.

The Device Driver Layer organization is shown on the figure below:



An option to use a customized driver is also supported by this scheme. The application need only to define custom display macro in the `HardwareProfile` header file. This macro must be unique to the display driver. For example the display macro for the SSD1926 driver is `GFX_USE_DISPLAY_CONTROLLER_SSD1926` (SSD1926). It is recommended that the application keep the same format when naming the display macro, `GFX_USE_DISPLAY_CONTROLLER_<DRIVER NAME>`.

6 Library API

The Microchip Graphics Library is implemented in layers. This section describes the APIs for each layer as well as the Graphics Library Configuration. In addition Advanced Display Device Driver Layer APIs that exists in specific drivers are also described.

6.1 Graphics Library Configuration

The Graphics Library can be customized by adding or specifying the compile time options located in the application code named GraphicsConfig.h (🔗) or the HardwareProfile.h files. The following compile time options are supported by the library.

6.1.1 Graphics Object Layer Configuration

The following compile time options configures the Graphics Library's Object Layer.

6.1.1.1 Input Device Selection

Macros

Name	Description
USE_KEYBOARD (🔗)	<p>Input devices macros that defines the messages that Objects will process. The following definitions indicate the usage of the different input devices:</p> <ul style="list-style-type: none"> • USE_TOUCHSCREEN - enables the touch screen support. • USE_KEYBOARD (🔗) - enables the key board support. <p>Define in GraphicsConfig.h</p>
USE_TOUCHSCREEN (🔗)	<p>Input devices macros that defines the messages that Objects will process. The following definitions indicate the usage of the different input devices:</p> <ul style="list-style-type: none"> • USE_TOUCHSCREEN - enables the touch screen support. • USE_KEYBOARD (🔗) - enables the key board support. <p>Define in GraphicsConfig.h</p>

Description

The Graphics Library comes with two pre-defined user interface. These are the:

- Keyboard interface
- Touchscreen interface

Enabling one or both requires the declaration of the compile switch macros in the GraphicsConfig.h file.

GOL widgets which supports the enabled input device's messages will respond to the user inputs.

For Example:

When using Button (🔗) Widget.

```
#define USE_TOUCHSCREEN (🔗) - will enable the buttons response to user touch on the button widget. The button will automatically be drawn with a pressed state when pressed and release state when user removes the touch on the screen.
```

The compile option selects the input devices used by GOL widgets. Remove or comment out the macro declarations for unused input devices to reduce code size.

6.1.1.1.1 USE_KEYBOARD Macro

File

GraphicsConfig.h

C

```
#define USE_KEYBOARD
```

Overview

Input devices macros that defines the messages that Objects will process. The following definitions indicate the usage of the different input devices:

- USE_TOUCHSCREEN - enables the touch screen support.
- USE_KEYBOARD - enables the key board support.

Define in GraphicsConfig.h

6.1.1.1.2 USE_TOUCHSCREEN Macro

File

GraphicsConfig.h

C

```
#define USE_TOUCHSCREEN
```

Overview

Input devices macros that defines the messages that Objects will process. The following definitions indicate the usage of the different input devices:

- USE_TOUCHSCREEN - enables the touch screen support.
- USE_KEYBOARD (█) - enables the key board support.

Define in GraphicsConfig.h

6.1.1.2 Focus Support Selection

Macros

Name	Description
USE_FOCUS (█)	Keyboard control on some objects can be used by enabling the GOL Focus (USE_FOCUS) support. Define this in GraphicsConfig.h

Description

This compile option allows keyboard input focus. GOLSetFocus (█)(), GOLGetFocus (█)(), GOLCanBeFocused (█)(), GOLGetFocusNext (█)() functions will be available. Focus is also changed by touch screen.

The USE_FOCUS (█) option is located in the GraphicsConfig.h header file.

6.1.1.2.1 USE_FOCUS Macro

File

GraphicsConfig.h

C

```
#define USE_FOCUS
```

Overview

Keyboard control on some objects can be used by enabling the GOL Focus (USE_FOCUS) support. Define this in GraphicsConfig.h

6.1.1.3 Graphics Object Selection

Macros

Name	Description
USE_ANALOGCLOCK (█)	Enable Analog Clock Object.
USE_BUTTON (█)	Enable Button (█) Object.
USE_BUTTON_MULTI_LINE (█)	Enable Multi-Line (█) Button (█) Object
USE_CHECKBOX (█)	Enable Checkbox (█) Object.
USE_DIGITALMETER (█)	Enable DigitalMeter Object.
USE_EDITBOX (█)	Enable Edit Box Object.
USE_GROUPBOX (█)	Enable Group Box Object.
USE_LISTBOX (█)	Enable List Box Object.
USE_METER (█)	Enable Meter (█) Object.
USE_PICTURE (█)	Enable Picture (█) Object.
USE_PROGRESSBAR (█)	Enable Progress Bar (█) Object.
USE_RADIOBUTTON (█)	Enable Radio Button (█) Object.
USE_ROUNDDIAL (█)	Enable Dial (█) Object.
USE_SLIDER (█)	Enable Slider (█) or Scroll (█) Bar (█) Object.
USE_STATICTEXT (█)	Enable Static Text Object.
USE_WINDOW (█)	Enable Window (█) Object.
USE_CUSTOM (█)	Enable Custom Control Object (an example to create customized Object).
USE_GOL (█)	Enable Graphics Object Layer.
USE_TEXTENTRY (█)	Enable TextEntry Object.

Description

These compile options selects objects used. Remove definitions for unused objects to reduce code size.

The USE_GOL (█) and USE_OBJECT options are located in the GraphicsConfig.h header file. USE_GOL (█) should be included if any of the objects are to be used.

6.1.1.3.1 USE_ANALOGCLOCK Macro

File

GraphicsConfig.h

C

```
#define USE_ANALOGCLOCK
```

Description

Enable Analog Clock Object.

6.1.1.3.2 USE_BUTTON Macro

File

GraphicsConfig.h

C

```
#define USE_BUTTON
```

Description

Enable Button (▣) Object.

6.1.1.3.3 USE_BUTTON_MULTI_LINE Macro

File

GraphicsConfig.h

C

```
#define USE_BUTTON_MULTI_LINE
```

Description

Enable Multi-Line (▣) Button (▣) Object

6.1.1.3.4 USE_CHECKBOX Macro

File

GraphicsConfig.h

C

```
#define USE_CHECKBOX
```

Description

Enable Checkbox (▣) Object.

6.1.1.3.5 USE_DIGITALMETER Macro

File

GraphicsConfig.h

C

```
#define USE_DIGITALMETER
```

Description

Enable DigitalMeter Object.

6.1.1.3.6 USE_EDITBOX Macro

File

GraphicsConfig.h

C

```
#define USE_EDITBOX
```

Description

Enable Edit Box Object.

6.1.1.3.7 USE_GROUPBOX Macro

File

GraphicsConfig.h

C

```
#define USE_GROUPBOX
```

Description

Enable Group Box Object.

6.1.1.3.8 USE_LISTBOX Macro

File

GraphicsConfig.h

C

```
#define USE_LISTBOX
```

Description

Enable List Box Object.

6.1.1.3.9 USE_METER Macro

File

GraphicsConfig.h

C

```
#define USE_METER
```

Description

Enable Meter (▣) Object.

6.1.1.3.10 USE_PICTURE Macro

File

GraphicsConfig.h

C

```
#define USE_PICTURE
```

Description

Enable Picture (▣) Object.

6.1.1.3.11 USE_PROGRESSBAR Macro

File

GraphicsConfig.h

C

```
#define USE_PROGRESSBAR
```

Description

Enable Progress Bar (▣) Object.

6.1.1.3.12 USE_RADIOBUTTON Macro

File

GraphicsConfig.h

C

```
#define USE_RADIOBUTTON
```

Description

Enable Radio Button (■) Object.

6.1.1.3.13 USE_ROUNDDIAL Macro

File

GraphicsConfig.h

C

```
#define USE_ROUNDDIAL
```

Description

Enable Dial (■) Object.

6.1.1.3.14 USE_SLIDER Macro

File

GraphicsConfig.h

C

```
#define USE_SLIDER
```

Description

Enable Slider (■) or Scroll (■) Bar (■) Object.

6.1.1.3.15 USE_STATICTEXT Macro

File

GraphicsConfig.h

C

```
#define USE_STATICTEXT
```

Description

Enable Static Text Object.

6.1.1.3.16 USE_WINDOW Macro

File

GraphicsConfig.h

C

```
#define USE_WINDOW
```

Description

Enable Window (■) Object.

6.1.1.3.17 USE_CUSTOM Macro

File

GraphicsConfig.h

C

```
#define USE_CUSTOM
```

Description

Enable Custom Control Object (an example to create customized Object).

6.1.1.3.18 USE_GOL Macro

File

GraphicsConfig.h

C

```
#define USE_GOL
```

Description

Enable Graphics Object Layer.

6.1.1.3.19 USE_TEXTENTRY Macro

File

GraphicsConfig.h

C

```
#define USE_TEXTENTRY
```

Description

Enable TextEntry Object.

6.1.2 Graphics Primitive Layer Configuration

The following compile time options configures the Graphics Library's Primitive Layer.

6.1.2.1 Image Compression Option

Macros

Name	Description
USE_COMP_IPU (✉)	To enable support for DEFLATE compressed images for PutImage (✉)(). When this macro is enabled, the PutImage (✉)() function will be able to process images generated by the Graphics Resource Converter (GRC) that are compressed using the DEFLATE algorithm. PutImage (✉)() will need the IPU module of the Microchip Graphics Module to decompress. Enable this feature only when the driver features the IPU module (example: PIC24FJ2456DA210). Define this in GraphicsConfig.h

USE_COMP_RLE (🔗)	To enable support for RLE compressed images for PutImage (🔗)(). When this macro is enabled, the PutImage (🔗)() function will be able to process images generated by the Graphics Resource Converter (GRC) that are RLE compressed. Define this in GraphicsConfig.h
------------------	--

Description

These compile options to set if the images used for OutImage() are RLE compressed or IPU compressed.

6.1.2.1.1 USE_COMP_IPU Macro

File

GraphicsConfig.h

C

```
#define USE_COMP_IPU
```

Overview

To enable support for DEFLATE compressed images for PutImage (🔗)(). When this macro is enabled, the PutImage (🔗)() function will be able to process images generated by the Graphics Resource Converter (GRC) that are compressed using the DEFLATE algorithm. PutImage (🔗)() will need the IPU module of the Microchip Graphics Module to decompress. Enable this feature only when the driver features the IPU module (example: PIC24FJ2456DA210). Define this in GraphicsConfig.h

6.1.2.1.2 USE_COMP_RLE Macro

File

GraphicsConfig.h

C

```
#define USE_COMP_RLE
```

Overview

To enable support for RLE compressed images for PutImage (🔗)(). When this macro is enabled, the PutImage (🔗)() function will be able to process images generated by the Graphics Resource Converter (GRC) that are RLE compressed. Define this in GraphicsConfig.h

6.1.2.2 Font Type Selection

Macros

Name	Description
USE_MULTIBYTECHAR (🔗)	To enable support for unicode fonts, USE_MULTIBYTECHAR must be defined. This sets the XCHAR (🔗) definition (0-2^16 range). See XCHAR (🔗) for details. Define this in GraphicsConfig.h
USE_UNSIGNED_XCHAR (🔗)	To enable support for unsigned characters data type for fonts, USE_UNSIGNED_XCHAR must be defined. This sets the XCHAR (🔗) definition (0-255 range). See XCHAR (🔗) for details. Define this in GraphicsConfig.h

Description

This compile option selects if the support for unicode fonts, unsigned char or the default signed char type fonts.

There are three types of font (characters) that can be used in the Graphics Library. This gives the user the option to implement multi-language application or use the default signed char type.

Define in GraphicsConfig.h	XCHAR (🔗) type	Description
#define USE_MULTIBYTECHAR (🔗)	#define XCHAR (🔗) unsigned short	Enable support for multi-byte fonts such as Unicode fonts.
#define USE_UNSIGNED_XCHAR (🔗)	#define XCHAR (🔗) unsigned char	Enable support for character range of 0-255.
none of the two are defined	#define XCHAR (🔗) char	Character range is set to 0-127.

Note: Only one of the two or none at all are defined in GraphicsConfig.h.

- #define USE_MULTIBYTECHAR (🔗)
- #define USE_UNSIGNED_XCHAR (🔗)
- when none are defined, XCHAR (🔗) defaults to type char.

See XCHAR (🔗) for details.

6.1.2.2.1 USE_MULTIBYTECHAR Macro

File

GraphicsConfig.h

C

```
#define USE_MULTIBYTECHAR
```

Overview

To enable support for unicode fonts, USE_MULTIBYTECHAR must be defined. This sets the XCHAR (🔗) definition (0-2^16 range). See XCHAR (🔗) for details. Define this in GraphicsConfig.h

6.1.2.2.2 USE_UNSIGNED_XCHAR Macro

File

GraphicsConfig.h

C

```
#define USE_UNSIGNED_XCHAR
```

Overview

To enable support for unsigned characters data type for fonts, USE_UNSIGNED_XCHAR must be defined. This sets the XCHAR (🔗) definition (0-255 range). See XCHAR (🔗) for details. Define this in GraphicsConfig.h

6.1.2.3 Advanced Font Features Selection

Macros

Name	Description
USE_ANTIALIASED_FONTS (🔗)	To enable support for Anti-aliased fonts. Use this feature if the font table generated through the "Graphics Resource Converter" tool has the anti-aliasing enabled. Define this in GraphicsConfig.h

Description

This compile option enables the advanced font features.

6.1.2.3.1 USE_ANTIALIASED_FONTS Macro

File

GraphicsConfig.h

C

```
#define USE_ANTIALIASED_FONTS
```

Overview

To enable support for Anti-aliased fonts. Use this feature if the font table generated through the "Graphics Resource Converter" tool has the anti-aliasing enabled. Define this in GraphicsConfig.h

6.1.2.4 Gradient Bar Rendering

Macros

Name	Description
USE_GRADIENT (🔗)	To enable support for Gradient bars and bevel primitives. Define this in GraphicsConfig.h.

Description

This compile option enables the usage of the Gradient Bar (🔗) and Bevel (🔗) function in the Primitive Layer.

6.1.2.4.1 USE_GRADIENT Macro

File

GraphicsConfig.h

C

```
#define USE_GRADIENT
```

Overview

To enable support for Gradient bars and bevel primitives. Define this in GraphicsConfig.h.

6.1.2.5 Transparent Color Feature in PutImage()

Macros

Name	Description
USE_TRANSPARENT_COLOR (🔗)	To enable support for transparent color in PutImage (🔗)(). Enabling this macro enables the use of a transparent color (set by TransparentColorEnable (🔗)()) in rendering images by PutImage (🔗)(). When a pixel in the image matches the transparent color set, the pixel is not rendered to the screen. This is useful in rendering rounded icons or images to the screen with a complex background. Define this in GraphicsConfig.h

Description

This compile option enables the transparent color feature in PutImage (🔗)().

6.1.2.5.1 USE_TRANSPARENT_COLOR Macro

File

GraphicsConfig.h

C

```
#define USE_TRANSPARENT_COLOR
```

Overview

To enable support for transparent color in PutImage (🔗). Enabling this macro enables the use of a transparent color (set by TransparentColorEnable (🔗)) in rendering images by PutImage (🔗). When a pixel in the image matches the transparent color set, the pixel is not rendered to the screen. This is useful in rendering rounded icons or images to the screen with a complex background. Define this in GraphicsConfig.h

6.1.2.6 Alpha Blend Option

Macros

Name	Description
USE_ALPHABLEND_LITE (🔗)	To enable support for Alpha Blending on the Primitive Layer. This feature is only limited on Alpha-Blending a Bar (🔗) of a specified color (set by SetColor (🔗)) to the destination defined by the parameters of the Bar (🔗) function call. The Alpha level used is set by SetAlpha (🔗). Define this in GraphicsConfig.h

Description

This compile option enables the Alpha-Blend feature in Primitive Layer.

6.1.2.6.1 USE_ALPHABLEND_LITE Macro

File

GraphicsConfig.h

C

```
#define USE_ALPHABLEND_LITE
```

Overview

To enable support for Alpha Blending on the Primitive Layer. This feature is only limited on Alpha-Blending a Bar (🔗) of a specified color (set by SetColor (🔗)) to the destination defined by the parameters of the Bar (🔗) function call. The Alpha level used is set by SetAlpha (🔗). Define this in GraphicsConfig.h

6.1.2.7 External Memory Buffer

see EXTERNAL_FONT_BUFFER_SIZE (🔗)

6.1.3 Display Device Driver Layer Configuration

Macros

Name	Description
USE_ALPHABLEND (🔗)	To enable support for Alpha Blending. Use this feature only if the display driver used can support alpha blending. Define this in GraphicsConfig.h
USE_DOUBLE_BUFFERING (🔗)	To enable support for double buffering. Use this feature only if the display driver used can support double buffering. Define this in GraphicsConfig.h

GFX_LCD_TYPE (🔗)	Sets the type of display glass used. Define this in the Hardware Profile. <ul style="list-style-type: none"> • <code>#define GFX_LCD_TYPE GFX_LCD_TFT</code> (🔗) - sets type TFT display • <code>#define GFX_LCD_TYPE GFX_LCD_CSTN</code> (🔗) - sets type color STN display • <code>#define GFX_LCD_TYPE GFX_LCD_MSTN</code> (🔗) - sets type mon STN display • <code>#define GFX_LCD_TYPE GFX_LCD_OFF</code> (🔗) - display is turned off
STN_DISPLAY_WIDTH (🔗)	Sets the STN glass data width. Define this in the Hardware Profile. <ul style="list-style-type: none"> • <code>#define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_4</code> (🔗) - use 4-bit wide interface • <code>#define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_8</code> (🔗) - Use 8-bit wide interface • <code>#define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_16</code> (🔗) - Use 16-bit wide interface

Description

The following compile time options configures the Graphics Library's Display Device Driver Layer. The choices are based on the specific hardware used.

See Hardware Profile ([🔗](#)) for more options.

6.1.3.1 USE_ALPHABLEND Macro

File

S1D13517.c

C

```
#define USE_ALPHABLEND
```

Overview

To enable support for Alpha Blending. Use this feature only if the display driver used can support alpha blending. Define this in GraphicsConfig.h

6.1.3.2 USE_DOUBLE_BUFFERING Macro

File

GraphicsConfig.h

C

```
#define USE_DOUBLE_BUFFERING
```

Overview

To enable support for double buffering. Use this feature only if the display driver used can support double buffering. Define this in GraphicsConfig.h

6.1.3.3 GFX_LCD_TYPE Macro

File

mchpGfxDrv.h

C

```
#define GFX_LCD_TYPE
```

Macros

Name	Description
GFX_LCD_CSTN (🔗)	Type Color STN Display
GFX_LCD_MSTN (🔗)	Type Mono STN Display
GFX_LCD_OFF (🔗)	display is turned off
GFX_LCD_TFT (🔗)	Type TFT Display

Overview

Sets the type of display glass used. Define this in the Hardware Profile.

- `#define GFX_LCD_TYPE GFX_LCD_TFT (🔗)` - sets type TFT display
- `#define GFX_LCD_TYPE GFX_LCD_CSTN (🔗)` - sets type color STN display
- `#define GFX_LCD_TYPE GFX_LCD_MSTN (🔗)` - sets type mon STN display
- `#define GFX_LCD_TYPE GFX_LCD_OFF (🔗)` - display is turned off

6.1.3.3.1 GFX_LCD_CSTN Macro

File

SSD1926.h

C

```
#define GFX_LCD_CSTN 0x03           // Type Color STN Display
```

Description

Type Color STN Display

6.1.3.3.2 GFX_LCD_MSTN Macro

File

SSD1926.h

C

```
#define GFX_LCD_MSTN 0x02           // Type Mono STN Display
```

Description

Type Mono STN Display

6.1.3.3.3 GFX_LCD_OFF Macro

File

mchpGfxDrv.h

C

```
#define GFX_LCD_OFF 0x00           // display is turned off
```

Description

display is turned off

6.1.3.3.4 GFX_LCD_TFT Macro

File

SSD1926.h

C

```
#define GFX_LCD_TFT 0x01           // Type TFT Display
```

Description

Type TFT Display

6.1.3.4 STN_DISPLAY_WIDTH Macro

File

mchpGfxDrv.h

C

```
#define STN_DISPLAY_WIDTH
```

Macros

Name	Description
STN_DISPLAY_WIDTH_16 (█)	display interface is 16 bits wide
STN_DISPLAY_WIDTH_4 (█)	display interface is 4 bits wide
STN_DISPLAY_WIDTH_8 (█)	display interface is 8 bits wide

Overview

Sets the STN glass data width. Define this in the Hardware Profile.

- #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_4 (█) - use 4-bit wide interface
- #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_8 (█) - Use 8-bit wide interface
- #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_16 (█) - Use 16-bit wide interface

6.1.3.4.1 STN_DISPLAY_WIDTH_16 Macro

File

mchpGfxDrv.h

C

```
#define STN_DISPLAY_WIDTH_16 0x02      // display interface is 16 bits wide
```

Description

display interface is 16 bits wide

6.1.3.4.2 STN_DISPLAY_WIDTH_4 Macro

File

mchpGfxDrv.h

C

```
#define STN_DISPLAY_WIDTH_4 0x00      // display interface is 4 bits wide
```

Description

display interface is 4 bits wide

6.1.3.4.3 STN_DISPLAY_WIDTH_8 Macro

File

mchpGfxDrv.h

C

```
#define STN_DISPLAY_WIDTH_8 0x01      // display interface is 8 bits wide
```

Description

display interface is 8 bits wide

6.1.4 Application Configuration

6.1.4.1 Configuration Setting

Macros

Name	Description
USE_NONBLOCKING_CONFIG (🔗)	Blocking and Non-Blocking configuration selection. To enable non-blocking configuration USE_NONBLOCKING_CONFIG must be defined. If this is not defined, blocking configuration is assumed. Define this in GraphicsConfig.h

Description

This selects the configuration of the library. When Non-blocking configuration is selected, state machine based rendering is used to perform object rendering.

When blocking configuration is used, this line MUST be commented. In this case object rendering will not exit until the object is fully rendered.

The USE_NONBLOCKING_CONFIG (🔗) option is located in the GraphicsConfig.h header file.

6.1.4.1.1 USE_NONBLOCKING_CONFIG Macro

File

GraphicsConfig.h

C

```
#define USE_NONBLOCKING_CONFIG
```

Overview

Blocking and Non-Blocking configuration selection. To enable non-blocking configuration USE_NONBLOCKING_CONFIG must be defined. If this is not defined, blocking configuration is assumed. Define this in GraphicsConfig.h

6.1.4.2 Font Source Selection

Macros

Name	Description
USE_FONT_FLASH (🔗)	Font data can be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for fonts located in internal flash and external memory. Define this in GraphicsConfig.h <ul style="list-style-type: none"> • USE_FONT_FLASH - Font in internal flash memory support. • USE_FONT_EXTERNAL (🔗) - Font in external memory support (including external memory mapped to EDS).
USE_FONT_EXTERNAL (🔗)	Font data can be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for fonts located in internal flash and external memory. Define this in GraphicsConfig.h <ul style="list-style-type: none"> • USE_FONT_FLASH - Font in internal flash memory support. • USE_FONT_EXTERNAL (🔗) - Font in external memory support (including external memory mapped to EDS).
USE_GFX_FONT_IN_PROGRAM_SECTION (🔗)	For XC16 or C30 builds only: When placing fonts in internal data memory, there is a 32K limit for data space. The total data should not exceed 32K. When this is unavoidable, the macro USE_GFX_FONT_IN_PROGRAM_SECTION will relocate the font in program space. This will remove the 32K restriction but at the expense of slower access. Define this in GraphicsConfig.h to enable the font to be placed in program space.

Description

Font data can be placed in multiple locations. Set these options in the GraphicsConfig.h header file.

- USE_FONT_FLASH (🔗) - Font in internal flash memory support. When placed in internal flash memory, it can further classified to be placed in program flash by adding USE_GFX_FONT_IN_PROGRAM_SECTION (🔗).
- USE_FONT_EXTERNAL (🔗) - Font in external memory support. Use this for fonts located in external memory like SPI Flash or external memory mapped to Extended Data Space.

6.1.4.2.1 USE_FONT_FLASH Macro

File

GraphicsConfig.h

C

```
#define USE_FONT_FLASH
```

Overview

Font data can be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for fonts located in internal flash and external memory. Define this in GraphicsConfig.h

- USE_FONT_FLASH - Font in internal flash memory support.
- USE_FONT_EXTERNAL (🔗) - Font in external memory support (including external memory mapped to EDS).

6.1.4.2.2 USE_FONT_EXTERNAL Macro

File

GraphicsConfig.h

C

```
#define USE_FONT_EXTERNAL
```

Overview

Font data can be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for fonts located in internal flash and external memory. Define this in GraphicsConfig.h

- USE_FONT_FLASH - Font in internal flash memory support.
- USE_FONT_EXTERNAL - Font in external memory support (including external memory mapped to EDS).

6.1.4.2.3 USE_GFX_FONT_IN_PROGRAM_SECTION Macro**File**

GraphicsConfig.h

C

```
#define USE_GFX_FONT_IN_PROGRAM_SECTION
```

Overview

For XC16 or C30 builds only: When placing fonts in internal data memory, there is a 32K limit for data space. The total data should not exceed 32K. When this is unavoidable, the macro USE_GFX_FONT_IN_PROGRAM_SECTION will relocate the font in program space. This will remove the 32K restriction but at the expense of slower access. Define this in GraphicsConfig.h to enable the font to be placed in program space.

6.1.4.3 Image Source Selection**Macros**

Name	Description
USE_BITMAP_FLASH (🔗)	<p>Similar to Font data bitmaps can also be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for bitmaps located in internal flash and external memory. Define this in GraphicsConfig.h</p> <ul style="list-style-type: none"> • USE_BITMAP_FLASH - Images located in internal flash memory. • USE_BITMAP_EXTERNAL (🔗) - Images located in external memory (including external memory mapped to EDS)..
USE_BITMAP_EXTERNAL (🔗)	<p>Similar to Font data bitmaps can also be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for bitmaps located in internal flash and external memory. Define this in GraphicsConfig.h</p> <ul style="list-style-type: none"> • USE_BITMAP_FLASH - Images located in internal flash memory. • USE_BITMAP_EXTERNAL (🔗) - Images located in external memory (including external memory mapped to EDS)..

Description

Similar to Font data bitmaps can also be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for bitmaps located in internal flash and external memory.

The USE_BITMAP_FLASH (🔗) and USE_BITMAP_EXTERNAL (🔗) options are located in the GraphicsConfig.h header file.

6.1.4.3.1 USE_BITMAP_FLASH Macro**File**

GraphicsConfig.h

C

```
#define USE_BITMAP_FLASH
```

Overview

Similar to Font data bitmaps can also be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for bitmaps located in internal flash and external memory. Define this in GraphicsConfig.h

- USE_BITMAP_FLASH - Images located in internal flash memory.
- USE_BITMAP_EXTERNAL (🔗) - Images located in external memory (including external memory mapped to EDS)..

6.1.4.3.2 USE_BITMAP_EXTERNAL Macro**File**

GraphicsConfig.h

C

```
#define USE_BITMAP_EXTERNAL
```

Overview

Similar to Font data bitmaps can also be placed in two locations. One is in FLASH memory and the other is from external memory. Definining one or both enables the support for bitmaps located in internal flash and external memory. Define this in GraphicsConfig.h

- USE_BITMAP_FLASH - Images located in internal flash memory.
- USE_BITMAP_EXTERNAL - Images located in external memory (including external memory mapped to EDS)..

6.1.4.4 Miscellaneous**Macros**

Name	Description
USE_BITMAP_NO_PADDING_LINE (🔗)	When this macro is enabled, bitmap images used has no padding. Define this in GraphicsConfig.h. When converting images for use in the Graphics Library, the Graphics Resource Converter has an option to set the images to be padded or not padded. When bitmaps are padded, this means that each horizontal line will start on a byte boundary. Unpadded bitmaps allows the least resource space for a bitmap. Unpadded bitmaps also allows support for display controllers with windowing and auto-increment features.
USE_PALETTE_EXTERNAL (🔗)	Palettes can also be specified to reside in external memory similar to fonts and images. Use this when the palette is located in external memory. Define this in GraphicsConfig.h
USE_PALETTE (🔗)	Using Palettes, different colors can be used with the same bit depth. Define this in GraphicsConfig.h
COLOR_DEPTH (🔗)	Specifies the color depth used in the application defined in GraphicsConfig.h.
GFX_free (🔗)	When using Operating Systems (OS), define the OS specific malloc() and free() functions for compatibility with the OS based systems. Define these in GraphicsConfig.h
GFX_malloc (🔗)	When using Operating Systems (OS), define the OS specific malloc() and free() functions for compatibility with the OS based systems. Define these in GraphicsConfig.h

Description

This contains miscellaneous macros and functions that can be redefined for various system support such as Operating System defined functions.

6.1.4.4.1 USE_BITMAP_NO_PADDING_LINE Macro

File

GraphicsConfig.h

C

```
#define USE_BITMAP_NO_PADDING_LINE
```

Overview

When this macro is enabled, bitmap images used has no padding. Define this in GraphicsConfig.h. When converting images for use in the Graphics Library, the Graphics Resource Converter has an option to set the images to be padded or not padded. When bitmaps are padded, this means that each horizontal line will start on a byte boundary. Unpadded bitmaps allows the least resource space for a bitmap. Unpadded bitmaps also allows support for display controllers with windowing and auto-increment features.

6.1.4.4.2 USE_PALETTE_EXTERNAL Macro

File

GraphicsConfig.h

C

```
#define USE_PALETTE_EXTERNAL
```

Overview

Palettes can also be specified to reside in external memory similar to fonts and images. Use this when the palette is located in external memory. Define this in GraphicsConfig.h

6.1.4.4.3 USE_PALETTE Macro

File

GraphicsConfig.h

C

```
#define USE_PALETTE
```

Overview

Using Palettes, different colors can be used with the same bit depth. Define this in GraphicsConfig.h

6.1.4.4.4 COLOR_DEPTH Macro

File

GraphicsConfig.h

C

```
#define COLOR_DEPTH 16
```

Overview

Specifies the color depth used in the application defined in GraphicsConfig.h.

6.1.4.4.5 GFX_free Macro

File

GraphicsConfig.h

C

```
#define GFX_free(pObj) free(pObj)           // <COPY GFX_malloc>
```

Overview

When using Operating Systems (OS), define the OS specific malloc() and free() functions for compatibility with the OS based systems. Define these in GraphicsConfig.h

6.1.4.4.6 GFX_malloc Macro**File**

GraphicsConfig.h

C

```
#define GFX_malloc(size) malloc(size)
```

Overview

When using Operating Systems (OS), define the OS specific malloc() and free() functions for compatibility with the OS based systems. Define these in GraphicsConfig.h

6.1.4.5 GraphicsConfig.h Example

This is an example of the GraphicsConfig.h file implementation:

```
/////////////////////// COMPILE OPTIONS AND DEFAULTS //////////////////

#define USE_NONBLOCKING_CONFIG          // Comment this line to use blocking configuration
#define USE_FOCUS                      // Comment this line when not using FOCUS
#define USE_TOUCHSCREEN                // Enable touch screen support.
#define USE_KEYBOARD                   // Enable key board support.

#define USE_GOL                         // Enable Graphics Object Layer.
#define USE_BUTTON                      // Enable Button Object.
#define USE_CHART                       // Enable Chart Object.
#define USE_WINDOW                      // Enable Window Object.
#define USE_CHECKBOX                    // Enable Checkbox Object.
#define USE_RADIOBUTTON                 // Enable Radio Button Object.
#define USE_EDITBOX                     // Enable Edit Box Object.
#define USE_LISTBOX                     // Enable List Box Object.
#define USE_SLIDER                      // Enable Slider or Scroll Bar Object.
#define USE_PROGRESSBAR                 // Enable Progress Bar Object.
#define USE_STATICTEXT                  // Enable Static Text Object.
#define USE_PICTURE                     // Enable Picture Object.
#define USE_GROUPBOX                    // Enable Group Box Object.
#define USE_ROUNDIAL                    // Enable Dial Object.
#define USE_METER                       // Enable Meter Object.
#define USE_TEXTENTRY                   // Enable Text Entry Object.
#define USE_CUSTOM                      // Enable Custom Control Object (an example to
create customized Object).

#define USE_MULTIBYTECHAR              // Enable unicode derived characters
#define USE_FONT_FLASH                 // Support for fonts located in internal flash
#define USE_BITMAP_FLASH               // Support for bitmaps located in internal flash

#define GOL_EMBOSS_SIZE                2
#define COLOR_DEPTH                    16 // The color depth used
```

6.1.5 Hardware Profile

These functions and macros are used to determine hardware profile settings on the chosen PIC microcontroller and

hardware such as demo boards used.

6.1.5.1 PMP Interface

Macros

Name	Description
USE_8BIT_PMP (🔗)	<p>Specifies the interface type to the Parallel Master Port (PMP) or Enhanced Parallel Master Port (EPMP).</p> <ul style="list-style-type: none"> • USE_8BIT_PMP - Use 8-bit interface to PMP or EPMP • USE_16BIT_PMP (🔗) - Use 16-bit interface to PMP or EPMP
USE_16BIT_PMP (🔗)	<p>Specifies the interface type to the Parallel Master Port (PMP) or Enhanced Parallel Master Port (EPMP).</p> <ul style="list-style-type: none"> • USE_8BIT_PMP - Use 8-bit interface to PMP or EPMP • USE_16BIT_PMP (🔗) - Use 16-bit interface to PMP or EPMP

Description

Specifies the interface type to the Parallel Master Port (PMP) or Enhanced Parallel Master Port (EPMP).

6.1.5.1.1 USE_8BIT_PMP Macro

File

HardwareProfile.h

C

```
#define USE_8BIT_PMP
```

Overview

Specifies the interface type to the Parallel Master Port (PMP) or Enhanced Parallel Master Port (EPMP).

- USE_8BIT_PMP - Use 8-bit interface to PMP or EPMP
- USE_16BIT_PMP (🔗) - Use 16-bit interface to PMP or EPMP

6.1.5.1.2 USE_16BIT_PMP Macro

File

HardwareProfile.h

C

```
#define USE_16BIT_PMP
```

Overview

Specifies the interface type to the Parallel Master Port (PMP) or Enhanced Parallel Master Port (EPMP).

- USE_8BIT_PMP - Use 8-bit interface to PMP or EPMP
- USE_16BIT_PMP - Use 16-bit interface to PMP or EPMP

6.1.5.2 Development Platform Used

Macros

Name	Description
EXPLORER_16 (🔗)	<p>Specifies the Development Platform used for the Microchip Graphics Library demos.</p> <ul style="list-style-type: none"> • EXPLORER_16 - Using the Explorer 16 Development Board (DM240001). • PIC24FJ256DA210_DEV_BOARD (🔗) - Using the PIC24FJ256DA210 Development Board (DM240312). • MEB_BOARD (🔗) - Using the Multi-Media Expansion Board (DM320005). • PIC_SK (🔗) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).
PIC24FJ256DA210_DEV_BOARD (🔗)	<p>Specifies the Development Platform used for the Microchip Graphics Library demos.</p> <ul style="list-style-type: none"> • EXPLORER_16 - Using the Explorer 16 Development Board (DM240001). • PIC24FJ256DA210_DEV_BOARD (🔗) - Using the PIC24FJ256DA210 Development Board (DM240312). • MEB_BOARD (🔗) - Using the Multi-Media Expansion Board (DM320005). • PIC_SK (🔗) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).
MEB_BOARD (🔗)	<p>Specifies the Development Platform used for the Microchip Graphics Library demos.</p> <ul style="list-style-type: none"> • EXPLORER_16 - Using the Explorer 16 Development Board (DM240001). • PIC24FJ256DA210_DEV_BOARD (🔗) - Using the PIC24FJ256DA210 Development Board (DM240312). • MEB_BOARD (🔗) - Using the Multi-Media Expansion Board (DM320005). • PIC_SK (🔗) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).
PIC_SK (🔗)	<p>Specifies the Development Platform used for the Microchip Graphics Library demos.</p> <ul style="list-style-type: none"> • EXPLORER_16 - Using the Explorer 16 Development Board (DM240001). • PIC24FJ256DA210_DEV_BOARD (🔗) - Using the PIC24FJ256DA210 Development Board (DM240312). • MEB_BOARD (🔗) - Using the Multi-Media Expansion Board (DM320005). • PIC_SK (🔗) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).

Description

Specifies the Development Platform used for the Microchip Graphics Library demos.

6.1.5.2.1 EXPLORER_16 Macro

File

HardwareProfile.h

C

```
#define EXPLORER_16
```

Overview

Specifies the Development Platform used for the Microchip Graphics Library demos.

- EXPLORER_16 - Using the Explorer 16 Development Board (DM240001).
- PIC24FJ256DA210_DEV_BOARD (█) - Using the PIC24FJ256DA210 Development Board (DM240312).
- MEB_BOARD (█) - Using the Multi-Media Expansion Board (DM320005).
- PIC_SK (█) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).

6.1.5.2.2 PIC24FJ256DA210_DEV_BOARD Macro

File

HardwareProfile.h

C

```
#define PIC24FJ256DA210_DEV_BOARD
```

Overview

Specifies the Development Platform used for the Microchip Graphics Library demos.

- EXPLORER_16 - Using the Explorer 16 Development Board (DM240001).
- PIC24FJ256DA210_DEV_BOARD - Using the PIC24FJ256DA210 Development Board (DM240312).
- MEB_BOARD (█) - Using the Multi-Media Expansion Board (DM320005).
- PIC_SK (█) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).

6.1.5.2.3 MEB_BOARD Macro

File

HardwareProfile.h

C

```
#define MEB_BOARD
```

Overview

Specifies the Development Platform used for the Microchip Graphics Library demos.

- EXPLORER_16 - Using the Explorer 16 Development Board (DM240001).
- PIC24FJ256DA210_DEV_BOARD (█) - Using the PIC24FJ256DA210 Development Board (DM240312).
- MEB_BOARD - Using the Multi-Media Expansion Board (DM320005).
- PIC_SK (█) - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).

6.1.5.2.4 PIC_SK Macro

File

HardwareProfile.h

C

```
#define PIC_SK
```

Overview

Specifies the Development Platform used for the Microchip Graphics Library demos.

- EXPLORER_16 - Using the Explorer 16 Development Board (DM240001).
- PIC24FJ256DA210_DEV_BOARD (🔗) - Using the PIC24FJ256DA210 Development Board (DM240312).
- MEB_BOARD (🔗) - Using the Multi-Media Expansion Board (DM320005).
- PIC_SK - Using PIC32 or dsPIC Starter Kit (examples: PIC32 Starter Kit (DM320001), PIC32 USB Starter Kit II (DM320003-2), PIC32 Ethernet Starter Kit (DM320004)).

6.1.5.3 Graphics PICtail Used

Macros

Name	Description
GFX_PICTAIL_LCC (🔗)	<p>Specifies the Graphics PICtail Display Panel used.</p> <ul style="list-style-type: none"> GFX_PICTAIL_V3 - Graphics LCD Controller PICtail Plus SSD1926 Board (AC164127-5) GFX_PICTAIL_V3E (🔗) - Graphics LCD Controller PICtail Plus S1D13517 Board (AC164127-7) GFX_PICTAIL_LCC (🔗) - Low Cost Controllerless (LCC) Graphics PICtail™ Plus Board (AC164144)
GFX_PICTAIL_V3 (🔗)	<p>Specifies the Graphics PICtail Display Panel used.</p> <ul style="list-style-type: none"> GFX_PICTAIL_V3 - Graphics LCD Controller PICtail Plus SSD1926 Board (AC164127-5) GFX_PICTAIL_V3E (🔗) - Graphics LCD Controller PICtail Plus S1D13517 Board (AC164127-7) GFX_PICTAIL_LCC (🔗) - Low Cost Controllerless (LCC) Graphics PICtail™ Plus Board (AC164144)
GFX_PICTAIL_V3E (🔗)	<p>Specifies the Graphics PICtail Display Panel used.</p> <ul style="list-style-type: none"> GFX_PICTAIL_V3 - Graphics LCD Controller PICtail Plus SSD1926 Board (AC164127-5) GFX_PICTAIL_V3E (🔗) - Graphics LCD Controller PICtail Plus S1D13517 Board (AC164127-7) GFX_PICTAIL_LCC (🔗) - Low Cost Controllerless (LCC) Graphics PICtail™ Plus Board (AC164144)

Description

Specifies the Graphics PICtail Display Panel used.

6.1.5.3.1 GFX_PICTAIL_LCC Macro

File

HardwareProfile.h

C

```
#define GFX_PICTAIL_LCC
```

Overview

Specifies the Graphics PICtail Display Panel used.

- GFX_PICTAIL_V3 - Graphics LCD Controller PICtail Plus SSD1926 Board (AC164127-5)

- GFX_PICTAIL_V3E (🔗) - Graphics LCD Controller PICtail Plus S1D13517 Board (AC164127-7)
- GFX_PICTAIL_LCC - Low Cost Controllerless (LCC) Graphics PICtail™ Plus Board (AC164144)

6.1.5.3.2 GFX_PICTAIL_V3 Macro

File

HardwareProfile.h

C

```
#define GFX_PICTAIL_V3
```

Overview

Specifies the Graphics PICtail Display Panel used.

- GFX_PICTAIL_V3 - Graphics LCD Controller PICtail Plus SSD1926 Board (AC164127-5)
- GFX_PICTAIL_V3E (🔗) - Graphics LCD Controller PICtail Plus S1D13517 Board (AC164127-7)
- GFX_PICTAIL_LCC (🔗) - Low Cost Controllerless (LCC) Graphics PICtail™ Plus Board (AC164144)

6.1.5.3.3 GFX_PICTAIL_V3E Macro

File

HardwareProfile.h

C

```
#define GFX_PICTAIL_V3E
```

Overview

Specifies the Graphics PICtail Display Panel used.

- GFX_PICTAIL_V3 - Graphics LCD Controller PICtail Plus SSD1926 Board (AC164127-5)
- GFX_PICTAIL_V3E - Graphics LCD Controller PICtail Plus S1D13517 Board (AC164127-7)
- GFX_PICTAIL_LCC (🔗) - Low Cost Controllerless (LCC) Graphics PICtail™ Plus Board (AC164144)

6.1.5.4 Display Controller Used

Macros

Name	Description
GFX_USE_DISPLAY_CONTROLLER_DMA (🔗)	<p>Specifies the controller used in the Graphics Library supplied demo.</p> <ul style="list-style-type: none"> • GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution. • GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (🔗) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) • GFX_USE_DISPLAY_CONTROLLER_SSD1926 (🔗) - Using the Solomon Systech SSD1926 Display Controller. • GFX_USE_DISPLAY_CONTROLLER_S1D13517 (🔗) - Using the Epson S1D13517 Display Controller.

GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (🔗)	Specifies the controller used in the Graphics Library supplied demo. <ul style="list-style-type: none"> • GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution. • GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (🔗) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) • GFX_USE_DISPLAY_CONTROLLER_SSD1926 (🔗) - Using the Solomon Systech SSD1926 Display Controller. • GFX_USE_DISPLAY_CONTROLLER_S1D13517 (🔗) - Using the Epson S1D13517 Display Controller.
GFX_USE_DISPLAY_CONTROLLER_S1D13517 (🔗)	Specifies the controller used in the Graphics Library supplied demo. <ul style="list-style-type: none"> • GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution. • GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (🔗) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) • GFX_USE_DISPLAY_CONTROLLER_SSD1926 (🔗) - Using the Solomon Systech SSD1926 Display Controller. • GFX_USE_DISPLAY_CONTROLLER_S1D13517 (🔗) - Using the Epson S1D13517 Display Controller.
GFX_USE_DISPLAY_CONTROLLER_SSD1926 (🔗)	Specifies the controller used in the Graphics Library supplied demo. <ul style="list-style-type: none"> • GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution. • GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (🔗) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) • GFX_USE_DISPLAY_CONTROLLER_SSD1926 (🔗) - Using the Solomon Systech SSD1926 Display Controller. • GFX_USE_DISPLAY_CONTROLLER_S1D13517 (🔗) - Using the Epson S1D13517 Display Controller.

Description

Specifies the controller used in the Graphics Library supplied demo.

6.1.5.4.1 GFX_USE_DISPLAY_CONTROLLER_DMA Macro**File**

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_CONTROLLER_DMA
```

Overview

Specifies the controller used in the Graphics Library supplied demo.

- GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution.
- GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 ([🔗](#)) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family)
- GFX_USE_DISPLAY_CONTROLLER_SSD1926 ([🔗](#)) - Using the Solomon Systech SSD1926 Display Controller.
- GFX_USE_DISPLAY_CONTROLLER_S1D13517 ([🔗](#)) - Using the Epson S1D13517 Display Controller.

6.1.5.4.2 GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 Macro

File

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210
```

Overview

Specifies the controller used in the Graphics Library supplied demo.

- GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution.
- GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family)
- GFX_USE_DISPLAY_CONTROLLER_SSD1926 (■) - Using the Solomon Systech SSD1926 Display Controller.
- GFX_USE_DISPLAY_CONTROLLER_S1D13517 (■) - Using the Epson S1D13517 Display Controller.

6.1.5.4.3 GFX_USE_DISPLAY_CONTROLLER_S1D13517 Macro

File

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_CONTROLLER_S1D13517
```

Overview

Specifies the controller used in the Graphics Library supplied demo.

- GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution.
- GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (■) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family)
- GFX_USE_DISPLAY_CONTROLLER_SSD1926 (■) - Using the Solomon Systech SSD1926 Display Controller.
- GFX_USE_DISPLAY_CONTROLLER_S1D13517 - Using the Epson S1D13517 Display Controller.

6.1.5.4.4 GFX_USE_DISPLAY_CONTROLLER_SSD1926 Macro

File

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_CONTROLLER_SSD1926
```

Overview

Specifies the controller used in the Graphics Library supplied demo.

- GFX_USE_DISPLAY_CONTROLLER_DMA - Using the PIC32 Low Cost Controllerless (LCC) solution.
- GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 (■) - Use the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family)
- GFX_USE_DISPLAY_CONTROLLER_SSD1926 - Using the Solomon Systech SSD1926 Display Controller.
- GFX_USE_DISPLAY_CONTROLLER_S1D13517 (■) - Using the Epson S1D13517 Display Controller.

6.1.5.5 Display Panel Used

Macros

Name	Description
GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q (█)	<p>Specifies the Graphics Display Panel used.</p> <ul style="list-style-type: none"> • GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_1 18W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4) • GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q (█) - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6) • GFX_USE_DISPLAY_PANEL_TFT_640480_8_E (█) - 5.7 inch VGA Truly TFT Display Board (AC164127-8) • GFX_USE_DISPLAY_PANEL_TFT_800480_33_E (█) - 7 inch WVGA Truly TFT Display Board (AC164127-8)
GFX_USE_DISPLAY_PANEL_TFT_640480_8_E (█)	<p>Specifies the Graphics Display Panel used.</p> <ul style="list-style-type: none"> • GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_1 18W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4) • GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q (█) - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6) • GFX_USE_DISPLAY_PANEL_TFT_640480_8_E (█) - 5.7 inch VGA Truly TFT Display Board (AC164127-8) • GFX_USE_DISPLAY_PANEL_TFT_800480_33_E (█) - 7 inch WVGA Truly TFT Display Board (AC164127-8)
GFX_USE_DISPLAY_PANEL_TFT_800480_33_E (█)	<p>Specifies the Graphics Display Panel used.</p> <ul style="list-style-type: none"> • GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_1 18W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4) • GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q (█) - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6) • GFX_USE_DISPLAY_PANEL_TFT_640480_8_E (█) - 5.7 inch VGA Truly TFT Display Board (AC164127-8) • GFX_USE_DISPLAY_PANEL_TFT_800480_33_E (█) - 7 inch WVGA Truly TFT Display Board (AC164127-8)

GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E ()	<p>Specifies the Graphics Display Panel used.</p> <ul style="list-style-type: none"> • GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4) • GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q () - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6) • GFX_USE_DISPLAY_PANEL_TFT_640480_8_E () - 5.7 inch VGA Truly TFT Display Board (AC164127-8) • GFX_USE_DISPLAY_PANEL_TFT_800480_33_E () - 7 inch WVGA Truly TFT Display Board (AC164127-8)
---	--

Description

Specifies the Graphics Display Panel used.

6.1.5.5.1 GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q Macro**File**

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q
```

Overview

Specifies the Graphics Display Panel used.

- GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4)
- GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6)
- GFX_USE_DISPLAY_PANEL_TFT_640480_8_E () - 5.7 inch VGA Truly TFT Display Board (AC164127-8)
- GFX_USE_DISPLAY_PANEL_TFT_800480_33_E () - 7 inch WVGA Truly TFT Display Board (AC164127-8)

6.1.5.5.2 GFX_USE_DISPLAY_PANEL_TFT_640480_8_E Macro**File**

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_PANEL_TFT_640480_8_E
```

Overview

Specifies the Graphics Display Panel used.

- GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4)
- GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q () - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6)
- GFX_USE_DISPLAY_PANEL_TFT_640480_8_E - 5.7 inch VGA Truly TFT Display Board (AC164127-8)
- GFX_USE_DISPLAY_PANEL_TFT_800480_33_E () - 7 inch WVGA Truly TFT Display Board (AC164127-8)

6.1.5.5.3 GFX_USE_DISPLAY_PANEL_TFT_800480_33_E Macro**File**

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_PANEL_TFT_800480_33_E
```

Overview

Specifies the Graphics Display Panel used.

- GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4)
- GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q (🔗) - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6)
- GFX_USE_DISPLAY_PANEL_TFT_640480_8_E (🔗) - 5.7 inch VGA Truly TFT Display Board (AC164127-8)
- GFX_USE_DISPLAY_PANEL_TFT_800480_33_E - 7 inch WVGA Truly TFT Display Board (AC164127-8)

6.1.5.5.4 GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E Macro**File**

HardwareProfile.h

C

```
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
```

Overview

Specifies the Graphics Display Panel used.

- GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E - 3.2 inch QVGA Truly TFT Display Board (AC164127-4)
- GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q (🔗) - 4.3 inch WQVGA Powertip TFT Display Board (AC164127-6)
- GFX_USE_DISPLAY_PANEL_TFT_640480_8_E (🔗) - 5.7 inch VGA Truly TFT Display Board (AC164127-8)
- GFX_USE_DISPLAY_PANEL_TFT_800480_33_E (🔗) - 7 inch WVGA Truly TFT Display Board (AC164127-8)

6.1.5.6 Device Driver Options**Macros**

Name	Description
DISP_DATA_WIDTH (🔗)	Defines the display controller's physical interface to the display panel. Valid Values: <ul style="list-style-type: none"> • 1, 4, 8, 16, 18, 24 • 1, 4, 8 are usually used in MSTN and CSTN displays • 16, 18 and 24 are usually used in TFT displays.
DISP_ORIENTATION (🔗)	Defines the display rotation with respect to its native orientation. For example, if the display has a resolution specifications that says 240x320 (QVGA), the display is natively in portrait mode. If the application uses the display in landscape mode (320x240), then the orientation must be defined as 90 or 180 degree rotation. Graphics Library will calculate the actual pixel location to rotate the contents of the screen. So when users view the display, the image on the screen will come out in the correct orientation. Valid values: <ul style="list-style-type: none"> • 0 : display in its native orientation • 90 : rotated 90 degrees clockwise... more (🔗)
DISP_HOR_RESOLUTION (🔗)	Defines the native horizontal dimension of the screen. This is the horizontal pixel count given by the display's data sheet. For example a 320x240 display will have DISP_HOR_RESOLUTION of 320. Valid Values: <ul style="list-style-type: none"> • dependent on the display glass resolution used.

DISP_VER_RESOLUTION (█)	Defines the native vertical dimension of the screen. This is the vertical pixel count given by the display's data sheet. For example a 320x240 display will have DISP_VER_RESOLUTION of 240. Valid Values: <ul style="list-style-type: none"> • dependent on the display glass resolution used.
DISP_HOR_FRONT_PORCH (█)	Defines the horizontal front porch. DISP_HOR_BACK_PORCH (█) + DISP_HOR_FRONT_PORCH + DISP_HOR_PULSE_WIDTH (█) makes up the horizontal blanking period. Value used will be based on the display panel used.
DISP_HOR_BACK_PORCH (█)	Defines the horizontal back porch. DISP_HOR_BACK_PORCH + DISP_HOR_FRONT_PORCH (█) + DISP_HOR_PULSE_WIDTH (█) makes up the horizontal blanking period. Value used will be based on the display panel used.
DISP_VER_FRONT_PORCH (█)	Defines the vertical front porch. DISP_VER_BACK_PORCH (█) + DISP_VER_FRONT_PORCH + DISP_VER_PULSE_WIDTH (█) makes up the vertical blanking period. Value used will be based on the display panel used.
DISP_VER_BACK_PORCH (█)	Defines the vertical back porch. DISP_VER_BACK_PORCH + DISP_VER_FRONT_PORCH (█) + DISP_VER_PULSE_WIDTH (█) makes up the vertical blanking period. Value used will be based on the display panel used.
DISP_HOR_PULSE_WIDTH (█)	Defines the horizontal sync signal pulse width in pixels. Value used will be based on the display panel used.
DISP_VER_PULSE_WIDTH (█)	Defines the vertical sync signal pulse width in lines. Value used will be based on the display panel used.
DISP_INV_LSHIFT (█)	Indicates that the color data is sampled in the falling edge of the pixel clock.

Description

The options Graphics Hardware Platform, DISPLAY_CONTROLLER and DISPLAY_PANEL are specific to the hardware used. The Graphics Hardware Platform selects the Graphics PICtail™ Plus Board version, PIC24FJ256DA210 Development Board or any other Microchip demo boards for the Graphics Library. Currently there are two Graphics PICtail™ Plus Board versions supported as shown in the Getting Started (█) section.

The rest of the settings are used to specify the display parameters when using an RGB type display controller such as SSD1906 and SSD1926 from Solomon Systech. The table below summarizes the generic parameters found in RGB type display controllers and when each type is used.

Options	Color STN	Mono STN	TFT
DISP_DATA_WIDTH (█)	YES	YES	YES
DISP_ORIENTATION (█)	YES	YES	YES
DISP_HOR_RESOLUTION (█)	YES	YES	YES
DISP_VER_RESOLUTION (█)	YES	YES	YES
DISP_HOR_BACK_PORCH (█)	NO	NO	YES
DISP_HOR_FRONT_PORCH (█)	NO	NO	YES
DISP_VER_FRONT_PORCH (█)	NO	NO	YES
DISP_VER_BACK_PORCH (█)	NO	NO	YES
DISP_HOR_PULSE_WIDTH (█)	YES	YES	YES
DISP_VER_PULSE_WIDTH (█)	YES	YES	YES
DISP_INV_LSHIFT (█)	YES	YES	YES

All the options listed here are set in the HardwareProfile.h header file implemented in the application layer. An example of this file is shown in HardwareProfile.h Example section.

6.1.5.6.1 DISP_DATA_WIDTH Macro

File

HardwareProfile.h

C

```
#define DISP_DATA_WIDTH 18
```

Example

```
// define in Hardware Profile
#define DISP_DATA_WIDTH 18
```

Overview

Defines the display controller's physical interface to the display panel. Valid Values:

- 1, 4, 8, 16, 18, 24
- 1, 4, 8 are usually used in MSTN and CSTN displays
- 16, 18 and 24 are usually used in TFT displays.

6.1.5.6.2 DISP_ORIENTATION Macro

File

HardwareProfile.h

C

```
#define DISP_ORIENTATION 0
```

Example

```
// define in Hardware Profile
#define DISP_ORIENTATION 90
```

Overview

Defines the display rotation with respect to its native orientation. For example, if the display has a resolution specifications that says 240x320 (QVGA), the display is natively in portrait mode. If the application uses the display in landscape mode (320x240), then the orientation must be defined as 90 or 180 degree rotation. Graphics Library will calculate the actual pixel location to rotate the contents of the screen. So when users view the display, the image on the screen will come out in the correct orientation. Valid values:

- 0 : display in its native orientation
- 90 : rotated 90 degrees clockwise direction
- 180 : rotated 180 degrees clockwise direction
- 270 : rotated 270 degrees clockwise direction

6.1.5.6.3 DISP_HOR_RESOLUTION Macro

File

HardwareProfile.h

C

```
#define DISP_HOR_RESOLUTION 320
```

Example

```
// define in Hardware Profile
#define DISP_HOR_RESOLUTION 320
```

Overview

Defines the native horizontal dimension of the screen. This is the horizontal pixel count given by the display's data sheet. For example a 320x240 display will have DISP_HOR_RESOLUTION of 320. Valid Values:

- dependent on the display glass resolution used.

6.1.5.6.4 DISP_VER_RESOLUTION Macro

File

HardwareProfile.h

C

```
#define DISP_VER_RESOLUTION 240
```

Example

```
// define in Hardware Profile
#define DISP_VER_RESOLUTION 240
```

Overview

Defines the native vertical dimension of the screen. This is the vertical pixel count given by the display's data sheet. For example a 320x240 display will have DISP_VER_RESOLUTION of 240. Valid Values:

- dependent on the display glass resolution used.

6.1.5.6.5 DISP_HOR_FRONT_PORCH Macro

File

HardwareProfile.h

C

```
#define DISP_HOR_FRONT_PORCH 10
```

Overview

Defines the horizontal front porch. DISP_HOR_BACK_PORCH (■) + DISP_HOR_FRONT_PORCH + DISP_HOR_PULSE_WIDTH (■) makes up the horizontal blanking period. Value used will be based on the display panel used.

6.1.5.6.6 DISP_HOR_BACK_PORCH Macro

File

HardwareProfile.h

C

```
#define DISP_HOR_BACK_PORCH 20
```

Overview

Defines the horizontal back porch. DISP_HOR_BACK_PORCH + DISP_HOR_FRONT_PORCH (■) + DISP_HOR_PULSE_WIDTH (■) makes up the horizontal blanking period. Value used will be based on the display panel used.

6.1.5.6.7 DISP_VER_FRONT_PORCH Macro

File

HardwareProfile.h

C

```
#define DISP_VER_FRONT_PORCH 5
```

Overview

Defines the vertical front porch. DISP_VER_BACK_PORCH (§) + DISP_VER_FRONT_PORCH + DISP_VER_PULSE_WIDTH (§) makes up the vertical blanking period. Value used will be based on the display panel used.

6.1.5.6.8 DISP_VER_BACK_PORCH Macro

File

HardwareProfile.h

C

```
#define DISP_VER_BACK_PORCH 20
```

Overview

Defines the vertical back porch. DISP_VER_BACK_PORCH + DISP_VER_FRONT_PORCH (§) + DISP_VER_PULSE_WIDTH (§) makes up the vertical blanking period. Value used will be based on the display panel used.

6.1.5.6.9 DISP_HOR_PULSE_WIDTH Macro

File

HardwareProfile.h

C

```
#define DISP_HOR_PULSE_WIDTH 10
```

Overview

Defines the horizontal sync signal pulse width in pixels. Value used will be based on the display panel used.

6.1.5.6.10 DISP_VER_PULSE_WIDTH Macro

File

HardwareProfile.h

C

```
#define DISP_VER_PULSE_WIDTH 5
```

Overview

Defines the vertical sync signal pulse width in lines. Value used will be based on the display panel used.

6.1.5.6.11 DISP_INV_LSHIFT Macro

File

HardwareProfile.h

C

```
#define DISP_INV_LSHIFT 18
```

Overview

Indicates that the color data is sampled in the falling edge of the pixel clock.

6.1.5.7 HardwareProfile.h Example

This is an example of the HardwareProfile.h file implementation:

```
*****
* GetSystemClock() returns system clock frequency.
* GetPeripheralClock() returns peripheral clock frequency.
* GetInstructionClock() returns instruction clock frequency.
*****
```

```
*****
* Macro: #define      GetSystemClock()
* Overview: This macro returns the system clock frequency in Hertz.
*           * value is 8 MHz x 4 PLL for PIC24
*           * value is 8 MHz/2 x 18 PLL for PIC32
*****
```

```
#if defined(__PIC24F__)
#define GetSystemClock()      (32000000ul)
#elif defined(__PIC32MX__)
#define GetSystemClock()      (72000000ul)
#elif defined(__dsPIC33F__) || defined(__PIC24H__)
#define GetSystemClock()      (80000000ul)
#endif
```

```
*****
* Macro: #define      GetPeripheralClock()
* Overview: This macro returns the peripheral clock frequency
*           used in Hertz.
*           * value for PIC24 is <PRE>(GetSystemClock()/2) </PRE>
*           * value for PIC32 is <PRE>(GetSystemClock()/(1<<OSCCONbits.PBDIV)) </PRE>
*****
```

```
#if defined(__PIC24F__) || defined(__PIC24H__) || defined(__dsPIC33F__)
#define GetPeripheralClock()    (GetSystemClock() / 2)
#elif defined(__PIC32MX__)
#define GetPeripheralClock()    (GetSystemClock() / (1 << OSCCONbits.PBDIV))
#endif
```

```
*****
* Macro: #define      GetInstructionClock()
* Overview: This macro returns instruction clock frequency
*           used in Hertz.
*           * value for PIC24 is <PRE>(GetSystemClock()/2) </PRE>
*           * value for PIC32 is (GetSystemClock() / PFMWSbits.CHECON) </PRE>
*****
```

```
#if defined(__PIC24F__) || defined(__PIC24H__) || defined(__dsPIC33F__)
#define GetInstructionClock()  (GetSystemClock() / 2)
#elif defined(__PIC32MX__)
#define GetInstructionClock()  (GetSystemClock() / PFMWSbits.CHECON)
#endif
```

```
//Auto Generated Code
#define PIC24FJ256DA210_DEV_BOARD
#define USE_16BIT_PMP
#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
#define GFX_CCLK_DIVIDER 61
#define GFX_DISPLAY_BUFFER_START_ADDRESS 0x00020000ul
#define GFX_DISPLAY_BUFFER_LENGTH 0x00025800ul
#define GFX_EPMP_CS1_BASE_ADDRESS 0x00020000ul
#define GFX_EPMP_CS1_MEMORY_SIZE 0x40000ul
//End Auto Generated Code
```

```
*****
* External Memory Programmer Settings
*****
```

```
#if defined (EXPLORER_16)
#define USE_COMM_PKT_MEDIA_SERIAL_PORT
#define BAUDRATE2          115200UL
```

```

#define BRG_DIV2          4
#define BRGH2             1
#else
#define USE_COMM_PKT_MEDIA_USB

///#define USE_SELF_POWER_SENSE_IO
#define tris_self_power    TRISAbits.TRISA2      // Input
#define self_power         1

///#define USE_USB_BUS_SENSE_IO
#define tris_usb_bus_sense TRISBbits.TRISB5      // Input
#define USB_BUS_SENSE      U1OTGSTATbits.SESVD // Special considerations required if
using SESVD for this purpose. See documentation.

#endif

#define COMM_PKT_RX_MAX_SIZE   (1024)

.....

```

6.2 Graphics Object Layer API

Description of Graphics Object Layer API.

6.2.1 GOL Objects

The Graphics Object Layer (GOL) contains the Advanced Graphics Objects or commonly known as widgets.

Enumerations

Name	Description
GOL_OBJ_TYPE (🔗)	This structure defines the Object types used in the library.

Modules

Name	Description
Analog Clock (🔗)	Analog Clock is an Object that emulates an analog clock with moving hands. It can be used with or without a bitmap image as the background source.
Button (🔗)	Button is an Object that emulates a press and release effect when operated upon.
Chart (🔗)	Chart is an Object that draws a bar chart or a pie chart representation of a single data or series of data.
Checkbox (🔗)	Check Box is an Object that simulates a check box on paper. Usually it is used as an option setting where the checked or filled state means the option is enabled and the unfilled or unchecked state means the option is disabled.
Round Dial (🔗)	Dial is an Object that can be used to display emulate a turn dial that can both go in clockwise or counterclockwise.
Digital Meter (🔗)	DigitalMeter is an Object that can be used to display a value of a sampled variable. This Object is ideal when fast refresh of the value is needed. The Object refreshes only the digits that needs to change. A limitation of this Object is that the font used should have equal character widths.
Edit Box (🔗)	Edit Box is is an Object that emulates a cell or a text area that can be edited dynamically.
Grid (🔗)	Grid is an Object that draws a grid on the screen with each cell capable of displaying an image or a string.
Group Box (🔗)	Group Box is an Object that can be used to group Objects together in the screen.

List Box (█)	List Box is an Object that defines a scrollable area where items are listed. User can select a single item or set of items.
Meter (█)	Meter is an Object that can be used to graphically display a sampled input.
Picture Control (█)	Picture is an Object that can be used to transform a bitmap to be an Object in the screen and have control on the bitmap rendering. This object can be used to create animation using a series of bitmaps.
Progress Bar (█)	Progress Bar (█) is an Object that can be used to display the progress of a task such as a file download or transfer.
Radio Button (█)	Radio Button (█) is an Object that can be used to offer set of choices to the user. Only one of the choices is selectable. Changing selection automatically removes the selection on the previous option.
Slider/Scrollbar (█)	Slider or Scrollbar is an Object that can be used to display a value or scrolling location in a predefined area.
Static Text (█)	Static Text is an Object that can be used to display a single or multi-line string of text in a predefined location.
Text Entry (█)	Text Entry is an Object that can be used to emulate a key pad entry with a display area for the entered characters. The Object has a feature where you can define a key to reply with a translated message that signifies a command key was pressed. A command key example can be your enter or carriage return key or an escape key. Multiple keys can be assigned command keys. Application can utilize the command key to define the behavior of the program based on a command key press.
Window (█)	Window is an Object that can be used to encapsulate objects into a group. Unlike the Group Box Object, the Window Object has additional features such as displaying an icon or a small bitmap on its Title Bar (█). It also has additional controls for both Title Bar (█) and Client Area.

Structures

Name	Description
OBJ_HEADER (█)	This structure defines the first nine fields of the Objects structure. This allows generic operations on library Objects.

Types

Name	Description
DRAW_FUNC (█)	object draw function pointer typedef
FREE_FUNC (█)	object free function pointer typedef
MSG_DEFAULT_FUNC (█)	object default message function pointer typedef
MSG_FUNC (█)	object message function pointer typedef

Description

All the GOL objects that will be shown in the display have a corresponding data structure that keeps and maintains its parameters.

Each object type has a set of functions that enables the user to change the state of the object. The Microchip graphics library supports the following set of GOL objects.

Object	Type Definition
Button (█)	OBJ_BUTTON
Chart (█)	OBJ_CHART
Checkbox (█)	OBJ_CHECKBOX
Dial (█)	OBJ_ROUNDDIAL
Digital Meter (█)	OBJ_DIGITALMETER
Edit Box (█)	OBJ_EDITBOX
Grid (█)	OBJ_GRID

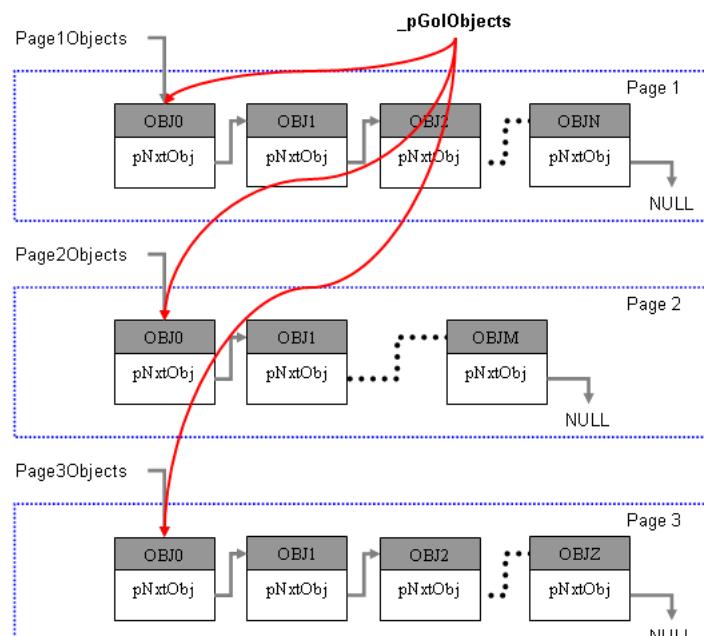
Group Box (█)	OBJ_GROUPBOX
List Box (█)	OBJ_LISTBOX
Meter (█)	OBJ_METER
Picture (█)	OBJ_PICTURE
Progress Bar (█)	OBJ_PROGRESSBAR
Radio Button (█)	OBJ_RADIOBUTTON
Slider (█)	OBJ_SLIDER
Static Text (█)	OBJ_STATICTEXT
Text Entry (█)	OBJ_TEXTENTRY
Window (█)	OBJ_WINDOW

Each GOL object type uses a style scheme. The style scheme defines the font and color settings. User can create new style schemes and assign it to objects. Multiple style schemes can be used for different objects.

To efficiently manage the different objects the first 9 fields of the structure for each object are defined the same. Collectively they are referred to as the object header structure (OBJ_HEADER (█)). Defining the OBJ_HEADER (█) enables the common APIs to manage the objects of different types in the same manner.

The GOL operation is centered on two major processes. First is the rendering process and second is the messaging process. These processes make use of linked list objects. The field pNxtObj in OBJ_HEADER (█) makes this possible. To manage the list a global pointer _pGolObjects (█) is utilized. It defines the current active linked list. Newly created objects are automatically added to the end of this list. The active linked list is the list that the messaging and rendering operation parses to evaluate if the messages received affect the objects and if objects need to be redrawn.

The use of the _pGolObjects (█) pointer also provides a way to easily manage objects. In applications where multiple pages are displayed on the screen, objects can be grouped according to pages. The pointer can be manipulated to switch from one page to another depending on the page currently shown on the screen.



Implementation of Multi-Page Objects

User can remove an object from a list by pointer manipulation. Objects that are removed from the list are not accessible by the rendering and messaging processes.

6.2.1.1 GOL_OBJ_TYPE Enumeration

File

GOL.h

C

```
typedef enum {
    OBJ_BUTTON,
    OBJ_WINDOW,
    OBJ_CHECKBOX,
    OBJ_RADIOBUTTON,
    OBJ_EDITBOX,
    OBJ_LISTBOX,
    OBJ_SLIDER,
    OBJ_PROGRESSBAR,
    OBJ_STATICTEXT,
    OBJ_PICTURE,
    OBJ_GROUPBOX,
    OBJ_CUSTOM,
    OBJ_ROUNDDIAL,
    OBJ_METER,
    OBJ_GRID,
    OBJ_CHART,
    OBJ_TEXTENTRY,
    OBJ_DIGITALMETER,
    OBJ_ANALOGCLOCK,
    OBJ_UNKNOWN
} GOL_OBJ_TYPE;
```

Members

Members	Description
OBJ_BUTTON	Type defined for Button (█) Object.
OBJ_WINDOW	Type defined for Window (█) Object.
OBJ_CHECKBOX	Type defined for Check Box Object.
OBJ_RADIOBUTTON	Type defined for Radio Button (█) Object.
OBJ_EDITBOX	Type defined for Edit Box Object.
OBJ_LISTBOX	Type defined for List Box Object.
OBJ_SLIDER	Type defined for Slider (█) and/or Scroll (█) Bar (█) Object.
OBJ_PROGRESSBAR	Type defined for Progress Object.
OBJ_STATICTEXT	Type defined for Static Text Object.
OBJ_PICTURE	Type defined for Picture (█) or Bitmap Object.
OBJ_GROUPBOX	Type defined for Group Box Object.
OBJ_CUSTOM	Type defined for Custom Object.
OBJ_ROUNDDIAL	Type defined for Dial (█) Object.
OBJ_METER	Type defined for Meter (█) Object.
OBJ_GRID	Type defined for Grid (█) Object.
OBJ_CHART	Type defined for Chart (█) Object.
OBJ_TEXTENTRY	Type defined for Text-Entry Object.
OBJ_DIGITALMETER	Type defined for DIGITALMETER (█) Object.
OBJ_ANALOGCLOCK	Type defined for ANALOGCLOCK (█) Object.
OBJ_UNKNOWN	Type is undefined and not supported by the library.

Overview

This structure defines the Object types used in the library.

6.2.1.2 OBJ_HEADER Structure

File

GOL.h

C

```
typedef struct {
    WORD ID;
    void * pNxtObj;
    GOL_OBJ_TYPE type;
    WORD state;
    SHORT left;
    SHORT top;
    SHORT right;
    SHORT bottom;
    GOL_SCHEME * pGolScheme;
    DRAW_FUNC DrawObj;
    FREE_FUNC FreeObj;
    MSG_FUNC MsgObj;
    MSG_DEFAULT_FUNC MsgDefaultObj;
} OBJ_HEADER;
```

Members

Members	Description
WORD ID;	Unique id assigned for referencing.
void * pNxtObj;	A pointer to the next object.
GOL_OBJ_TYPE type;	Identifies the type of GOL object.
WORD state;	State of object.
SHORT left;	Left position of the Object.
SHORT top;	Top position of the Object.
SHORT right;	Right position of the Object.
SHORT bottom;	Bottom position of the Object.
GOL_SCHEME * pGolScheme;	Pointer to the scheme used.
DRAW_FUNC DrawObj;	function pointer to the object draw function
FREE_FUNC FreeObj;	function pointer to the object free function
MSG_FUNC MsgObj;	function pointer to the object message function
MSG_DEFAULT_FUNC MsgDefaultObj;	function pointer to the object default message function

Overview

This structure defines the first nine fields of the Objects structure. This allows generic operations on library Objects.

6.2.1.3 DRAW_FUNC Type

File

GOL.h

C

```
typedef WORD (* DRAW_FUNC)(void *);
```

Description

object draw function pointer typedef

6.2.1.4 FREE_FUNC Type

File

GOL.h

C

```
typedef void (* FREE_FUNC)(void *);
```

Description

object free function pointer typedef

6.2.1.5 MSG_DEFAULT_FUNC Type

File

GOL.h

C

```
typedef void (* MSG_DEFAULT_FUNC)(WORD, void *, GOL_MSG *);
```

Description

object default message function pointer typedef

6.2.1.6 MSG_FUNC Type

File

GOL.h

C

```
typedef WORD (* MSG_FUNC)(void *, GOL_MSG *);
```

Description

object message function pointer typedef

6.2.1.7 Analog Clock

Analog Clock is an Object that emulates an analog clock with moving hands. It can be used with or without a bitmap image as the background source.

Functions

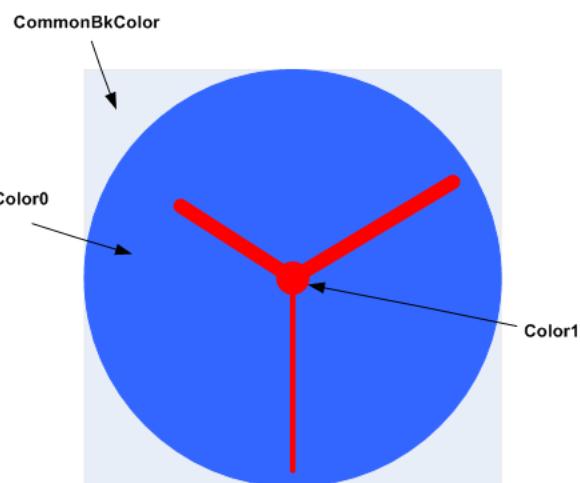
	Name	Description
≡	AcCreate (█)	This function creates an Analog Clock object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
≡	AcDraw (█)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.
≡	AcSetHour (█)	Sets the hour value of the analog clock.
≡	AcSetMinute (█)	Sets the minute value of the analog clock.
≡	AcSetSecond (█)	Sets the second value of the analog clock.

Structures

Name	Description
ANALOGCLOCK (█)	Defines the parameters required for a clock Object. The following relationships of the parameters determines the general shape of the clock: 1. centerx and centery determine the middle of the clock. 2. radius defines the radius of the clock. 4. *pBitmap points to the background image for the analog clock.

Description

The Analog Clock object is rendered using the assigned style scheme. The following figure illustrates the color assignments for the clock.



6.2.1.7.1 Analog Clock States

Macros

Name	Description
AC_DRAW (█)	Bit to indicate button must be redrawn.
AC_DISABLED (█)	Bit for disabled state.
AC_HIDE (█)	Bit to indicate button must be removed from screen.
AC_PRESSED (█)	Bit for press state.
AC_TICK (█)	Bit to tick second hand
UPDATE_HOUR (█)	Bit to indicate hour hand must be redrawn
UPDATE_MINUTE (█)	Bit to indicate minute hand must be redrawn
UPDATE_SECOND (█)	Bit to indicate minute hand must be redrawn

Module

Analog Clock (█)

Description

List of Analog Clock bit states.

6.2.1.7.1.1 AC_DRAW Macro

File

AnalogClock.h

C

```
#define AC_DRAW 0x4000 // Bit to indicate button must be redrawn.
```

Description

Bit to indicate button must be redrawn.

6.2.1.7.1.2 AC_DISABLED Macro

File

AnalogClock.h

C

```
#define AC_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.7.1.3 AC_HIDE Macro

File

AnalogClock.h

C

```
#define AC_HIDE 0x8000 // Bit to indicate button must be removed from screen.
```

Description

Bit to indicate button must be removed from screen.

6.2.1.7.1.4 AC_PRESSED Macro

File

AnalogClock.h

C

```
#define AC_PRESSED 0x0004 // Bit for press state.
```

Description

Bit for press state.

6.2.1.7.1.5 AC_TICK Macro

File

AnalogClock.h

C

```
#define AC_TICK 0x1000 // Bit to tick second hand
```

Description

Bit to tick second hand

6.2.1.7.1.6 UPDATE_HOUR Macro

File

AnalogClock.h

C

```
#define UPDATE_HOUR 0x2000 // Bit to indicate hour hand must be redrawn
```

Description

Bit to indicate hour hand must be redrawn

6.2.1.7.1.7 UPDATE_MINUTE Macro**File**

AnalogClock.h

C

```
#define UPDATE_MINUTE 0x0100 // Bit to indicate minute hand must be redrawn
```

Description

Bit to indicate minute hand must be redrawn

6.2.1.7.1.8 UPDATE_SECOND Macro**File**

AnalogClock.h

C

```
#define UPDATE_SECOND 0x0200 // Bit to indicate minute hand must be redrawn
```

Description

Bit to indicate minute hand must be redrawn

6.2.1.7.2 AcCreate Function**File**

AnalogClock.h

C

```
ANALOGCLOCK * AcCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    SHORT hour,
    SHORT minute,
    SHORT radius,
    BOOL sechand,
    WORD state,
    void * pBitmap,
    GOL_SCHEME * pscheme
);
```

Module

Analog Clock ([🔗](#))

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.

SHORT bottom	Bottom most position of the object.
SHORT hour	Initial hour value.
SHORT minute	Initial minute value.
SHORT radius	Sets the radius of the clock.
BOOL sechand	Flag to indicate if the second hand will be drawn or not.
WORD state	Sets the initial state of the object.
void * pBitmap	Pointer to the bitmap used on the face of the analog clock dimension of the image must match the dimension of the widget.
GOL_SCHEME * pScheme	Pointer to the style scheme used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
WORD state;

pScheme = GOLCreateScheme();
state = AC_DRAW;

AnalogClock = AcCreate(ANALOGCLOCK_ID, 20, 64, 50, 118, 1, 20, 30, FALSE, state, NULL,
pScheme);
```

Overview

This function creates an Analog Clock object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
ANALOGCLOCK ( ) *AcCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT radius, void
*pBitmap, XCHAR ( ) *pText, GOL_SCHEME ( ) *pScheme)
```

6.2.1.7.3 AcDraw Function

File

AnalogClock.h

C

```
WORD AcDraw(
    void * pAc
);
```

Module

Analog Clock ()

Input Parameters

Input Parameters	Description
void * pAc	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

```
void MyGOLDraw() {
    static OBJ_HEADER *pCurrentObj = NULL;
    int done;

    // There are no objects
    if(GOLGetList() == NULL)
        return;

    // If it's last object jump to head
    if(pCurrentObj == NULL)
        pCurrentObj = GOLGetList();

    done = 0;

    // this only process Button and Window
    while(pCurrentObj != NULL){
        // check if object state indicates redrawing
        if(pCurrentObj->state&0xFC00) {
            switch(pCurrentObj->type){
                case OBJ_ANALOGCLOCK:
                    done = AcDraw((ANALOGCLOCK*)pCurrentObj);
                    break;
                case OBJ_WINDOW:
                    done = WndDraw((WINDOW*)pCurrentObj);
                    break;
                default:
                    done = 1;
                    break;
            }
            if(done){
                // reset only the state if drawing was finished
                pCurrentObj->state = 0;
            }else{
                // done processing the list
                return;
            }
        }
        // go to next object
        pCurrentObj = pCurrentObj->pNxtObj;
    }
}
```

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

Syntax

WORD AcDraw(ANALOGCLOCK (■) *pAc)

6.2.1.7.4 AcSetHour Function

File

AnalogClock.h

C

```
void AcSetHour(
    ANALOGCLOCK * pAc,
    SHORT hour
);
```

Module

Analog Clock (

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT hour	New hour time.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets the hour value of the analog clock.

Syntax

```
AcSetHour(ANALOGCLOCK () *pAc, SHORT hour)
```

6.2.1.7.5 AcSetMinute Function

File

AnalogClock.h

C

```
void AcSetMinute(
    ANALOGCLOCK * pAc,
    SHORT minute
);
```

Module

Analog Clock (

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT minute	New minute time.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets the minute value of the analog clock.

Syntax

```
AcSetMinute(ANALOGCLOCK (■) *pAc, SHORT minute)
```

6.2.1.7.6 AcSetSecond Function

File

AnalogClock.h

C

```
void AcSetSecond(
    ANALOGCLOCK * pAc,
    SHORT second
);
```

Module

Analog Clock (■)

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT second	New second time.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets the second value of the analog clock.

Syntax

```
AcSetSecond(ANALOGCLCOK *pAc, SHORT second)
```

6.2.1.7.7 ANALOGCLOCK Structure

File

AnalogClock.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT radius;
    SHORT centerx;
    SHORT centery;
```

```

    SHORT values;
    SHORT prev_values;
    SHORT valueM;
    SHORT prev_valueM;
    SHORT valueH;
    SHORT prev_valueH;
    void * pBitmap;
} ANALOGCLOCK;

```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
SHORT radius;	Radius of the clock.
SHORT centerx;	center location in X-axis.
SHORT centery;	center location in Y-axis.
SHORT valueS;	time in Second
SHORT prev_valueS;	previous time in Second
SHORT valueM;	time in Minute
SHORT prev_valueM;	previous time in Minute
SHORT valueH;	time in Hour
SHORT prev_valueH;	previous time in Hour
void * pBitmap;	Pointer to bitmap used.

Module

Analog Clock (■)

Overview

Defines the parameters required for a clock Object. The following relationships of the parameters determines the general shape of the clock:

1. centerx and centery determine the middle of the clock.
2. radius defines the radius of the clock.
4. *pBitmap points to the background image for the analog clock.

6.2.1.8 Button

Button is an Object that emulates a press and release effect when operated upon.

Functions

	Name	Description
💡	BtnCreate (■)	This function creates a BUTTON (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
💡	BtnDraw (■)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. The text on the face of the button is drawn on top of the bitmap. Text is always rendered centered on the face of the button. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid... more (■)
💡	BtnSetText (■)	This function sets the string used for the object.
💡	BtnMsgDefault (■)	This function performs the actual state change based on the translated message given. The following state changes are supported:

	<code>BtnTranslateMsg ()</code>	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.
--	----------------------------------	--

Macros

Name	Description
<code>BtnGetText ()</code>	This macro returns the address of the current text string used for the object.
<code>BtnGetBitmap ()</code>	This macro returns the location of the currently used bitmap for the object.
<code>BtnSetBitmap ()</code>	This macro sets the bitmap used in the object. The size of the bitmap must match the face of the button.

Structures

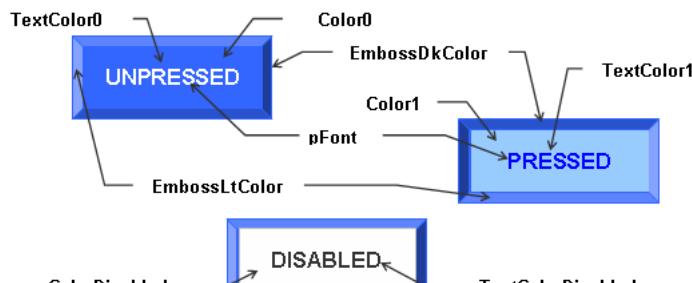
Name	Description
<code>BUTTON ()</code>	Defines the parameters required for a button Object. The following relationships of the parameters determines the general shape of the button: <ol style="list-style-type: none"> 1. Width is determined by right - left. 2. Height is determined by top - bottom. 3. Radius - specifies if the button will have a rounded edge. If zero then the button will have sharp (cornered) edge. 4. If $2 * \text{radius} = \text{height} = \text{width}$, the button is a circular button.

Description

Button supports Keyboard and Touchscreen inputs, replying to their events with the following messages:

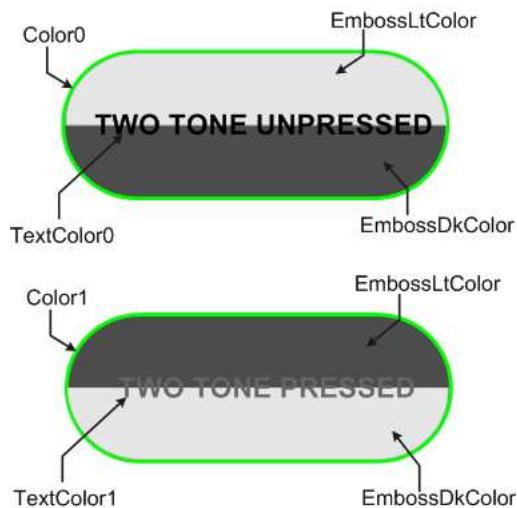
1. `BTN_MSG_PRESSED`
2. `BTN_MSG_RELEASED`

The button object is rendered using the assigned style scheme. The following figure illustrates the color assignments for the button.



`CommonBkColor` – used to hide the button on the screen.

A variant of the button widget, is to implement it as a two tone button. The button object is rendered using the modified color assignments in the style scheme. The two tone button is rendered when the `BTN_TWOTONE ()` object state bit is set.



Two Tone Button

When using images on the button widget, the image is drawn on the center of the widget. If the image size is larger than the widget dimension, the image will cover the widget. When this happens, it is recommended to disable the rendering of the button panel so no time is spent rendering the panel which will be covered by the image. To do this enable the `BTN_NOPANEL` (■) object state bit.

The text or string used with the button widget is always rendered last. Even with images, the string will always be rendered on top of the image. Text rendering on the button widget is not affected by `BTN_NOPANEL` (■) object state bit. Text will only be affected by the following object state bits `BTN_TEXTRIGHT` (■), `BTN_TEXTLEFT` (■), `BTN_TEXTBOTTOM` (■), and `BTN_TEXTTOP` (■).

To enable multiple lines of text on the button widget, enable the `USE_BUTTON_MULTI_LINE` (■) in the `GraphicsConfig.h`. Enabling the `USE_BUTTON` (■) will only enable single lines of text.

6.2.1.8.1 Button States

Macros

Name	Description
<code>BTN_DISABLED</code> (■)	Bit for disabled state.
<code>BTN_DRAW</code> (■)	Bit to indicate button must be redrawn.
<code>BTN_DRAW_FOCUS</code> (■)	Bit to indicate focus must be redrawn.
<code>BTN_FOCUSED</code> (■)	Bit for focus state.
<code>BTN_HIDE</code> (■)	Bit to indicate button must be removed from screen.
<code>BTN_PRESSED</code> (■)	Bit for press state.
<code>BTN_TEXTBOTTOM</code> (■)	Bit to indicate text is top aligned.
<code>BTN_TEXTLEFT</code> (■)	Bit to indicate text is left aligned.
<code>BTN_TEXTRIGHT</code> (■)	Bit to indicate text is right aligned.
<code>BTN_TEXTTOP</code> (■)	Bit to indicate text is bottom aligned.
<code>BTN_TOGGLE</code> (■)	Bit to indicate button will have a toggle behavior.
<code>BTN_TWOTONE</code> (■)	Bit to indicate the button is a two tone type.
<code>BTN_NOPANEL</code> (■)	Bit to indicate the button will be drawn without a panel (for faster drawing when the button image used is larger than the button panel).

Module

`Button` (■)

Description

List of Button (■) bit states.

6.2.1.8.1.1 **BTN_DISABLED** Macro

File

Button.h

C

```
#define BTN_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.8.1.2 **BTN_DRAW** Macro

File

Button.h

C

```
#define BTN_DRAW 0x4000 // Bit to indicate button must be redrawn.
```

Description

Bit to indicate button must be redrawn.

6.2.1.8.1.3 **BTN_DRAW_FOCUS** Macro

File

Button.h

C

```
#define BTN_DRAW_FOCUS 0x2000 // Bit to indicate focus must be redrawn.
```

Description

Bit to indicate focus must be redrawn.

6.2.1.8.1.4 **BTN_FOCUSED** Macro

File

Button.h

C

```
#define BTN_FOCUSED 0x0001 // Bit for focus state.
```

Description

Bit for focus state.

6.2.1.8.1.5 **BTN_HIDE** Macro

File

Button.h

C

```
#define BTN_HIDE 0x8000 // Bit to indicate button must be removed from screen.
```

Description

Bit to indicate button must be removed from screen.

6.2.1.8.1.6 **BTN_PRESSED** Macro

File

Button.h

C

```
#define BTN_PRESSED 0x0004 // Bit for press state.
```

Description

Bit for press state.

6.2.1.8.1.7 **BTN_TEXTBOTTOM** Macro

File

Button.h

C

```
#define BTN_TEXTBOTTOM 0x0040 // Bit to indicate text is top aligned.
```

Description

Bit to indicate text is top aligned.

6.2.1.8.1.8 **BTN_TEXTLEFT** Macro

File

Button.h

C

```
#define BTN_TEXTLEFT 0x0020 // Bit to indicate text is left aligned.
```

Description

Bit to indicate text is left aligned.

6.2.1.8.1.9 **BTN_TEXTRIGHT** Macro

File

Button.h

C

```
#define BTN_TEXTRIGHT 0x0010 // Bit to indicate text is right aligned.
```

Description

Bit to indicate text is right aligned.

6.2.1.8.1.10 **BTN_TEXTTOP** Macro

File

Button.h

C

```
#define BTN_TEXTTOP 0x0080 // Bit to indicate text is bottom aligned.
```

Description

Bit to indicate text is bottom aligned.

6.2.1.8.1.11 BTN_TOGGLE Macro**File**

Button.h

C

```
#define BTN_TOGGLE 0x0008 // Bit to indicate button will have a toggle behavior.
```

Description

Bit to indicate button will have a toggle behavior.

6.2.1.8.1.12 BTN_TWOTONE Macro**File**

Button.h

C

```
#define BTN_TWOTONE 0x0100 // Bit to indicate the button is a two tone type.
```

Description

Bit to indicate the button is a two tone type.

6.2.1.8.1.13 BTN_NOPANEL Macro**File**

Button.h

C

```
#define BTN_NOPANEL 0x0200 // Bit to indicate the button will be drawn without a panel  
(for faster drawing when the button image used is larger than the button panel).
```

Description

Bit to indicate the button will be drawn without a panel (for faster drawing when the button image used is larger than the button panel).

6.2.1.8.2 BtnCreate Function**File**

Button.h

C

```
BUTTON * BtnCreate(  
    WORD ID,  
    SHORT left,  
    SHORT top,  
    SHORT right,  
    SHORT bottom,  
    SHORT radius,  
    WORD state,  
    void * pBitmap,  
    XCHAR * pText,  
    GOL_SCHEME * pscheme  
) ;
```

Module

Button (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
SHORT radius	Radius of the rounded edge.
WORD state	Sets the initial state of the object.
void * pBitmap	Pointer to the bitmap used on the face of the button dimension of the bitmap must match the dimension of the button.
XCHAR * pText	Pointer to the text of the button.
GOL_SCHEME * pScheme	Pointer to the style scheme used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```

GOL_SCHEME *pScheme;
BUTTON *buttons[3];
WORD state;

pScheme = GOLCreateScheme();
state = BTN_DRAW;

buttons[0] = BtnCreate(1,20,64,50,118,0, state, NULL, "ON", pScheme);
// check if button 0 is created
if (buttons[0] == NULL)
    return 0;

buttons[1] = BtnCreate(2,52,64,82,118,0, state, NULL, "OFF", pScheme);
// check if button 1 is created
if (buttons[1] == NULL)
    return 0;

buttons[2] = BtnCreate(3,84,64,114,118,0, state, NULL, "HI", pScheme);
// check if button 2 is created
if (buttons[2] == NULL)
    return 0;

return 1;

```

Overview

This function creates a BUTTON (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
BUTTON (█) *BtnCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT radius, void *pBitmap,
XCHAR (█) *pText, GOL_SCHEME (█) *pScheme)
```

6.2.1.8.3 BtnDraw Function

File

Button.h

C

```
WORD BtnDraw(
    void * pObj
);
```

Module

Button (█)

Input Parameters

Input Parameters	Description
pB	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

```
void MyGOLDraw() {
    static OBJ_HEADER *pCurrentObj = NULL;
    int done;

    // There are no objects
    if(GOLGetList() == NULL)
        return;

    // If it's last object jump to head
    if(pCurrentObj == NULL)
        pCurrentObj = GOLGetList();

    done = 0;

    // this only process Button and Window
    while(pCurrentObj != NULL){
        // check if object state indicates redrawing
        done = pCurrentObj->draw(pCurrentObj);

        if(done){
            // reset only the state if drawing was finished
            pCurrentObj->state = 0;
        }else{
            // done processing the list
            return;
        }
    }
    // go to next object
    pCurrentObj = pCurrentObj->pNxtObj;
}
```

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the button is drawn on top of the bitmap. Text is always rendered centered on the face of the button.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD BtnDraw(void *pObj)
```

6.2.1.8.4 BtnGetText Macro

File

Button.h

C

```
#define BtnGetText(pB) ((BUTTON *)pB)->pText
```

Module

Button (█)

Input Parameters

Input Parameters	Description
pB	Pointer to the object.

Side Effects

none

Returns

Returns pointer to the text string being used.

Preconditions

none

Example

```
XCHAR *pChar;
BUTTON Button[2];

pChar = BtnGetText(Button[0]);
```

Overview

This macro returns the address of the current text string used for the object.

Syntax

```
BtnGetText(pB)
```

6.2.1.8.5 BtnSetText Function

File

Button.h

C

```
void BtnSetText(
    BUTTON * pB,
```

```
XCHAR * pText  
) ;
```

Module

Button (█)

Input Parameters

Input Parameters	Description
BUTTON * pB	The pointer to the object whose text will be modified.
XCHAR * pText	Pointer to the text that will be used.

Side Effects

none

Returns

none

Preconditions

none

Example

```
XCHAR Label0[] = "ON";  
XCHAR Label1[] = "OFF";  
BUTTON Button[2];  
  
BtnSetText(Button[0], Label0);  
BtnSetText(Button[1], Label1);
```

Overview

This function sets the string used for the object.

Syntax

```
BtnSetText(BUTTON (█) *pB, XCHAR (█) *pText)
```

6.2.1.8.6 BtnGetBitmap Macro

File

Button.h

C

```
#define BtnGetBitmap(pB) ((BUTTON *)pB)->pBitmap
```

Module

Button (█)

Input Parameters

Input Parameters	Description
pB	Pointer to the object.

Side Effects

none

Returns

Returns the pointer to the current bitmap used.

Preconditions

none

Example

```
BUTTON *pButton;
BITMAP_FLASH *pUsedBitmap;

pUsedBitmap = BtnGetBitmap(pButton);
```

Overview

This macro returns the location of the currently used bitmap for the object.

Syntax

```
BtnGetBitmap(pB)
```

6.2.1.8.7 BtnSetBitmap Macro**File**

Button.h

C

```
#define BtnSetBitmap(pB, pBitmap) ((BUTTON *)pB)->pBitmap = pBitmap
```

Module

Button ()

Input Parameters

Input Parameters	Description
pB	Pointer to the object.
pBitmap	Pointer to the bitmap to be used.

Side Effects

none

Returns

none

Preconditions

none

Example

```
extern BITMAP_FLASH myIcon;
BUTTON *pButton;

BtnSetBitmap(pButton, &myIcon);
```

Overview

This macro sets the bitmap used in the object. The size of the bitmap must match the face of the button.

Syntax

```
BtnSetBitmap(pB, pBitmap)
```

6.2.1.8.8 BtnMsgDefault Function**File**

Button.h

C

```
void BtnMsgDefault(
    WORD translatedMsg,
```

```

    void * pObj,
    GOL_MSG * pMsg
);

```

Module

Button (█)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
void * pObj	The pointer to the object whose state will be modified.
GOL_MSG * pMsg	The pointer to the GOL message.

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg (█)() example.

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
BTN_MSG_PRESSED	Touch Screen, Keyboard	Set BTN_PRESSED (█)	Button (█) will be redrawn in the pressed state.
BTN_MSG_RELEASED	Touch Screen, Keyboard	Clear BTN_PRESSED (█)	Button (█) will be redrawn in the unpressed state.
BTN_MSG_CANCELPRESS	Touch Screen,	Clear BTN_PRESSED (█)	Button (█) will be redrawn in the unpressed state.

Syntax

BtnMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (█)* pMsg)

6.2.1.8.9 BtnTranslateMsg Function**File**

Button.h

C

```

WORD BtnTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);

```

Module

Button (█)

Input Parameters

Input Parameters	Description
void * pObj	The pointer to the object where the message will be evaluated to check if the message will affect the object.
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- BTN_MSG_PRESSED – Button (■) is pressed
- BTN_MSG_RELEASED – Button (■) is released
- BTN_MSG_CANCELPRESS – Button (■) will be released, user cancels press action on the button
- OBJ_MSG_INVALID – Button (■) is not affected

Preconditions

none

Example

```
void MyGOLMsg (GOL_MSG *pMsg) {
    OBJ_HEADER *pCurrentObj;
    WORD objMsg;

    if(pMsg->uiEvent == EVENT_INVALID)
        return;
    pCurrentObj = GOLGetList();

    while(pCurrentObj != NULL){
        // If the object must be redrawn
        // It cannot accept message
        if(!IsObjUpdated(pCurrentObj)){
            translatedMsg = pCurrentObj->MsgObj(pCurrentObj, pMsg);

            if(translatedMsg != OBJ_MSG_INVALID)
            {
                if(GOLMsgCallback(translatedMsg, pCurrentObj, pMsg))
                    if(pCurrentObj->MsgDefaultObj)
                        pCurrentObj->MsgDefaultObj(translatedMsg, pCurrentObj, pMsg);
            }
        }
        pCurrentObj = pCurrentObj->pNxtObj;
    }
}
```

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
BTN_MSG_PRESSED	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the face of the button while the button is not pressed.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED (■) or SCAN_SPACE_PRESSED (■) while the button is not pressed.

BTN_MSG_STILLPRESSED	Touch Screen	EVENT_STILLPRESS	If event occurs and the x,y position does not change from the previous press position in the face of the button.
BTN_MSG_RELEASED	Touch Screen	EVENT_RELEASE	If the event occurs and the x,y position falls in the face of the button while the button is pressed.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_RELEASED (█) or SCAN_SPACE_RELEASED (█) while the button is pressed.
BTN_MSG_CANCELPRESS	Touch Screen	EVENT_MOVE	If the event occurs outside the face of the button and the button is currently pressed.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
BtnTranslateMsg(void *pObj, GOL_MSG (█) *pMsg)
```

6.2.1.8.10 BUTTON Structure**File**

Button.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT radius;
    SHORT textWidth;
    SHORT textHeight;
    XCHAR * pText;
    void * pBitmap;
    GFX_COLOR previousAlphaColor;
} BUTTON;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (█)).
SHORT radius;	Radius for rounded buttons.
SHORT textWidth;	Computed text width, done at creation.
SHORT textHeight;	Computed text height, done at creation.
XCHAR * pText;	Pointer to the text used.
void * pBitmap;	Pointer to bitmap used.

Module

Button (█)

Overview

Defines the parameters required for a button Object. The following relationships of the parameters determines the general shape of the button:

1. Width is determined by right - left.
2. Height is determined by top - bottom.
3. Radius - specifies if the button will have a rounded edge. If zero then the button will have sharp (cornered) edge.
4. If $2 \times \text{radius} = \text{height} = \text{width}$, the button is a circular button.

6.2.1.9 Chart

Chart is an Object that draws a bar chart or a pie chart representation of a single data or series of data.

Functions

Name	Description
ChCreate ()	This function creates a CHART () object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
ChDraw ()	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. The colors of the bars of the bar chart or sectors of the pie chart can be the default color table or user defined color table set by ChSetColorTable ()() function. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is... more ()
ChAddDataSeries ()	This function creates a DATASERIES () object and populates the structure with the given parameters.
ChRemoveDataSeries ()	This function removes DATASERIES () object from the list of DATASERIES () objects and frees the memory used of that removed object. The position of the object to be removed is specified by the number parameter. If the list has only one member, it removes the member regardless of the number given.
ChSetValueRange ()	This function sets the minimum and maximum range of values that the bar chart will show. The criteria is that min <= max.
ChSetPercentRange ()	This function sets the minimum and maximum range of percentage that the bar chart will show. The criteria is that min <= max. This affects bar charts only and CH_PERCENTAGE bit state is set.
ChSetSampleRange ()	This function sets the sample start and sample end when drawing the chart. Together with the data series' SHOW_DATA () flags the different way of displaying the chart data is achieved.
ChFreeDataSeries ()	This function removes DATASERIES () object from the list of DATASERIES () objects and frees the memory used of that removed object.
ChTranslateMsg ()	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
ChShowSeries ()	This macro sets the specified data series number show flag to be set to SHOW_DATA ().
ChHideSeries ()	This macro sets the specified data series number show flag to be set to HIDE_DATA ().
ChGetShowSeriesCount ()	This macro shows the number of data series that has its show flag set to SHOW_DATA ().
ChGetShowSeriesStatus ()	This macro returns the show ID status of the DATASERIES ().
ChSetValueLabel ()	This macro sets the address of the current text string used for the value axis label of the bar chart.
ChGetValueLabel ()	This macro returns the address of the current text string used for the value axis label of the bar chart.
ChGetValueMax ()	This macro returns the current maximum value that will be drawn for bar charts.
ChGetValueMin ()	This macro returns the current minimum value that will be drawn for bar charts.
ChGetValueRange ()	This macro gets the current range for bar charts. The value returned is calculated from the current (valMax - valMin) set. To get the minimum use ChGetValueMin ()() and to get the maximum use ChGetValueMax ()().

ChSetSampleLabel (█)	This macro sets the address of the current text string used for the sample axis label of the bar chart.
ChGetSampleLabel (█)	This macro returns the address of the current text string used for the sample axis label of the bar chart.
ChGetSampleStart (█)	This macro returns the sampling start value.
ChGetSampleEnd (█)	This macro returns the sampling end value.
ChGetPercentRange (█)	This macro gets the percentage range for bar charts. The value returned is calculated from percentage max - min. To get the minimum use ChGetPercentMin (█)() and to get the maximum use ChGetPercentMax (█)().
ChGetSampleRange (█)	This macro gets the sample range for pie or bar charts. The value returned is calculated from smplEnd - smplStart.
ChGetPercentMax (█)	This macro returns the current maximum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.
ChGetPercentMin (█)	This macro returns the current minimum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.
ChSetColorTable (█)	This macro sets the color table used to draw the data in pie and bar charts.
ChGetColorTable (█)	This macro returns the current color table used for the pie and bar charts.
ChSetTitle (█)	This macro sets the address of the current text string used for the title of the chart.
ChGetTitle (█)	This macro returns the address of the current text string used for the title of the chart.
ChSetTitleFont (█)	This macro sets the location of the font used for the title of the chart.
ChGetTitleFont (█)	This macro returns the location of the font used for the title of the chart.
ChGetAxisLabelFont (█)	This macro returns the location of the font used for the X and Y axis labels of the chart.
ChSetAxisLabelFont (█)	This macro sets the location of the font used for the X and Y axis labels of the chart.
ChGetGridLabelFont (█)	This macro returns the location of the font used for the X and Y axis grid labels of the chart.
ChSetGridLabelFont (█)	This macro sets the location of the font used for the X and Y axis grid labels of the chart.

Structures

Name	Description
CHART (█)	Defines the parameters required for a chart Object.
DATASERIES (█)	Defines a variable for the CHART (█) object. It specifies the number of samples, pointer to the array of samples for the data series and pointer to the next data series. A member of this structure (show) is used as a flag to determine if the series is to be drawn or not. Together with the smplStart and smplEnd it will determine what kind of chart will be drawn.
CHARTPARAM (█)	Defines the parameters for the CHART (█) object.

Description

It supports only Keyboard inputs, replying to any touch screen events with the message: CH_MSG_SELECTED.

The Chart Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.

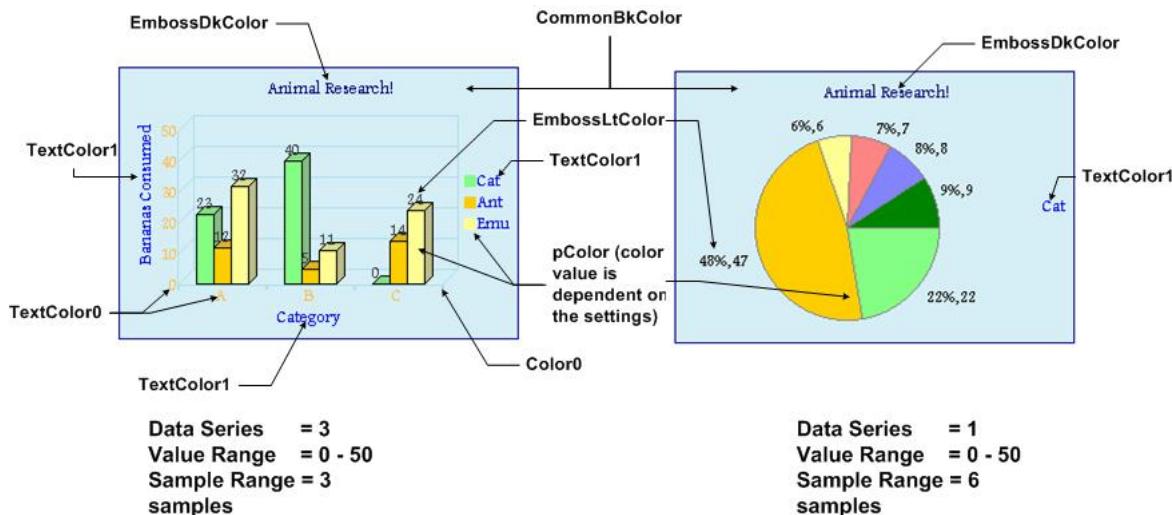


Chart Terminologies

1. Value Axis - This is the vertical range of a chart for normal bar charts and horizontal range of the chart for horizontally drawn bar charts. In most cases this axis will represent values (\$ amounts), temperatures, or other numeric data.
2. Sample Axis - This is the horizontal range of a chart for normal bar charts and vertical range of the chart for horizontally drawn bar charts. In most cases this axis will represent categories, such as months, sample segments, or other non-numeric data.
3. Title - The text used to define the Title of the chart.
4. Data Points (or the sample points) These are the individual points where a value is graphed, as a point on a line, a bar, or a pie slice.
5. Data Series - A complete series of data, distinguished by the same color and type of sample point.
6. Legend - Labels that indicate how each data series is displayed on the chart. Each color represents a different data series. For pie charts with only one data series shown, each color represents one sector or one sample point.
7. Data Sample Range - The scale for the data sample axis. Example: months from January to December. Internally, this range is represented by:
 - Numeric Sequence 1, 2, 3, ... and so on
 - Alphabet Sequence A, B, C, .. and so on.
8. Value Range - The scale for the value axis. Example: range of numbers from the lowest to the highest to be charted.

6.2.1.9.1 Chart States

Macros

Name	Description
CH_DISABLED (█)	Bit for disabled state.
CH_DRAW (█)	Bit to indicate chart must be redrawn.
CH_DRAW_DATA (█)	Bit to indicate data portion of the chart must be redrawn.
CH_3D_ENABLE (█)	Bit to indicate that bar charts are to be drawn with 3-D effect
CH_BAR (█)	Bit to indicate the chart is type bar. If both PIE and BAR types are set BAR type has higher priority.
CH_BAR_HOR (█)	These bits (with CH_BAR (█) bit set), sets the bar chart to be drawn horizontally.
CH_DONUT (█)	These bits (with CH_PIE (█) bit set), sets the pie chart to be drawn in a donut shape.
CH_LEGEND (█)	Bit to indicate that legend is to be shown. Usable only when seriesCount > 1.

CH_NUMERIC (█)	This bit is used only for bar charts. If this bit is set, it indicates that the bar chart labels for variables are numeric. If this bit is not set, it indicates that the bar chart labels for variables are alphabets.
CH_PERCENT (█)	Bit to indicate that the pie chart will be drawn with percentage values shown for the sample data. For bar chart, if CH_VALUE (█) is set, it toggles the value shown to percentage.
CH_PIE (█)	Bit to indicate the chart is type pie. If both PIE and BAR types are set BAR type has higher priority.
CH_VALUE (█)	Bit to indicate that the values of the bar chart data or pie chart data are to be shown
CH_HIDE (█)	Bit to indicate chart must be removed from screen.

Module

Chart (█)

Description

List of Chart (█) bit states.

6.2.1.9.1.1 CH_DISABLED Macro**File**

Chart.h

C

#define CH_DISABLED 0x0002 // Bit for disabled state.

Description

Bit for disabled state.

6.2.1.9.1.2 CH_DRAW Macro**File**

Chart.h

C

#define CH_DRAW 0x4000 // Bit to indicate chart must be redrawn.

Description

Bit to indicate chart must be redrawn.

6.2.1.9.1.3 CH_DRAW_DATA Macro**File**

Chart.h

C

#define CH_DRAW_DATA 0x2000 // Bit to indicate data portion of the chart must be redrawn.

Description

Bit to indicate data portion of the chart must be redrawn.

6.2.1.9.1.4 CH_3D_ENABLE Macro**File**

Chart.h

C

```
#define CH_3D_ENABLE 0x0008      // Bit to indicate that bar charts are to be drawn with
3-D effect
```

Description

Bit to indicate that bar charts are to be drawn with 3-D effect

6.2.1.9.1.5 CH_BAR Macro**File**

Chart.h

C

```
#define CH_BAR 0x0200      // Bit to indicate the chart is type bar. If both PIE and BAR
types are set BAR type has higher priority.
```

Description

Bit to indicate the chart is type bar. If both PIE and BAR types are set BAR type has higher priority.

6.2.1.9.1.6 CH_BAR_HOR Macro**File**

Chart.h

C

```
#define CH_BAR_HOR 0x0240      // These bits (with CH_BAR bit set), sets the bar chart to
be drawn horizontally.
```

Description

These bits (with CH_BAR (■) bit set), sets the bar chart to be drawn horizontally.

6.2.1.9.1.7 CH_DONUT Macro**File**

Chart.h

C

```
#define CH_DONUT 0x0140      // These bits (with CH_PIE bit set), sets the pie chart to be
drawn in a donut shape.
```

Description

These bits (with CH_PIE (■) bit set), sets the pie chart to be drawn in a donut shape.

6.2.1.9.1.8 CH_LEGEND Macro**File**

Chart.h

C

```
#define CH_LEGEND 0x0001      // Bit to indicate that legend is to be shown. Usable only
when seriesCount > 1.
```

Description

Bit to indicate that legend is to be shown. Usable only when seriesCount > 1.

6.2.1.9.1.9 CH_NUMERIC Macro

File

Chart.h

C

```
#define CH_NUMERIC 0x0080      // This bit is used only for bar charts. If this bit is set,  
it indicates that the
```

Description

This bit is used only for bar charts. If this bit is set, it indicates that the bar chart labels for variables are numeric. If this bit is not set, it indicates that the bar chart labels for variables are alphabets.

6.2.1.9.1.10 CH_PERCENT Macro

File

Chart.h

C

```
#define CH_PERCENT 0x0010      // Bit to indicate that the pie chart will be drawn with  
percentage values shown for the sample data.
```

Description

Bit to indicate that the pie chart will be drawn with percentage values shown for the sample data. For bar chart, if CH_VALUE (■) is set, it toggles the value shown to percentage.

6.2.1.9.1.11 CH_PIE Macro

File

Chart.h

C

```
#define CH_PIE 0x0100      // Bit to indicate the chart is type pie. If both PIE and BAR  
types are set BAR type has higher priority.
```

Description

Bit to indicate the chart is type pie. If both PIE and BAR types are set BAR type has higher priority.

6.2.1.9.1.12 CH_VALUE Macro

File

Chart.h

C

```
#define CH_VALUE 0x0004      // Bit to indicate that the values of the bar chart data or  
pie chart data are to be shown
```

Description

Bit to indicate that the values of the bar chart data or pie chart data are to be shown

6.2.1.9.1.13 CH_HIDE Macro

File

Chart.h

C

```
#define CH_HIDE 0x8000      // Bit to indicate chart must be removed from screen.
```

Description

Bit to indicate chart must be removed from screen.

6.2.1.9.2 Data Series Status Settings

Macros

Name	Description
HIDE_DATA (█)	Macro used to reset the data series show flag or indicate that the data series will not be shown when the chart is drawn.
SHOW_DATA (█)	Macro used to set the data series show flag or indicate that the data series will be shown when the chart is drawn.

Module

Chart (█)

Description

Data Series show status flag settings.

6.2.1.9.2.1 HIDE_DATA Macro

File

Chart.h

C

```
#define HIDE_DATA 0           // Macro used to reset the data series show flag or
                           // indicate that the data series will not be shown when the chart is drawn.
```

Description

Macro used to reset the data series show flag or indicate that the data series will not be shown when the chart is drawn.

6.2.1.9.2.2 SHOW_DATA Macro

File

Chart.h

C

```
#define SHOW_DATA 1          // Macro used to set the data series show flag or
                           // indicate that the data series will be shown when the chart is drawn.
```

Description

Macro used to set the data series show flag or indicate that the data series will be shown when the chart is drawn.

6.2.1.9.3 Chart Examples

Examples of generated bar charts based on settings.

Module

Chart (█)

Description

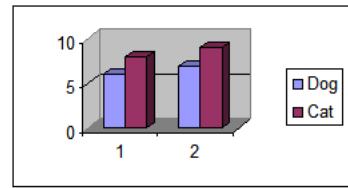
The following are some examples of settings and output chart that will be generated.

Example 1

```
CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series
```

ChSetSampleRange(1,2);

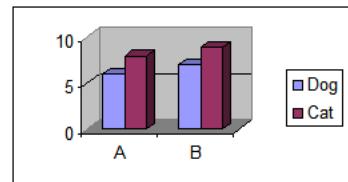
	Sample 1	Sample 2	Show
Dog	6	7	x
Cat	8	9	x

**Example 2**

```
CH_NUMERIC = 0
SERIESDATA Series
SERIESDATA Series
```

ChSetSampleRange(1,2);

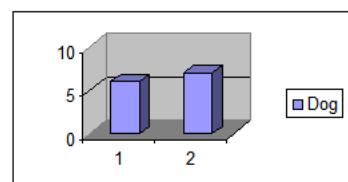
	Sample 1	Sample 2	Show
Dog	6	7	x
Cat	8	9	x

**Example 3**

```
CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series
```

ChSetSampleRange(1,2);

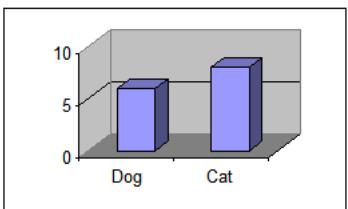
	Sample 1	Sample 2	Show
Dog	6	7	x
Cat	8	9	

**Example 4**

```
CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series
```

ChSetSampleRange(1,1);

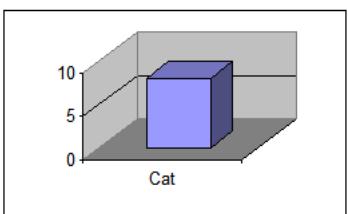
	Sample 1	Sample 2	Show
Dog	6	7	x
Cat	8	9	x

**Example 5**

```
CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series
```

ChSetSampleRange(1,1);

	Sample 1	Sample 2	Show
Dog	6	7	
Cat	8	9	x

**More Customization...**

```
CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series
```

ChSetSampleRange(1,1);

	Sample 1	Sample 2	Show
Dog	6	7	x
Cat	8	9	x

smplStart smplEnd

*pTitle
*pSmplLabel
*pValLabel

Animal Trend
Year
Number of Animal

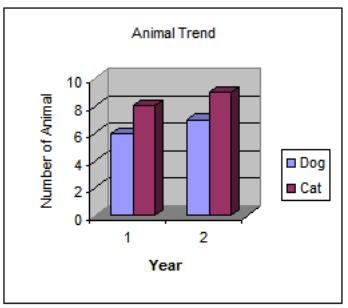
**6.2.1.9.4 ChCreate Function****File**

Chart.h

C

```
CHART * ChCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    DATASERIES * pData,
    CHARTPARAM * pParam,
    GOL_SCHEME * pScheme
);
```

Module

Chart (

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
DATASERIES * pData	Pointer to the data for the contents of the chart. NULL can be assigned initially when creating the object.
CHARTPARAM * pParam	Pointer to the chart parameters. NULL can be assigned initially when creating the object and the chart parameters can be populated using the API provided.
GOL_SCHEME * pScheme	Pointer to the style scheme used. When set to NULL, the default style scheme will be used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
extern const FONT_FLASH GOLSsmallFont;
extern const FONT_FLASH GOLMediumFont;

// Note that strings are declared as such to cover cases
// where XCHAR type is declared as short (2 bytes).
XCHAR ChTitle[] = {'E','x','a','m','p','l','e',0};
XCHAR SampleName[] = {'C','a','t','e','g','o','r','y',0};
XCHAR ValueName[] = {'#','H','i','t','s',0};
XCHAR SeriesName[2] = {
    {'V','1',0},
    {'V','2',0},
};
V1Data[2] = { 50, 100 };
V2Data[2] = { 5, 10 };

GOL_SCHEME *pScheme;
CHART *pChart;
CHARTPARAM Contents;
WORD state;

pScheme = GOLCreateScheme();
```

```

state = CH_BAR|CH_DRAW|CH_BAR_HOR; // Bar Chart to be drawn with horizontal orientation

pChart = ChCreate(0x01, // ID
                  0, 0, // dimensions
                  GetMaxX(), // state of the chart
                  GetMaxY(), // data not initialized yet
                  state, // no parameters yet
                  NULL, // style scheme used
                  NULL,
                  pScheme);

if (pMyChart == NULL) // check if chart was allocated memory
    return 0;

ChSetTitleFont(pChart, (void*)&GOLMediumFont);
ChSetTitle(pChart, ChTitle); // set the title

// set the grid labels and axis labels font
ChSetGridLabelFont(pChart, (void*)&GOLS SmallFont);
ChSetAxisLabelFont(pChart, (void*)&GOLS SmallFont);

// set the labels for the X and Y axis
ChSetSampleLabel(pChart, (XCHAR*)SampleName);
ChSetValueLabel(pChart, (XCHAR*)ValueName);

ChAddDataSeries(pChart, 2, V1Data, (XCHAR*)SeriesName[0]);
ChAddDataSeries(pChart, 2, V2Data, (XCHAR*)SeriesName[1]);

// set the range of the sample values
ChSetValueRange(pChart, 0, 100);

// show all two samples to be displayed (start = 1, end = 2)
ChSetSampleRange(pChart, 1, 2);

GOLDraw(); // draw the chart

```

Overview

This function creates a CHART (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
CHART (■) *ChCreate( WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, DATASERIES (■) *pData, CHARTPARAM (■) *pParam, GOL_SCHEME (■) *pScheme)
```

6.2.1.9.5 ChDraw Function

File

Chart.h

C

```
WORD ChDraw(
    void * pObj
);
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object to be rendered.

Side Effects

none.

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The colors of the bars of the bar chart or sectors of the pie chart can be the default color table or user defined color table set by ChSetColorTable (■)() function.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD ChDraw(void *pObj)
```

6.2.1.9.6 ChAddDataSeries Function**File**

Chart.h

C

```
DATASERIES * ChAddDataSeries(
    CHART * pCh,
    WORD nSamples,
    WORD * pData,
    XCHAR * pName
);
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
CHART * pCh	Pointer to the chart object.
WORD nSamples	The number of samples or data points.
WORD * pData	Pointer to the array of samples or data points.
XCHAR * pName	Pointer to the string used to label the data series.

Side Effects

Appends to the list of DATASERIES (■) that the chart is operating on. By default, the show flag of the newly added data series is set to SHOW_DATA (■) or enabled.

Returns

Returns the pointer to the data variable (DATASERIES (■)) object created. If NULL is returned, the addition of the new object failed due to not enough memory for the object.

Preconditions

none

Example

See ChCreate (§) example.

Overview

This function creates a DATASERIES (§) object and populates the structure with the given parameters.

Syntax

```
ChAddDataSeries(CHART (§) *pCh, WORD nSamples, WORD *pData, XCHAR (§) *pName)
```

6.2.1.9.7 ChRemoveDataSeries Function

File

Chart.h

C

```
void ChRemoveDataSeries(
    CHART * pCh,
    WORD number
);
```

Module

Chart (§)

Input Parameters

Input Parameters	Description
CHART * pCh	Pointer to the chart object.
WORD number	The position of the object to be removed in the list where the first object in the list is assigned a value of 1. If this parameter is set to zero, the whole list of DATA_SERIES is removed.

Side Effects

none.

Returns

none.

Preconditions

none

Example

```
void ClearChartData(CHART *pCh) {
    if(pCh->pChData != NULL)
        // remove the all data series
        ChRemoveDataSeries(pCh, 0);
}
```

Overview

This function removes DATASERIES (§) object from the list of DATASERIES (§) objects and frees the memory used of that removed object. The position of the object to be removed is specified by the number parameter. If the list has only one member, it removes the member regardless of the number given.

Syntax

```
ChRemoveDataSeries(CHART (§) *pCh, WORD number)
```

6.2.1.9.8 ChShowSeries Macro

File

Chart.h

C

```
#define ChShowSeries(pCh, seriesNum) (ChSetDataSeries(pCh, seriesNum, SHOW_DATA))
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the chart object.
seriesNum	The data series number that will be modified. If this number is zero, all the entries' flag in the list will be set to SHOW_DATA (█).

Side Effects

none

Returns

Returns the same passed number if successful otherwise -1 if unsuccessful.

Preconditions

none

Example

```
// from the example in ChCreate() we change the items to be shown when
// GOLDraw() is called.

// reset all data series to be HIDE_DATA
ChHideSeries(pMyChart, 0);
// set data series 1 (V1Data) to be shown
ChShowSeries(pMyChart, 1);
// draw the chart
GOLDraw();
.....
*
```

Overview

This macro sets the specified data series number show flag to be set to SHOW_DATA (█).

Syntax

SHORT ChShowSeries(CHART (█) *pCh, WORD seriesNum)

6.2.1.9.9 ChHideSeries Macro

File

Chart.h

C

```
#define ChHideSeries(pCh, seriesNum) (ChSetDataSeries(pCh, seriesNum, HIDE_DATA))
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the chart object.
seriesNum	The data series number that will be modified. If this number is zero, all the entries' flag in the list will be set to HIDE_DATA (█).

Side Effects

none

Returns

Returns the same passed number if successful otherwise -1 if unsuccessful.

Preconditions

none

Example

See ChShowSeries (█)() example.

Overview

This macro sets the specified data series number show flag to be set to HIDE_DATA (█).

Syntax

```
SHORT ChHideSeries(CHART (█) *pCh, WORD seriesNum)
```

6.2.1.9.10 ChGetShowSeriesCount Macro

File

Chart.h

C

```
#define ChGetShowSeriesCount(pCh) (pCh->prm.seriesCount)
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the number of data series with its show flag set to SHOW_DATA (█).

Preconditions

none

Overview

This macro shows the number of data series that has its show flag set to SHOW_DATA (█)

Syntax

```
ChGetShowSeriesCount(pCh)
```

6.2.1.9.11 ChGetShowSeriesStatus Macro

File

Chart.h

C

```
#define ChGetShowSeriesStatus(pDSeries) (pDSeries->show)
```

Module

Chart ([🔗](#))

Input Parameters

Input Parameters	Description
pDSeries	Pointer to the data series(DATASERIES (🔗)) that is being checked.

Side Effects

none

Returns

Returns the status of the show flag. 1 - (SHOW_DATA ([🔗](#))) means that the show status flag is set. 0 - (HIDE_DATA ([🔗](#))) means that the show status flag is not set.

Preconditions

none

Overview

This macro returns the show ID status of the DATASERIES ([🔗](#)).

Syntax

```
ChGetShowSeriesStatus(pDSeries)
```

6.2.1.9.12 ChSetValueLabel Macro

File

Chart.h

C

```
#define ChSetValueLabel(pCh, pNewValueLabel) (((CHART *)pCh)->prm.pValLabel = pNewValueLabel)
```

Module

Chart ([🔗](#))

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewYLabel	pointer to the string to be used as an value axis label of the bar chart.

Side Effects

none

Returns

none.

Preconditions

none

Example

See ChCreate (■) example.

Overview

This macro sets the address of the current text string used for the value axis label of the bar chart.

Syntax

```
ChSetValueLabel(pCh, pNewYLabel)
```

6.2.1.9.13 ChGetValueLabel Macro**File**

Chart.h

C

```
#define ChGetValueLabel(pCh) (((CHART *)pCh)->prm.pValLabel)
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the pointer to the current value axis label text of the bar chart.

Preconditions

none

Overview

This macro returns the address of the current text string used for the value axis label of the bar chart.

Syntax

```
ChGetValueLabel(pCh)
```

6.2.1.9.14 ChGetValueMax Macro**File**

Chart.h

C

```
#define ChGetValueMax(pCh) (pCh->prm.valMax)
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the maximum value set when bar charts are drawn.

Preconditions

none

Overview

This macro returns the current maximum value that will be drawn for bar charts.

Syntax

```
ChGetValueMax(pCh)
```

6.2.1.9.15 ChGetValueMin Macro

File

Chart.h

C

```
#define ChGetValueMin(pCh) (pCh->prm.valMin)
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the minimum value set when bar charts are drawn.

Preconditions

none

Overview

This macro returns the current minimum value that will be drawn for bar charts.

Syntax

```
ChGetValueMin(pCh)
```

6.2.1.9.16 ChSetValueRange Function

File

Chart.h

C

```
void ChSetValueRange(
    CHART * pCh,
    WORD min,
    WORD max
);
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
CHART * pCh	Pointer to the chart object.
WORD min	Minimum value that will be displayed in the bar chart.
WORD max	Maximum value that will be displayed in the bar chart.

Side Effects

none.

Returns

none.

Preconditions

none

Overview

This function sets the minimum and maximum range of values that the bar chart will show. The criteria is that min <= max.

Syntax

```
ChSetValueRange(CHART (■) *pCh, WORD min, WORD max)
```

6.2.1.9.17 ChGetValueRange Macro

File

Chart.h

C

```
#define ChGetValueRange(pCh) (pCh->prm.valMax - pCh->prm.valMin)
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
pCh	Pointer to the chart object.

Side Effects

none.

Returns

Value range computed from valMax-valMin.

Preconditions

none

Overview

This macro gets the current range for bar charts. The value returned is calculated from the current (valMax - valMin) set. To get the minimum use ChGetValueMin (■)() and to get the maximum use ChGetValueMax (■)().

Syntax

```
ChGetValueRange(pCh)
```

6.2.1.9.18 ChSetSampleLabel Macro

File

Chart.h

C

```
#define ChSetSampleLabel(pCh, pNewXLabel) (((CHART *)pCh)->prm.pSmplLabel = pNewXLabel)
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewXLabel	pointer to the string to be used as an sample axis label of the bar chart.

Side Effects

none

Returns

none.

Preconditions

none

Example

See ChCreate (█)() example.

Overview

This macro sets the address of the current text string used for the sample axis label of the bar chart.

Syntax

```
ChSetSampleLabel(pCh, pNewXLabel)
```

6.2.1.9.19 ChGetSampleLabel Macro

File

Chart.h

C

```
#define ChGetSampleLabel(pCh) (((CHART *)pCh)->prm.pSmplLabel)
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the pointer to the current sample axis label text of the bar chart.

Preconditions

none

Overview

This macro returns the address of the current text string used for the sample axis label of the bar chart.

Syntax

```
ChGetSampleLabel(pCh)
```

6.2.1.9.20 ChGetSampleStart Macro**File**

Chart.h

C

```
#define ChGetSampleStart(pCh) (((CHART *)pCh)->prm.smplStart)
```

Module

Chart (

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the sample start point.

Preconditions

none

Overview

This macro returns the sampling start value.

Syntax

```
ChGetSampleStart(pCh)
```

6.2.1.9.21 ChGetSampleEnd Macro**File**

Chart.h

C

```
#define ChGetSampleEnd(pCh) ((CHART *)pCh)->prm.smplEnd
```

Module

Chart (

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the sample end point.

Preconditions

none

Overview

This macro returns the sampling end value.

Syntax

```
ChGetSampleEnd(pCh)
```

6.2.1.9.22 ChSetPercentRange Function

File

Chart.h

C

```
void ChSetPercentRange(
    CHART * pCh,
    WORD min,
    WORD max
);
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
CHART * pCh	Pointer to the chart object.
WORD min	Minimum percentage value that will be displayed in the bar chart.
WORD max	Maximum percentage value that will be displayed in the bar chart.

Side Effects

none.

Returns

none.

Preconditions

none

Overview

This function sets the minimum and maximum range of percentage that the bar chart will show. The criteria is that min <= max. This affects bar charts only and CH_PERCENTAGE bit state is set.

Syntax

```
ChSetPercentRange(CHART (█) *pCh, WORD min, WORD max)
```

6.2.1.9.23 ChGetPercentRange Macro

File

Chart.h

C

```
#define ChGetPercentRange(pCh) (pCh->prm.perMax - pCh->prm.perMin)
```

ModuleChart ([🔗](#))**Input Parameters**

Input Parameters	Description
pCh	Pointer to the chart object.

Side Effects

none.

Returns

Percentage range computed from max-min.

Preconditions

none

Overview

This macro gets the percentage range for bar charts. The value returned is calculated from percentage max - min. To get the minimum use ChGetPercentMin ([🔗](#)()) and to get the maximum use ChGetPercentMax ([🔗](#)()).

Syntax

ChGetPercentRange(pCh)

6.2.1.9.24 ChSetSampleRange Function

File

Chart.h

C

```
void ChSetSampleRange(
    CHART * pCh,
    WORD start,
    WORD end
);
```

ModuleChart ([🔗](#))**Input Parameters**

Input Parameters	Description
CHART * pCh	Pointer to the chart object.
WORD start	Start point of the data samples to be displayed.
WORD end	End point of the data samples to be displayed.

Side Effects

none.

Returns

none.

Preconditions

none

Example

See ChCreate ([🔗](#)()) example.

Overview

This function sets the sample start and sample end when drawing the chart. Together with the data series' SHOW_DATA (■) flags the different way of displaying the chart data is achieved.

Start & End Value	The # of Data Series Flag Set	Chart (■) Description
Start <= End	1	Show the data indicated by Start and End points of the DATASERIES (■) with the flag set
Start = End	1	Show the data indicated by Start or End points of the DATASERIES (■) with the flag set
Start, End = > 1		Show the data indicated by Start point of the DATASERIES (■) with the flag set. Each samples of all checked data series are grouped together according to sample number.

Syntax

```
ChSetSampleRange(CHART (■) *pCh, WORD start, WORD end)
```

6.2.1.9.25 ChGetSampleRange Macro

File

Chart.h

C

```
#define ChGetSampleRange(pCh) (ChGetSampleEnd(pCh) - ChGetSampleStart(pCh))
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
pCh	Pointer to the chart object.

Side Effects

none.

Returns

Sample range computed from smplEnd - smplStart.

Preconditions

none

Overview

This macro gets the sample range for pie or bar charts. The value returned is calculated from smplEnd - smplStart.

Syntax

```
ChGetSampleRange(pCh)
```

6.2.1.9.26 ChGetPercentMax Macro

File

Chart.h

C

```
#define ChGetPercentMax(pCh) (pCh->prm.perMax)
```

Module

Chart ([Diagram](#))

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the maximum percentage value set when bar charts are drawn.

Preconditions

none

Overview

This macro returns the current maximum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.

Syntax

```
ChGetPercentMax(pCh)
```

6.2.1.9.27 ChGetPercentMin Macro

File

Chart.h

C

```
#define ChGetPercentMin(pCh) (pCh->prm.perMin)
```

Module

Chart ([Diagram](#))

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the minimum percentage value when bar charts are drawn.

Preconditions

none

Overview

This macro returns the current minimum value of the percentage range that will be drawn for bar charts when CH_PERCENTAGE bit state is set.

Syntax

```
ChGetPercentMin(pCh)
```

6.2.1.9.28 ChSetColorTable Macro

File

Chart.h

C

```
#define ChSetColorTable(pCh, pNewTable) (((CHART *)pCh)->prm.pColor) = pNewTable
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewTable	Pointer to the color table that will be used.

Side Effects

none

Returns

none.

Preconditions

none

Overview

This macro sets the color table used to draw the data in pie and bar charts.

Syntax

```
ChSetColorTable(pCh, pNewTable)
```

6.2.1.9.29 ChGetColorTable Macro

File

Chart.h

C

```
#define ChGetColorTable(pCh) (((CHART *)pCh)->prm.pColor)
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the address of the color table used.

Preconditions

none

Overview

This macro returns the current color table used for the pie and bar charts.

Syntax

```
ChGetColorTable(pCh)
```

6.2.1.9.30 ChSetTitle Macro**File**

Chart.h

C

```
#define ChSetTitle(pCh, pNewTitle) (((CHART *)pCh)->prm.pTitle = pNewTitle)
```

Module

Chart ()

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewTitle	pointer to the string to be used as a title of the chart.

Side Effects

none

Returns

none.

Preconditions

none

Example

See ChCreate ()() example.

Overview

This macro sets the address of the current text string used for the title of the chart.

Syntax

```
ChSetTitle(pCh, pNewTitle)
```

6.2.1.9.31 ChGetTitle Macro**File**

Chart.h

C

```
#define ChGetTitle(pCh) (((CHART *)pCh)->prm.pTitle)
```

Module

Chart ()

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the pointer to the current title text used.

Preconditions

none

Overview

This macro returns the address of the current text string used for the title of the chart.

Syntax

```
ChGetTitle(pCh)
```

6.2.1.9.32 ChSetTitleFont Macro**File**

Chart.h

C

```
#define ChSetTitleFont(pCh, pNewFont) (((CHART *)pCh)->prm.pTitleFont = pNewFont)
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewFont	Pointer to the font used.

Side Effects

none

Returns

none.

Preconditions

none

Example

See ChCreate (█)() example.

Overview

This macro sets the location of the font used for the title of the chart.

Syntax

```
ChSetTitleFont(pCh, pNewFont)
```

6.2.1.9.33 ChGetTitleFont Macro**File**

Chart.h

C

```
#define ChGetTitleFont(pCh) (((CHART *)pCh)->prm.pTitleFont)
```

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the address of the current font used for the title text.

Preconditions

none

Overview

This macro returns the location of the font used for the title of the chart.

Syntax

ChGetTitleFont(pCh)

6.2.1.9.34 ChGetAxisLabelFont Macro

File

Chart.h

C

#define ChGetAxisLabelFont(pCh) (((CHART *)pCh)->prm.pAxisLabelsFont)

Module

Chart (█)

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the address of the current font used for the title text.

Preconditions

none

Overview

This macro returns the location of the font used for the X and Y axis labels of the chart.

Syntax

ChGetAxisLabelFont(pCh)

6.2.1.9.35 ChSetAxisLabelFont Macro

File

Chart.h

C

```
#define ChSetAxisLabelFont(pCh, pNewFont) (((CHART *)pCh)->prm.pAxisLabelsFont = pNewFont)
```

Module

Chart (

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewFont	Pointer to the font used.

Side Effects

none

Returns

none.

Preconditions

none

Example

See ChCreate () example.

Overview

This macro sets the location of the font used for the X and Y axis labels of the chart.

Syntax

```
ChSetAxisLabelFont(pCh, pNewFont)
```

6.2.1.9.36 ChGetGridLabelFont Macro

File

Chart.h

C

```
#define ChGetGridLabelFont(pCh) (((CHART *)pCh)->prm.pGridLabelsFont)
```

Module

Chart (

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.

Side Effects

none

Returns

Returns the address of the current font used for the title text.

Preconditions

none

Overview

This macro returns the location of the font used for the X and Y axis grid labels of the chart.

Syntax

```
ChGetGridLabelFont(pCh)
```

6.2.1.9.37 ChSetGridLabelFont Macro**File**

Chart.h

C

```
#define ChSetGridLabelFont(pCh, pNewFont) (((CHART *)pCh)->prm.pGridLabelsFont = pNewFont)
```

Module

Chart ([🔗](#))

Input Parameters

Input Parameters	Description
pCh	Pointer to the object.
pNewFont	Pointer to the font used.

Side Effects

none

Returns

none.

Preconditions

none

Example

See ChCreate ([🔗](#)()) example.

Overview

This macro sets the location of the font used for the X and Y axis grid labels of the chart.

Syntax

```
ChSetGridLabelFont(pCh, pNewFont)
```

6.2.1.9.38 ChFreeDataSeries Function**File**

Chart.h

C

```
void ChFreeDataSeries(
    void * pObj
);
```

Module

Chart ([🔗](#))

Input Parameters

Input Parameters	Description
pCh	Pointer to the chart object.

Side Effects

none.

Returns

none.

Preconditions

none

Example

```
void ClearChartData(CHART *pCh) {
    if(pCh->pChData != NULL)
        // remove the all data series
        ChFreeDataSeries(pCh);
}
```

Overview

This function removes DATASERIES (■) object from the list of DATASERIES (■) objects and frees the memory used of that removed object.

Syntax

```
void ChFreeDataSeries(void *pObj)
```

6.2.1.9.39 ChTranslateMsg Function

File

Chart.h

C

```
WORD ChTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Chart (■)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pCh	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- CH_MSG_SELECTED – Chart (■) area is selected
- OBJ_MSG_INVALID – Chart (■) is not affected

none.

Preconditions

none

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
CH_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the area of the chart.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
ChTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.9.40 CHART Structure**File**

Chart.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    CHARTPARAM prm;
    DATASERIES * pChData;
} CHART;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
CHARTPARAM prm;	Structure for the parameters of the chart.
DATASERIES * pChData;	Pointer to the first chart data series in the link list of data series.

Module

Chart (■)

Overview

Defines the parameters required for a chart Object.

6.2.1.9.41 DATASERIES Structure**File**

Chart.h

C

```
typedef struct {
    XCHAR * pSData;
    WORD samples;
    BYTE show;
    WORD * pData;
    void * pNextData;
} DATASERIES;
```

Members

Members	Description
XCHAR * pSData;	Pointer to the data series name.
WORD samples;	Indicates the number of data samples (or data points) contained in the array specified by pData.
BYTE show;	The flag to indicate if the data series will be shown or not. If this flag is set to SHOW_DATA (█), the data series will be shown. If HIDE_DATA (█), the data series will not be shown.
WORD * pData;	Pointer to the array of data samples.
void * pNextData;	Pointer to the next data series. NULL if no other data series follows.

Module

Chart (█)

Overview

Defines a variable for the CHART (█) object. It specifies the number of samples, pointer to the array of samples for the data series and pointer to the next data series. A member of this structure (show) is used as a flag to determine if the series is to be drawn or not. Together with the smplStart and smplEnd it will determine what kind of chart will be drawn.

6.2.1.9.42 CHARTPARAM Structure

File

Chart.h

C

```
typedef struct {
    XCHAR * pTitle;
    XCHAR * pSmplLabel;
    XCHAR * pValLabel;
    SHORT seriesCount;
    WORD smplStart;
    WORD smplEnd;
    WORD valMax;
    WORD valMin;
    WORD perMax;
    WORD perMin;
    GFX_COLOR * pColor;
    void * pTitleFont;
    void * pAxisLabelsFont;
    void * pGridLabelsFont;
} CHARTPARAM;
```

Members

Members	Description
XCHAR * pTitle;	Pointer to the Title of the chart.
XCHAR * pSmplLabel;	Pointer to the bar chart sample axis label. Depending
XCHAR * pValLabel;	Pointer to the bar chart value axis label. Depending
SHORT seriesCount;	Number of data series that will be displayed when chart is drawn.
WORD smplStart;	Start point of data sample range to be displayed (minimum/default value = 1)
WORD smplEnd;	End point of data sample range to be displayed.
WORD valMax;	Maximum value of a sample that can be displayed.
WORD valMin;	Minimum value of a sample that can be displayed.
WORD perMax;	Maximum value of the percentage range that can be displayed.
WORD perMin;	Minimum value of the percentage range that can be displayed.
GFX_COLOR * pColor;	Pointer to the color table used to draw the chart data.
void * pTitleFont;	Pointer to the font used for the title label of the chart.

<code>void * pAxisLabelsFont;</code>	Pointer to the font used for X and Y axis labels.
<code>void * pGridLabelsFont;</code>	Pointer to the font used for X and Y axis grid labels.

Module

Chart (■)

Overview

Defines the parameters for the CHART (■) object.

6.2.1.9.43 Color Table**Macros**

Name	Description
<code>CH_CLR0 (■)</code>	Bright Blue
<code>CH_CLR1 (■)</code>	Bright Red
<code>CH_CLR2 (■)</code>	Bright Green
<code>CH_CLR3 (■)</code>	Bright Yellow
<code>CH_CLR4 (■)</code>	Orange
<code>CH_CLR5 (■)</code>	Blue
<code>CH_CLR6 (■)</code>	Red
<code>CH_CLR7 (■)</code>	Green
<code>CH_CLR8 (■)</code>	Yellow
<code>CH_CLR9 (■)</code>	Dark Orange
<code>CH_CLR10 (■)</code>	Light Blur
<code>CH_CLR11 (■)</code>	Light Red
<code>CH_CLR12 (■)</code>	Light Green
<code>CH_CLR13 (■)</code>	Light Yellow
<code>CH_CLR14 (■)</code>	Light Orange
<code>CH_CLR15 (■)</code>	Gold

Module

Chart (■)

Description

Default color table used to draw data points in a chart.

6.2.1.9.43.1 CH_CLR0 Macro**File**

Chart.h

C`#define CH_CLR0 WHITE`**Description**

Bright Blue

6.2.1.9.43.2 CH_CLR1 Macro**File**

Chart.h

C

```
#define CH_CLR1 BLACK
```

Description

Bright Red

6.2.1.9.43.3 CH_CLR2 Macro

File

Chart.h

C

```
#define CH_CLR2 WHITE
```

Description

Bright Green

6.2.1.9.43.4 CH_CLR3 Macro

File

Chart.h

C

```
#define CH_CLR3 BLACK
```

Description

Bright Yellow

6.2.1.9.43.5 CH_CLR4 Macro

File

Chart.h

C

```
#define CH_CLR4 WHITE
```

Description

Orange

6.2.1.9.43.6 CH_CLR5 Macro

File

Chart.h

C

```
#define CH_CLR5 BLACK
```

Description

Blue

6.2.1.9.43.7 CH_CLR6 Macro

File

Chart.h

C

```
#define CH_CLR6 WHITE
```

Description

Red

6.2.1.9.43.8 CH_CLR7 Macro

File

Chart.h

C

```
#define CH_CLR7 BLACK
```

Description

Green

6.2.1.9.43.9 CH_CLR8 Macro

File

Chart.h

C

```
#define CH_CLR8 WHITE
```

Description

Yellow

6.2.1.9.43.10 CH_CLR9 Macro

File

Chart.h

C

```
#define CH_CLR9 BLACK
```

Description

Dark Orange

6.2.1.9.43.11 CH_CLR10 Macro

File

Chart.h

C

```
#define CH_CLR10 WHITE
```

Description

Light Blur

6.2.1.9.43.12 CH_CLR11 Macro

File

Chart.h

C

```
#define CH_CLR11 BLACK
```

Description

Light Red

6.2.1.9.43.13 CH_CLR12 Macro

File

Chart.h

C

```
#define CH_CLR12 WHITE
```

Description

Light Green

6.2.1.9.43.14 CH_CLR13 Macro

File

Chart.h

C

```
#define CH_CLR13 BLACK
```

Description

Light Yellow

6.2.1.9.43.15 CH_CLR14 Macro

File

Chart.h

C

```
#define CH_CLR14 WHITE
```

Description

Light Orange

6.2.1.9.43.16 CH_CLR15 Macro

File

Chart.h

C

```
#define CH_CLR15 BLACK
```

Description

Gold

6.2.1.10 Checkbox

Check Box is an Object that simulates a check box on paper. Usually it is used as an option setting where the checked or filled state means the option is enabled and the unfilled or unchecked state means the option is disabled.

Functions

Name	Description
CbCreate (■)	This function creates a CHECKBOX (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
CbDraw (■)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
CbSetText (■)	This function sets the text that will be used.
CbMsgDefault (■)	This function performs the actual state change based on the translated message given. The following state changes are supported:
CbTranslateMsg (■)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
CbGetText (■)	This macro returns the location of the text used for the check box.

Structures

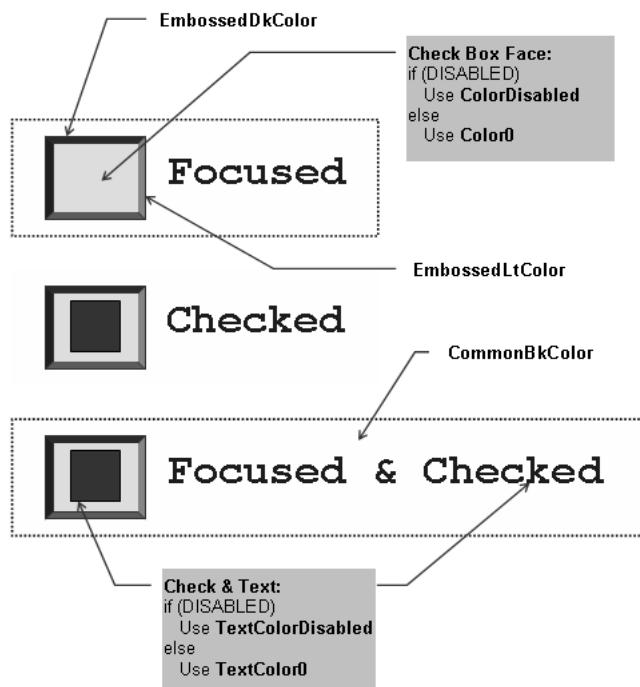
Name	Description
CHECKBOX (■)	The structure contains check box data

Description

Check Box supports Keyboard and Touchscreen inputs, replying to their events with the following messages:

1. CB_MSG_UNCHECKED - When the check box is unchecked.
2. CB_MSG_CHECKED - When check box is checked.

The Check Box Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



6.2.1.10.1 Check Box States

Macros

Name	Description
CB_CHECKED (█)	Checked state
CB_DISABLED (█)	Disabled state
CB_DRAW (█)	Whole check box must be redrawn
CB_DRAW_CHECK (█)	Check box mark should be redrawn
CB_DRAW_FOCUS (█)	Focus must be redrawn
CB_FOCUSED (█)	Focus state
CB_HIDE (█)	Check box must be removed from screen

Module

Checkbox (█)

Description

List of Checkbox (█) bit states.

6.2.1.10.1.1 CB_CHECKED Macro

File

CheckBox.h

C

```
#define CB_CHECKED 0x0004 // Checked state
```

Description

Checked state

6.2.1.10.1.2 CB_DISABLED Macro

File

CheckBox.h

C

```
#define CB_DISABLED 0x0002 // Disabled state
```

Description

Disabled state

6.2.1.10.1.3 CB_DRAW Macro

File

CheckBox.h

C

```
#define CB_DRAW 0x4000 // Whole check box must be redrawn
```

Description

Whole check box must be redrawn

6.2.1.10.1.4 CB_DRAW_CHECK Macro

File

CheckBox.h

C

```
#define CB_DRAW_CHECK 0x1000 // Check box mark should be redrawn
```

Description

Check box mark should be redrawn

6.2.1.10.1.5 CB_DRAW_FOCUS Macro

File

CheckBox.h

C

```
#define CB_DRAW_FOCUS 0x2000 // Focus must be redrawn
```

Description

Focus must be redrawn

6.2.1.10.1.6 CB_FOCUSED Macro

File

CheckBox.h

C

```
#define CB_FOCUSED 0x0001 // Focus state
```

Description

Focus state

6.2.1.10.1.7 CB_HIDE Macro

File

CheckBox.h

C

```
#define CB_HIDE 0x8000 // Check box must be removed from screen
```

Description

Check box must be removed from screen

6.2.1.10.2 CbCreate Function

File

CheckBox.h

C

```
CHECKBOX * CbCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    GOL_SCHEME * pScheme
);
```

Module

Checkbox (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object
SHORT bottom	Bottom most position of the object
WORD state	Sets the initial state of the object
XCHAR * pText	Pointer to the text of the check box.
GOL_SCHEME * pScheme	Pointer to the style scheme

Side Effects

none

Returns

Returns the pointer to the object created

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
CHECKBOX *pCb[ 2 ];

pScheme = GOLCreateScheme();
pCb = CbCreate(ID_CHECKBOX1,           // ID
                20,135,150,175,      // dimension
```

```

        CB_DRAW,           // Draw the object
        "Scale",          // text
        pScheme);         // use this scheme

pCb = CbCreate(ID_CHECKBOX2,
                170,135,300,175,
                CB_DRAW,
                "Animate",
                pScheme);         // ID
                           // dimension
                           // Draw the object
                           // text
                           // use this scheme

while( !CbDraw(pCb[0]));           // draw the objects
while( !CbDraw(pCb[1]));

```

Overview

This function creates a CHECKBOX (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
CHECKBOX (█) *CbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR (█)
*pText, GOL_SCHEME (█) *pScheme)
```

6.2.1.10.3 CbDraw Function

File

CheckBox.h

C

```
WORD CbDraw(
    void * pObj
);
```

Module

Checkbox (█)

Input Parameters

Input Parameters	Description
pCb	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See CbCreate (█)() Example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object

is started. This is to avoid incomplete object rendering.

Syntax

```
WORD CbDraw(void *pObj)
```

6.2.1.10.4 CbGetText Macro

File

CheckBox.h

C

```
#define CbGetText(pCb) pCb->pText
```

Module

Checkbox (█)

Input Parameters

Input Parameters	Description
pCb	Pointer to the object

Side Effects

none

Returns

Returns the location of the text used.

Preconditions

none

Overview

This macro returns the location of the text used for the check box.

Syntax

```
CbGetText(pCb)
```

6.2.1.10.5 CbSetText Function

File

CheckBox.h

C

```
void CbSetText(
    CHECKBOX * pCb,
    XCHAR * pText
);
```

Module

Checkbox (█)

Input Parameters

Input Parameters	Description
CHECKBOX * pCb	The pointer to the check box whose text will be modified.
XCHAR * pText	The pointer to the text that will be used.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the text that will be used.

Syntax

```
CbSetText(CHECKBOX (■) *pCb, XCHAR (■) *pText)
```

6.2.1.10.6 CbMsgDefault Function

File

CheckBox.h

C

```
void CbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Checkbox (■)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL message
pCb	The pointer to the object whose state will be modified

Side Effects

none

Returns

none

Preconditions

none

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
CB_MSG_CHECKED	Touch Screen, Keyboard	Set CB_CHECKED (■)	Check Box will be redrawn in checked state.
CB_MSG_UNCHECKED	Touch Screen, Keyboard	Clear CB_CHECKED (■)	Check Box will be redrawn in un-checked state.

Syntax

```
CbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (■)* pMsg)
```

6.2.1.10.7 CbTranslateMsg Function

File

CheckBox.h

C

```
WORD CbTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Checkbox (■)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	pointer to the message struct containing the message the user
pCb	the pointer to the object where the message will be evaluated to check if the message will affect the object

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- CB_MSG_CHECKED – Check Box is checked.
- CB_MSG_UNCHECKED – Check Box is unchecked.
- OBJ_MSG_INVALID – Check Box is not affected.

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (■)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
CB_MSG_CHECKED	Touch Screen	EVENT_PRESS	If events occurs and the x,y position falls in the area of the check box while the check box is unchecked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED (■) or SCAN_SPACE_PRESSED (■) while the check box is unchecked.
CB_MSG_UNCHECKED	Touch Screen	EVENT_PRESS	If events occurs and the x,y position falls in the area of the check box while the check box is checked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED (■) or SCAN_SPACE_PRESSED (■) while the check box is checked.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD CbTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.10.8 CHECKBOX Structure**File**

CheckBox.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT textHeight;
    XCHAR * pText;
} CHECKBOX;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
SHORT textHeight;	Pre-computed text height
XCHAR * pText;	Pointer to text

Module

Checkbox (■)

Overview

The structure contains check box data

6.2.1.11 Round Dial

Dial is an Object that can be used to display emulate a turn dial that can both go in clockwise or counterclockwise.

Functions

	Name	Description
✿	RdiaCreate (■)	This function creates a ROUNDDIAL (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
✿	RdiaDraw (■)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the center (x,y) position and the radius parameters. The colors used are dependent on the state of the object. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
✿	RdiaMsgDefault (■)	This function performs the actual state change based on the translated message given. The following state changes are supported:
✿	RdiaTranslateMsg (■)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Macros

Name	Description
RdialIncVal (■)	Used to directly increment the value. The delta change used is the resolution setting (res).
RdiaDecVal (■)	Used to directly decrement the value. The delta change used is the resolution setting (res).
RdiaGetVal (■)	Returns the current dial value. Value is always in the 0-max range inclusive.

RdiaSetVal (█)	Sets the value to the given new value. Value set must be in 0-max range inclusive.
----------------	--

Structures

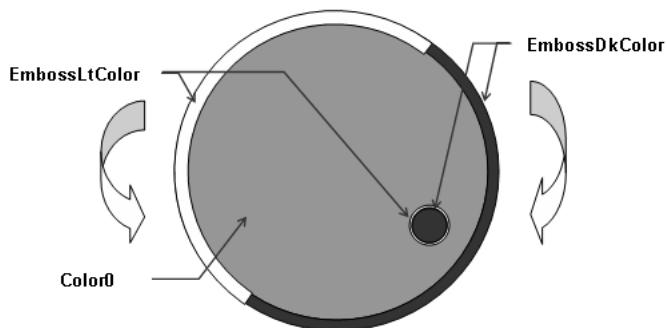
Name	Description
ROUNDDIAL (█)	Defines the parameters required for a dial Object. The curr_xPos, curr_yPos, new_xPos and new_yPos parameters are internally generated to aid in the redrawing of the dial. User must avoid modifying these values.

Description

Dial supports only Touchscreen inputs, replying to their events with the following messages:

1. RD_MSG_CLOCKWISE - When movement of the touch is in the face of the dial and in the clockwise direction.
2. RD_MSG_CTR_CLOCKWISE - When movement of the touch is in the face of the dial and in the counter clockwise direction.

The Dial object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



CommonBkColor – used to hide the slider from the screen.

6.2.1.11.1 Dial States

Macros

Name	Description
RDIA_DISABLED (█)	Bit for disabled state.
RDIA_DRAW (█)	Bit to indicate object must be redrawn.
RDIA_HIDE (█)	Bit to indicate object must be removed from screen.
RDIA_ROT_CCW (█)	Bit for rotate counter clockwise state.
RDIA_ROT_CW (█)	Bit for rotate clockwise state.

Module

Round Dial (█)

Description

List of Dial (█) bit states.

6.2.1.11.1.1 RDIA_DISABLED Macro

File

RoundDial.h

C

```
#define RDIA_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.11.1.2 RDIA_DRAW Macro

File

RoundDial.h

C

```
#define RDIA_DRAW 0x4000 // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

6.2.1.11.1.3 RDIA_HIDE Macro

File

RoundDial.h

C

```
#define RDIA_HIDE 0x8000 // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

6.2.1.11.1.4 RDIA_ROT_CCW Macro

File

RoundDial.h

C

```
#define RDIA_ROT_CCW 0x0008 // Bit for rotate counter clockwise state.
```

Description

Bit for rotate counter clockwise state.

6.2.1.11.1.5 RDIA_ROT_CW Macro

File

RoundDial.h

C

```
#define RDIA_ROT_CW 0x0004 // Bit for rotate clockwise state.
```

Description

Bit for rotate clockwise state.

6.2.1.11.2 RdiaCreate Function

File

RoundDial.h

C

```
ROUNDDIAL * RdiaCreate(
    WORD ID,
    SHORT x,
    SHORT y,
    SHORT radius,
    WORD state,
    SHORT res,
    SHORT value,
    SHORT max,
    GOL_SCHEME * pScheme
);
```

Module

Round Dial (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT x	Location of the center of the dial in the x coordinate.
SHORT y	Location of the center of the dial in the y coordinate.
SHORT radius	Defines the radius of the dial.
WORD state	Sets the initial state of the object.
SHORT res	Sets the resolution of the dial when rotating clockwise or counter clockwise.
SHORT value	Sets the initial value of the dial.
SHORT max	Sets the maximum value of the dial.
GOL_SCHEME * pScheme	Pointer to the style scheme used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
ROUNDDIAL *pDial;
WORD state;

pScheme = GOLCreateScheme();
state = RDIA_DRAW;

// creates a dial at (50,50) x,y location, with an initial value
// of 50, a resolution of 2 and maximum value of 100.
pDial = RdiaCreate(1,50,50,25,118,0, state, 2, 50, 100, pScheme);
// check if dial was created
if (pDial == NULL)
    return 0;

return 1;
```

Overview

This function creates a ROUNDDIAL (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
ROUNDDIAL (█) *RdiaCreate( WORD ID, SHORT x, SHORT y, SHORT radius, WORD state, SHORT res, SHORT value,
                           SHORT max, GOL_SCHEME (█) *pScheme);
```

6.2.1.11.3 RdiaDraw Function

File

RoundDial.h

C

```
WORD RdiaDraw(
    void * pObj
);
```

Module

Round Dial (█)

Input Parameters

Input Parameters	Description
pDia	Pointer to the object

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the center (x,y) position and the radius parameters. The colors used are dependent on the state of the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD RdiaDraw(void *pObj)
```

6.2.1.11.4 RdiaIncVal Macro

File

RoundDial.h

C

```
#define RdiaIncVal(pDia) RdiaSetVal(pDia, (pDia->val + pDia->res))
```

Module

Round Dial (█)

Input Parameters

Input Parameters	Description
pDia	Pointer to the object.

Side Effects

none

Returns

none

Preconditions

none

Example

```

WORD updatedVal, prevVal;
ROUNDDIAL *pDia;

// assuming pDia is initialized to an existing dial Object
// assume GetInput() is a function that retrieves source data
prevVal = RdiaGetVal(pDia);
updatedVal = GetInput();
if (updatedVal > prevVal)
    RdiaIncVal(pDia);
if (updatedVal < prevVal)
    RdiaDecVal(pDia);

```

Overview

Used to directly increment the value. The delta change used is the resolution setting (res).

Syntax

```
RdiaIncVal(pDia)
```

6.2.1.11.5 RdiaDecVal Macro**File**

```
RoundDial.h
```

C

```
#define RdiaDecVal(pDia) RdiaSetVal(pDia, (pDia->pos - pDia->res))
```

Module

```
Round Dial (█)
```

Input Parameters

Input Parameters	Description
pDia	Pointer to the object.

Side Effects

none

Returns

none

Preconditions

none

Example

Refer to RdialIncVal (█)() example.

Overview

Used to directly decrement the value. The delta change used is the resolution setting (res).

Syntax

```
RdiaDecVal(pDia)
```

6.2.1.11.6 RdiaGetVal Macro**File**

RoundDial.h

C

```
#define RdiaGetVal(pDia) (pDia)->value
```

Module

Round Dial ([🔗](#))

Input Parameters

Input Parameters	Description
pDia	Pointer to the object.

Side Effects

none

Returns

Returns the current value of the dial.

Preconditions

none

Example

```
WORD currVal;
ROUNDDIAL *pDia;

// assuming pDia is initialized to an existing dial Object
currVal = RdiaGetVal(pDia);
```

Overview

Returns the current dial value. Value is always in the 0-max range inclusive.

Syntax

```
RdiaGetVal(pDia)
```

6.2.1.11.7 RdiaSetVal Macro**File**

RoundDial.h

C

```
#define RdiaSetVal(pDia, newVal) (pDia)->value = newVal
```

Module

Round Dial ([🔗](#))

Input Parameters

Input Parameters	Description
pDia	Pointer to the object.
newVal	New dial value.

Side Effects

none

Returns

none

Preconditions

none

Example

```
WORD updatedVal;
ROUNDDIAL *pDia;

// assuming pDia is initialized to an existing dial Object
// assume GetInput() is a function that retrieves source data
updatedVal = GetInput();
RdiaSetVal(pDia, updatedVal);
```

Overview

Sets the value to the given new value. Value set must be in 0-max range inclusive.

Syntax

```
RdiaSetVal(pDia, newVal)
```

6.2.1.11.8 RdiaMsgDefault Function

File

```
RoundDial.h
```

C

```
void RdiaMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

```
Round Dial (■)
```

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL message
pDia	The pointer to the object whose state will be modified

Side Effects

none

Returns

none

Preconditions

none

Example

See RdiaTranslateMsg (■)() example.

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
RD_MSG_CLOCKWISE	Touch Screen	Set RDIA_ROT_CW (■), Set RDIA_DRAW (■)	Dial (■) will be redrawn with clockwise update.
RD_MSG_CTR_CLOCKWISE	Touch Screen	Set RDIA_ROT_CCW (■), Set RDIA_DRAW (■)	Dial (■) will be redrawn with counter clockwise update.

Syntax

```
RdiaMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (■)* pMsg)
```

6.2.1.11.9 RdiaTranslateMsg Function

File

RoundDial.h

C

```
WORD RdiaTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Round Dial (■)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pDia	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- RD_MSG_CLOCKWISE – Dial (■) is moved in a clockwise direction.
- RD_MSG_CTR_CLOCKWISE – Dial (■) is moved in a counter clockwise direction.
- OBJ_MSG_INVALID – Dial (■) is not affected

Preconditions

none

Example

```
void MyGOLMsg (GOL_MSG *pMsg) {
    OBJ_HEADER *pCurrentObj;
    WORD objMsg;

    if (pMsg->event == EVENT_INVALID)
```

```

    return;
pCurrentObj = GOLGetList();

while(pCurrentObj != NULL){
    // Process only ROUNDDIAL
    if(!IsObjUpdated(pCurrentObj))){
        switch(pCurrentObj->type){
            case OBJ_ROUNDDIAL:
                objMsg = RdiaTranslateMsg((ROUNDDIAL*)pCurrentObj, pMsg);
                if(objMsg == OBJ_MSG_INVALID)
                    break;
                if(GOLMsgCallback(objMsg,pCurrentObj,pMsg))
                    RdiaMsgDefault(objMsg,(ROUNDDIAL*)pCurrentObj);
                break;
            default: break;
        }
    }
    pCurrentObj = pCurrentObj->pNxtObj;
}

```

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
RD_MSG_CLOCKWISE	Touch Screen	EVENT_MOVE	If events occurs and the x,y position falls in the face of the Dial (■) and moving in the clockwise rotation.
RD_MSG_CTR_CLOCKWISE	Touch Screen	EVENT_MOVE	If events occurs and the x,y position falls in the face of the Dial (■) and moving in the counter clockwise rotation.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

RdiaTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)

6.2.1.11.10 ROUNDDIAL Structure

File

RoundDial.h

C

```

typedef struct {
    OBJ_HEADER hdr;
    SHORT xCenter;
    SHORT yCenter;
    SHORT radius;
    SHORT value;
    WORD max;
    WORD res;
    SHORT curr_xPos;
    SHORT curr_yPos;
    SHORT new_xPos;
    SHORT new_yPos;
    SHORT vAngle;
} ROUNDDIAL;

```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).

SHORT xCenter;	x coordinate center position.
SHORT yCenter;	y coordinate center position.
SHORT radius;	Radius of the dial.
SHORT value;	Initial value of the dial.
WORD max;	Maximum value of variable value (maximum = 65535).
WORD res;	Resolution of movement.
SHORT curr_xPos;	Current x position.
SHORT curr_yPos;	Current y position.
SHORT new_xPos;	New x position.
SHORT new_yPos;	New y position.

Module

Round Dial ([🔗](#))

Overview

Defines the parameters required for a dial Object. The curr_xPos, curr_yPos, new_xPos and new_yPos parameters are internally generated to aid in the redrawing of the dial. User must avoid modifying these values.

6.2.1.11.11 Files**6.2.1.12 Digital Meter**

DigitalMeter is an Object that can be used to display a value of a sampled variable. This Object is ideal when fast refresh of the value is needed. The Object refreshes only the digits that needs to change. A limitation of this Object is that the font used should have equal character widths.

Functions

	Name	Description
💡	DmCreate (🔗)	This function creates a DIGITALMETER (🔗) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
💡	DmDraw (🔗)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
💡	DmSetValue (🔗)	This function sets the value that will be used for the object.
💡	DmTranslateMsg (🔗)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
DmGetValue (🔗)	This macro returns the current value used for the object.
DmDecVal (🔗)	This macro is used to directly decrement the value.
DmIncVal (🔗)	This macro is used to directly increment the value.

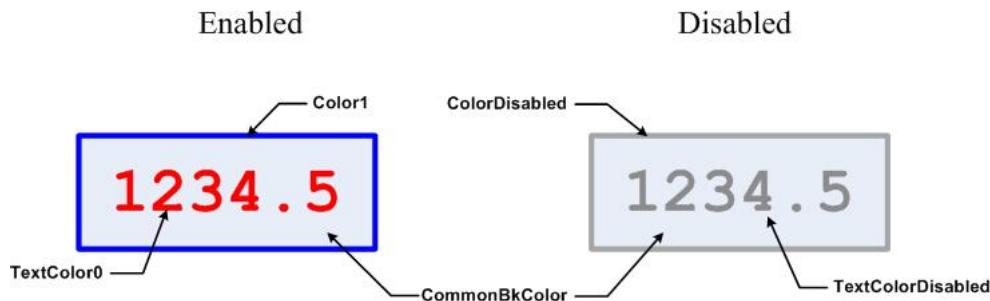
Structures

Name	Description
DIGITALMETER (🔗)	Defines the parameters required for a Digital Meter Object.

Description

DigitalMeter supports only Touchscreen inputs, replying to touch screen events with the message:
DM_MSG_SELECTED.

The DigitalMeter object is rendered using the assigned style scheme. The following figure illustrates the color assignments for the digital meter.



6.2.1.12.1 Digital Meter States

Macros

Name	Description
DM_DISABLED (■)	Bit for disabled state.
DM_DRAW (■)	Bit to indicate object must be redrawn.
DM_HIDE (■)	Bit to remove object from screen.
DM_CENTER_ALIGN (■)	Bit to indicate value is center aligned.
DM_RIGHT_ALIGN (■)	Bit to indicate value is left aligned.
DM_FRAME (■)	Bit to indicate frame is displayed.
DM_UPDATE (■)	Bit to indicate that only text must be redrawn.

Module

Digital Meter (■)

Description

List of Digital Meter (■) bit states.

6.2.1.12.1.1 DM_DISABLED Macro

File

DigitalMeter.h

C

```
#define DM_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.12.1.2 DM_DRAW Macro

File

DigitalMeter.h

C

```
#define DM_DRAW 0x4000 // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

6.2.1.12.1.3 DM_HIDE Macro

File

DigitalMeter.h

C

```
#define DM_HIDE 0x8000 // Bit to remove object from screen.
```

Description

Bit to remove object from screen.

6.2.1.12.1.4 DM_CENTER_ALIGN Macro

File

DigitalMeter.h

C

```
#define DM_CENTER_ALIGN 0x0008 // Bit to indicate value is center aligned.
```

Description

Bit to indicate value is center aligned.

6.2.1.12.1.5 DM_RIGHT_ALIGN Macro

File

DigitalMeter.h

C

```
#define DM_RIGHT_ALIGN 0x0004 // Bit to indicate value is left aligned.
```

Description

Bit to indicate value is left aligned.

6.2.1.12.1.6 DM_FRAME Macro

File

DigitalMeter.h

C

```
#define DM_FRAME 0x0010 // Bit to indicate frame is displayed.
```

Description

Bit to indicate frame is displayed.

6.2.1.12.1.7 DM_UPDATE Macro

File

DigitalMeter.h

C

```
#define DM_UPDATE 0x2000 // Bit to indicate that only text must be redrawn.
```

Description

Bit to indicate that only text must be redrawn.

6.2.1.12.2 DmCreate Function**File**

DigitalMeter.h

C

```
DIGITALMETER * DmCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    DWORD Value,
    BYTE NoOfDigits,
    BYTE DotPos,
    GOL_SCHEME * pScheme
);
```

Module

Digital Meter (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
DWORD Value	Sets the initial value to be displayed
BYTE NoOfDigits	Sets the number of digits to be displayed
BYTE DotPos	Sets the position of decimal point in the display
GOL_SCHEME * pScheme	Pointer to the style scheme. Set to NULL if default style scheme is used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
DIGITALMETER *pDm;

pScheme = GOLCreateScheme();
state = DM_DRAW | DM_FRAME | DM_CENTER_ALIGN;
DmCreate(ID_DIGITALMETER1, // ID
         30,80,235,160, // dimension
```

```

state,           // has frame and center aligned
789,4,1,        // to display 078.9
pScheme);       // use given scheme

while(!DmDraw(pDm));      // draw the object

```

Overview

This function creates a DIGITALMETER (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
DIGITALMETER (■) *DmCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state , DWORD
Value, BYTE NoOfDigits, BYTE DotPos, GOL_SCHEME (■) *pScheme)
```

6.2.1.12.3 DmDraw Function

File

DigitalMeter.h

C

```
WORD DmDraw(
    void * pObj
);
```

Module

Digital Meter (■)

Input Parameters

Input Parameters	Description
pDm	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See DmCreate (■)() Example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD DmDraw(void *pObj)
```

6.2.1.12.4 DmGetValue Macro

File

DigitalMeter.h

C

```
#define DmGetValue(pDm) pDm->Cvalue
```

Module

Digital Meter ([🔗](#))

Input Parameters

Input Parameters	Description
pDm	Pointer to the object.

Side Effects

none

Returns

Returns the value used.

Preconditions

none

Overview

This macro returns the current value used for the object.

Syntax

```
DmGetValue(pDm)
```

6.2.1.12.5 DmSetValue Function

File

DigitalMeter.h

C

```
void DmSetValue(
    DIGITALMETER * pDm,
    DWORD Value
);
```

Module

Digital Meter ([🔗](#))

Input Parameters

Input Parameters	Description
DIGITALMETER * pDm	The pointer to the object whose value will be modified.
DWORD Value	New value to be set for the Digital Meter (🔗).

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the value that will be used for the object.

Syntax

```
DmSetValue(DIGITALMETER (■) *pDm, DWORD Value)
```

6.2.1.12.6 DmDecVal Macro

File

DigitalMeter.h

C

```
#define DmDecVal(pDm, deltaValue) DmSetValue(pDm, (pDm->Cvalue - deltaValue))
```

Module

Digital Meter (■)

Input Parameters

Input Parameters	Description
pDm	Pointer to the object.
deltaValue	Number to be subtracted to the current Digital Meter (■) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly decrement the value.

Syntax

```
DmDecVal(pDm, deltaValue)
```

6.2.1.12.7 DmIncVal Macro

File

DigitalMeter.h

C

```
#define DmIncVal(pDm, deltaValue) DmSetValue(pDm, (pDm->Cvalue + deltaValue))
```

Module

Digital Meter (■)

Input Parameters

Input Parameters	Description
pDm	Pointer to the object.
deltaValue	Number to be added to the current Digital Meter (■) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly increment the value.

Syntax

```
DmIncVal(pDm, deltaValue)
```

6.2.1.12.8 DmTranslateMsg Function

File

DigitalMeter.h

C

```
WORD DmTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Digital Meter (

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pDm	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- DM_MSG_SELECTED – Digital Meter () is selected
- OBJ_MSG_INVALID – Digital Meter () is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg () example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
DM_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the Digital Meter (█).
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD DmTranslateMsg(void *pObj, GOL_MSG (█) *pMsg)
```

6.2.1.12.9 DIGITALMETER Structure**File**

DigitalMeter.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT textHeight;
    DWORD Cvalue;
    DWORD Pvalue;
    BYTE NoOfDigits;
    BYTE DotPos;
} DIGITALMETER;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (█)).
SHORT textHeight;	Pre-computed text height
DWORD Cvalue;	Current value
DWORD Pvalue;	Previous value
BYTE NoOfDigits;	Number of digits to be displayed
BYTE DotPos;	Position of decimal point

Module

Digital Meter (█)

Description

Structure: DIGITALMETER

Overview

Defines the parameters required for a Digital Meter (█) Object.

6.2.1.13 Edit Box

Edit Box is an Object that emulates a cell or a text area that can be edited dynamically.

Functions

	Name	Description
💡	EbCreate (█)	This function creates a EDITBOX (█) object with the parameters given and initializes the default settings. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

	EbDraw (Link)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
	EbSetText (Link)	This function sets the text to be used for the object.
	EbAddChar (Link)	This function inserts a character at the end of the text used by the object.
	EbDeleteChar (Link)	This function removes a character at the end of the text used by the object.
	EbMsgDefault (Link)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	EbTranslateMsg (Link)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
EbGetText (Link)	This macro returns the address of the current text string used for the object.

Structures

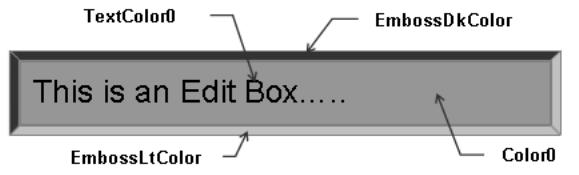
Name	Description
EDITBOX (Link)	Defines the parameters required for a Edit Box Object.

Description

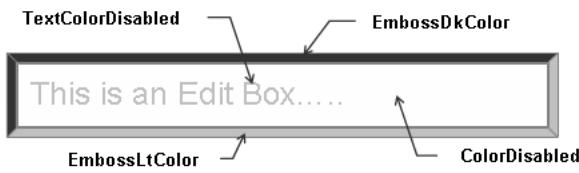
Edit Box supports only Keyboard inputs, replying to their events with the following messages:

1. EB_MSG_CHAR - when a character is to be inserted at the end of the current text.
2. EB_MSG_DEL - when a character is to be removed from the current text.

The Edit Box Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



Edit Box in enabled state



Edit Box in disabled state

6.2.1.13.1 Edit Box States

Macros

Name	Description
EB_CENTER_ALIGN (█)	Bit to indicate text is center aligned.
EB_DISABLED (█)	Bit for disabled state.
EB_DRAW (█)	Bit to indicate whole edit box must be redrawn.
EB_HIDE (█)	Bit to remove object from screen.
EB_FOCUSED (█)	Bit for focused state. Cursor caret will be drawn when EB_DRAW_CARET (█) is also set.
EB_RIGHT_ALIGN (█)	Bit to indicate text is left aligned.
EB_DRAW_CARET (█)	Bit to indicate the cursor caret will be drawn if EB_FOCUSED (█) state bit is set and erase when EB_FOCUSED (█) state bit is not set.
EB_CARET (█)	Bit to indicate the cursor caret will always be shown.

Module

Edit Box (█)

Description

List of Edit Box bit states.

6.2.1.13.1.1 EB_CENTER_ALIGN Macro

File

EditBox.h

C

```
#define EB_CENTER_ALIGN 0x0008 // Bit to indicate text is center aligned.
```

Description

Bit to indicate text is center aligned.

6.2.1.13.1.2 EB_DISABLED Macro

File

EditBox.h

C

```
#define EB_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.13.1.3 EB_DRAW Macro

File

EditBox.h

C

```
#define EB_DRAW 0x4000 // Bit to indicate whole edit box must be redrawn.
```

Description

Bit to indicate whole edit box must be redrawn.

6.2.1.13.1.4 EB_HIDE Macro

File

EditBox.h

C

```
#define EB_HIDE 0x8000 // Bit to remove object from screen.
```

Description

Bit to remove object from screen.

6.2.1.13.1.5 EB_FOCUSED Macro

File

EditBox.h

C

```
#define EB_FOCUSED 0x0001 // Bit for focused state. Cursor caret will be drawn when  
EB_DRAW_CARET is also set.
```

Description

Bit for focused state. Cursor caret will be drawn when EB_DRAW_CARET (☒) is also set.

6.2.1.13.1.6 EB_RIGHT_ALIGN Macro

File

EditBox.h

C

```
#define EB_RIGHT_ALIGN 0x0004 // Bit to indicate text is left aligned.
```

Description

Bit to indicate text is left aligned.

6.2.1.13.1.7 EB_DRAW_CARET Macro

File

EditBox.h

C

```
#define EB_DRAW_CARET 0x2000 // Bit to indicate the cursor caret will be drawn if  
EB_FOCUSED state bit is set and erase when EB_FOCUSED (☒) state bit is not set.
```

Description

Bit to indicate the cursor caret will be drawn if EB_FOCUSED (☒) state bit is set and erase when EB_FOCUSED (☒) state bit is not set.

6.2.1.13.1.8 EB_CARET Macro

File

EditBox.h

C

```
#define EB_CARET 0x0010 // Bit to indicate the cursor caret will always be shown.
```

Description

Bit to indicate the cursor caret will always be shown.

6.2.1.13.2 EbCreate Function

File

EditBox.h

C

```
EDITBOX * EbCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    WORD charMax,
    GOL_SCHEME * pScheme
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
XCHAR * pText	Pointer to the text to be used.
WORD charMax	Defines the maximum number of characters in the edit box.
GOL_SCHEME * pScheme	Pointer to the style scheme.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
#define ID_MYEDITBOX      101
EDITBOX *pEb;

pEb = EbCreate(ID_MYEDITBOX,          // ID
                10,                 // left
                10,                 // top
                100,                // right
                30,                 // bottom
                EB_DRAW,             // redraw after creation
                NULL,                // no text yet
                4,                  // display only four characters
                pScheme);            // pointer to the style scheme

if( pEb == NULL )
{
    // MEMORY ERROR. Object was not created.
}
```

Overview

This function creates a EDITBOX (█) object with the parameters given and initializes the default settings. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
EDITBOX (█) *EbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state , XCHAR (█)
*pText, WORD charMax, GOL_SCHEME (█) *pScheme)
```

6.2.1.13.3 EbDraw Function

File

EditBox.h

C

```
WORD EbDraw(
    void * pObj
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
pEb	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD EbDraw(void *pObj)
```

6.2.1.13.4 EbGetText Macro

File

EditBox.h

C

```
#define EbGetText(pEb) (pEb->pBuffer)
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
pEb	Pointer to the object

Side Effects

none

Returns

Returns pointer to the text string being used.

Preconditions

none

Overview

This macro returns the address of the current text string used for the object.

Syntax

EbGetText(pEb)

6.2.1.13.5 EbSetText Function

File

EditBox.h

C

```
void EbSetText(
    EDITBOX * pEb,
    XCHAR * pText
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
EDITBOX * pEb	The pointer to the object whose text will be modified.
XCHAR * pText	Pointer to the text that will be used.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the text to be used for the object.

Syntax

EbSetText(EDITBOX (█) *pEb, XCHAR (█) *pText)

6.2.1.13.6 EbAddChar Function

File

EditBox.h

C

```
void EbAddChar(
    EDITBOX * pEb,
    XCHAR ch
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
EDITBOX * pEb	The pointer to the object whose text will be modified.
XCHAR ch	Character to be inserted.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function inserts a character at the end of the text used by the object.

Syntax

```
void EbAddChar(EDITBOX (█)* pEb, XCHAR (█) ch)
```

6.2.1.13.7 EbDeleteChar Function

File

EditBox.h

C

```
void EbDeleteChar(
    EDITBOX * pEb
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
EDITBOX * pEb	The pointer to the object to be modified.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function removes a character at the end of the text used by the object.

Syntax

```
void EbDeleteChar(EDITBOX (*) pEb)
```

6.2.1.13.8 EbMsgDefault Function

File

EditBox.h

C

```
void EbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL message.
pEb	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Example

Usage is similar to BtnMsgDefault (█)() example.

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
EB_MSG_CHAR	Keyboard	Set EB_DRAW (█)	New character is added and Edit Box will be redrawn.
EB_MSG_DEL	Keyboard	Set EB_DRAW (█)	Last character is removed and Edit Box will be redrawn.

Syntax

```
void EbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (*) pMsg)
```

6.2.1.13.9 EbTranslateMsg Function

File

EditBox.h

C

```
WORD EbTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Edit Box (█)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pEb	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- EB_MSG_CHAR – New character should be added.
- EB_MSG_DEL – Last character should be removed.
- OBJ_MSG_INVALID – Object is not affected.

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (█)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
EB_MSG_CHAR	Keyboard	EVENT_CHARCODE	New character should be added.
EB_MSG_DEL	Keyboard	EVENT_KEYPRESS	Last character should be removed.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD EbTranslateMsg(void *pObj, GOL_MSG (█) *pMsg)
```

6.2.1.13.10 EDITBOX Structure

File

EditBox.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT textHeight;
    XCHAR * pBuffer;
    WORD charMax;
    WORD length;
} EDITBOX;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (█)).
SHORT textHeight;	Pre-computed text height.
XCHAR * pBuffer;	Pointer to text buffer.
WORD charMax;	Maximum number of characters in the edit box.
WORD length;	Current text length.

Module

Edit Box (█)

Overview

Defines the parameters required for a Edit Box Object.

6.2.1.14 Grid

Grid is an Object that draws a grid on the screen with each cell capable of displaying an image or a string.

Functions

	Name	Description
≡	GridCreate (█)	This function creates a GRID (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
≡	GridDraw (█)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
≡	GridClearCellState (█)	This function clears the state of the cell (or Grid Item) specified by the column and row.
≡	GridFreeItems (█)	This function removes all grid items for the given Grid and frees the memory used.
≡	GridGetCell (█)	This function removes all grid items for the given Grid and frees the memory used.
≡	GridSetCell (█)	This function sets the Grid Item state and data.
≡	GridSetCellState (█)	This function sets the state of the Grid Item or cell.
≡	GridSetFocus (█)	This function sets the focus of the specified Grid Item or cell.
≡	GridMsgDefault (█)	This function performs the actual state change based on the translated message given. The following state changes are supported:
≡	GridTranslateMsg (█)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
GridGetFocusX (§)	This macro returns the x position of the focused cell.
GridGetFocusY (§)	This macro returns the y position of the focused cell.
GRID_OUT_OF_BOUNDS (§)	Status of an out of bounds cell GridSetCell (§)() operation.
GRID_SUCCESS (§)	Status of a successful GridSetCell (§)() operation.

Structures

Name	Description
GRID (§)	Defines the parameters required for a grid Object. Clipping is not supported in grid object.
GRIDITEM (§)	Defines the grid item.

Description

Grid supports Keyboard and Touchscreen inputs, replying to their events with the following messages:

1. GRID_MSG_TOUCHED
2. GRID_MSG_ITEM_SELECTED
3. GRID_MSG_LEFT
4. GRID_MSG_RIGHT
5. GRID_MSG_UP
6. GRID_MSG_DOWN

See GridTranslateMsg (§)() and GridMsgDefault (§)() for details.

The Grid lines are drawn using the EmbossLitColor, the string drawn using the TextColor0 and the background is drawn using the CommonBkColor.

6.2.1.14.1 Grid States**Macros**

Name	Description
GRID_FOCUSED (§)	Bit for focused state
GRID_DISABLED (§)	Bit for disabled state
GRID_SHOW_LINES (§)	Display grid lines
GRID_SHOW_FOCUS (§)	Highlight the focused cell.
GRID_SHOW_BORDER_ONLY (§)	Draw only the outside border of the grid.
GRID_SHOW_SEPARATORS_ONLY (§)	Draw only the lines between cells (like Tic-tac-toe)
GRID_DRAW_ITEMS (§)	Bit to indicate that at least one item must be redrawn, but not the entire grid.
GRID_DRAW_ALL (§)	Bit to indicate whole edit box must be redrawn
GRID_HIDE (§)	Bit to remove object from screen

Module

Grid (§)

Description

List of Grid (§) bit states.

6.2.1.14.1.1 GRID_FOCUSED Macro

File

Grid.h

C

```
#define GRID_FOCUSED 0x0001 // Bit for focused state
```

Description

Bit for focused state

6.2.1.14.1.2 GRID_DISABLED Macro

File

Grid.h

C

```
#define GRID_DISABLED 0x0002 // Bit for disabled state
```

Description

Bit for disabled state

6.2.1.14.1.3 GRID_SHOW_LINES Macro

File

Grid.h

C

```
#define GRID_SHOW_LINES 0x0004 // Display grid lines
```

Description

Display grid lines

6.2.1.14.1.4 GRID_SHOW_FOCUS Macro

File

Grid.h

C

```
#define GRID_SHOW_FOCUS 0x0008 // Highlight the focused cell.
```

Description

Highlight the focused cell.

6.2.1.14.1.5 GRID_SHOW_BORDER_ONLY Macro

File

Grid.h

C

```
#define GRID_SHOW_BORDER_ONLY 0x0010 // Draw only the outside border of the grid.
```

Description

Draw only the outside border of the grid.

6.2.1.14.1.6 GRID_SHOW_SEPARATORS_ONLY Macro

File

Grid.h

C

```
#define GRID_SHOW_SEPARATORS_ONLY 0x0020 // Draw only the lines between cells (like Tic-tac-toe)
```

Description

Draw only the lines between cells (like Tic-tac-toe)

6.2.1.14.1.7 GRID_DRAW_ITEMS Macro

File

Grid.h

C

```
#define GRID_DRAW_ITEMS 0x1000 // Bit to indicate that at least one item must be redrawn, but not the entire grid.
```

Description

Bit to indicate that at least one item must be redrawn, but not the entire grid.

6.2.1.14.1.8 GRID_DRAW_ALL Macro

File

Grid.h

C

```
#define GRID_DRAW_ALL 0x4000 // Bit to indicate whole edit box must be redrawn
```

Description

Bit to indicate whole edit box must be redrawn

6.2.1.14.1.9 GRID_HIDE Macro

File

Grid.h

C

```
#define GRID_HIDE 0x8000 // Bit to remove object from screen
```

Description

Bit to remove object from screen

6.2.1.14.2 Grid Item States

Macros

Name	Description
GRIDITEM_SELECTED (▣)	The cell is selected.
GRIDITEM_IS_TEXT (▣)	The grid item is a text string.
GRIDITEM_IS_BITMAP (▣)	The grid item is a bitmap.
GRIDITEM_TEXTBOTTOM (▣)	Bit to indicate text is top aligned.
GRIDITEM_TEXTLEFT (▣)	Text in the cell is left aligned.

GRIDITEM_TEXtright (█)	Text in the cell is right aligned.
GRIDITEM_TEXTTOP (█)	Bit to indicate text is bottom aligned.
GRIDITEM_DRAW (█)	Draw this cell

Module

Grid (█)

Description

List of Grid (█) Items bit states.

6.2.1.14.2.1 GRIDITEM_SELECTED Macro**File**

Grid.h

C

```
#define GRIDITEM_SELECTED 0x0001 // The cell is selected.
```

Description

The cell is selected.

6.2.1.14.2.2 GRIDITEM_IS_TEXT Macro**File**

Grid.h

C

```
#define GRIDITEM_IS_TEXT 0x0000 // The grid item is a text string.
```

Description

The grid item is a text string.

6.2.1.14.2.3 GRIDITEM_IS_BITMAP Macro**File**

Grid.h

C

```
#define GRIDITEM_IS_BITMAP 0x0008 // The grid item is a bitmap.
```

Description

The grid item is a bitmap.

6.2.1.14.2.4 GRIDITEM_TEXTBOTTOM Macro**File**

Grid.h

C

```
#define GRIDITEM_TEXTBOTTOM 0x0040 // Bit to indicate text is top aligned.
```

Description

Bit to indicate text is top aligned.

6.2.1.14.2.5 GRIDITEM_TEXTLEFT Macro

File

Grid.h

C

```
#define GRIDITEM_TEXTLEFT 0x0020 // Text in the cell is left aligned.
```

Description

Text in the cell is left aligned.

6.2.1.14.2.6 GRIDITEM_TEXTRIGHT Macro

File

Grid.h

C

```
#define GRIDITEM_TEXTRIGHT 0x0010 // Text in the cell is right aligned.
```

Description

Text in the cell is right aligned.

6.2.1.14.2.7 GRIDITEM_TEXTTOP Macro

File

Grid.h

C

```
#define GRIDITEM_TEXTTOP 0x0080 // Bit to indicate text is bottom aligned.
```

Description

Bit to indicate text is bottom aligned.

6.2.1.14.2.8 GRIDITEM_DRAW Macro

File

Grid.h

C

```
#define GRIDITEM_DRAW 0x0100 // Draw this cell
```

Description

Draw this cell

6.2.1.14.3 GridCreate Function

File

Grid.h

C

```
GRID * GridCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
```

```

    SHORT numColumns,
    SHORT numRows,
    SHORT cellWidth,
    SHORT cellHeight,
    GOL_SCHEME * pScheme
);

```

Module

Grid (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
SHORT numColumns	Sets the number of columns for the grid.
SHORT numRows	Sets the number of rows for the grid.
SHORT cellWidth	Sets the width of each cell of the grid.
SHORT cellHeight	Sets the height of each cell of the grid.
GOL_SCHEME * pScheme	Pointer to the style scheme used for the object. Set to NULL if default style scheme is used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Overview

This function creates a GRID (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
GRID (█) *GridCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT
numColumns, SHORT numRows, SHORT cellWidth, SHORT cellHeight, GOL_SCHEME (█) *pScheme)
```

6.2.1.14.4 GridDraw Function**File**

Grid.h

C

```

WORD GridDraw(
    void * pObj
);

```

Module

Grid (█)

Input Parameters

Input Parameters	Description
pGb	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD GridDraw(void *pObj)
```

6.2.1.14.5 GridClearCellState Function**File**

Grid.h

C

```
void GridClearCellState(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD state
);
```

Module

Grid (grid)

Input Parameters

Input Parameters	Description
GRID * pGrid	Pointer to the object.
SHORT column	column index of the cell
SHORT row	row index of the cell
state	specifies the state to be cleared. See Grid (grid) Item State.

Side Effects

none

Returns

none.

Preconditions

none

Overview

This function clears the state of the cell (or Grid (█) Item) specified by the column and row.

Syntax

```
GridClearCellState(GRID (█) *pGrid, SHORT column, SHORT row, WORD state)
```

6.2.1.14.6 GridGetFocusX Macro**File**

Grid.h

C

```
#define GridGetFocusX(pGrid) pGrid->focusX
```

Module

Grid (█)

Input Parameters

Input Parameters	Description
pGrid	Pointer to the object.

Side Effects

none

Returns

Returns the x position of the focused cell.

Preconditions

none

Overview

This macro returns the x position of the focused cell.

Syntax

```
GridGetFocusX(pGrid)
```

6.2.1.14.7 GridGetFocusY Macro**File**

Grid.h

C

```
#define GridGetFocusY(pGrid) pGrid->focusY
```

Module

Grid (█)

Input Parameters

Input Parameters	Description
pGrid	Pointer to the object.

Side Effects

none

Returns

Returns the y position of the focused cell.

Preconditions

none

Overview

This macro returns the y position of the focused cell.

Syntax

```
GridGetFocusY(pGrid)
```

6.2.1.14.8 GRID_OUT_OF_BOUNDS Macro

File

Grid.h

C

```
#define GRID_OUT_OF_BOUNDS 0x0001 // Status of an out of bounds cell GridSetCell() operation.
```

Module

Grid (

Description

Status of an out of bounds cell GridSetCell ()() operation.

6.2.1.14.9 GRID_SUCCESS Macro

File

Grid.h

C

```
#define GRID_SUCCESS 0x0000 // Status of a successful GridSetCell() operation.
```

Module

Grid (

Description

Status of a successful GridSetCell ()() operation.

6.2.1.14.10 GridFreeItems Function

File

Grid.h

C

```
void GridFreeItems(  
    void * pObj  
) ;
```

Module

Grid (

Input Parameters

Input Parameters	Description
pGrid	The pointer to the Grid (grid) object.

Side Effects

none

Returns

none.

Preconditions

Object must be created before this function is called.

Overview

This function removes all grid items for the given Grid (grid) and frees the memory used.

Syntax

```
GridFreeItems(void *pObj)
```

6.2.1.14.11 GridGetCell Function

File

Grid.h

C

```
void * GridGetCell(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD * cellType
);
```

Module

Grid (grid)

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (grid) object.
SHORT column	the column index of the cell
SHORT row	the row index of the cell
WORD * cellType	pointer that will receive the type of grid item or cell (GRIDITEM_IS_TEXT (text) or GRIDITEM_IS_BITMAP (bitmap)).

Side Effects

none

Returns

Returns a pointer to the grid item or cell data.

Preconditions

Object must be created before this function is called.

Overview

This function removes all grid items for the given Grid (grid) and frees the memory used.

Syntax

```
*GridGetCell(GRID (grid) *pGrid, SHORT column, SHORT row, WORD *cellType)
```

6.2.1.14.12 GridSetCell Function

File

Grid.h

C

```
WORD GridSetCell(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD state,
    void * data
);
```

Module

Grid (

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid () object.
SHORT column	the column index of the cell
SHORT row	the row index of the cell
WORD state	sets the state of the Grid () Item specified.
void * data	pointer to the data used for the Grid () Item.

Side Effects

none

Returns

Returns the status of the operation

- GRID_SUCCESS () - if the set succeeded
- GRID_OUT_OF_BOUNDS () - if the row and column given results in an out of bounds location.

Preconditions

Object must be created before this function is called.

Overview

This function sets the Grid () Item state and data.

Syntax

```
WORD GridSetCell(GRID *pGrid, SHORT column, SHORT row, WORD state, void *data)
```

6.2.1.14.13 GridSetCellState Function

File

Grid.h

C

```
void GridSetCellState(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD state
);
```

Module

Grid (

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (grid) object.
SHORT column	the column index of the cell
SHORT row	the row index of the cell
WORD state	sets the state of the Grid (grid) Item specified.

Side Effects

none

Returns

none.

Preconditions

Object must be created before this function is called.

Overview

This function sets the state of the Grid (grid) Item or cell.

Syntax

```
GridSetCellState(GRID (grid) *pGrid, SHORT column, SHORT row, WORD state)
```

6.2.1.14.14 GridSetFocus Function**File**

Grid.h

C

```
void GridSetFocus(
    GRID * pGrid,
    SHORT column,
    SHORT row
);
```

Module

Grid (grid)

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (grid) object.
SHORT column	the column index of the cell
SHORT row	the row index of the cell

Side Effects

none

Returns

none.

Preconditions

Object must be created before this function is called.

Overview

This function sets the focus of the specified Grid (grid) Item or cell.

Syntax

```
GridSetFocus(GRID (■) *pGrid, SHORT column, SHORT row)
```

6.2.1.14.15 GridMsgDefault Function**File**

Grid.h

C

```
void GridMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Grid (■)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL message.
pGrid	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GRID_MSG_TOUCHED	Touch Screen	none	Grid (■) will have no state change because of this event.
GRID_MSG_ITEM_SELECTED	Keyboard	Set GRIDITEM_SELECTED (■), GRID_DRAW_ITEMS (■)	Grid (■) Item selected will be redrawn.
GRID_MSG_UP	Keyboard	Set GRIDITEM_DRAW (■), GRID_DRAW_ITEMS (■)	Grid (■) Item above the currently focused item will be redrawn.
GRID_MSG_DOWN	Keyboard	Set GRIDITEM_DRAW (■), GRID_DRAW_ITEMS (■)	Grid (■) Item below the currently focused item will be redrawn.
GRID_MSG_LEFT	Keyboard	Set GRIDITEM_DRAW (■), GRID_DRAW_ITEMS (■)	Grid (■) Item to the left of the currently focused item will be redrawn.

GRID_MSG_RIGHT	Keyboard	Set (, GRID_DRAW_ITEMS (GRIDITEM_DRAW	Grid () Item to the right of the currently focused item will be redrawn.

Syntax

```
GridMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG () *pMsg)
```

6.2.1.14.16 GridTranslateMsg Function**File**

Grid.h

C

```
WORD GridTranslateMsg(  
    void * pObj,  
    GOL_MSG * pMsg  
) ;
```

Module

Grid (

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pGrid	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- GRID_MSG_TOUCHED - when the grid object is touched.
- GRID_MSG_ITEM_SELECTED – when key scan SCAN_SPACE_PRESSED () or SCAN_CR_PRESSED () are detected.
- GRID_MSG_UP – when key scan SCAN_UP_PRESSED () is detected.
- GRID_MSG_DOWN – when key scan SCAN_DOWN_PRESSED () is detected.
- GRID_MSG_LEFT – when key scan SCAN_LEFT_PRESSED () is detected.
- GRID_MSG_RIGHT – when key scan SCAN_RIGHT_PRESSED () is detected.
- OBJ_MSG_INVALID – Button () is not affected

Preconditions

none

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GRID_MSG_TOUCHED	Touch Screen	none	If any touch events occurs and the x,y position falls in the face of the grid.

GRID_MSG_ITEM_SELECTED	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_SPACE_PRESSED (█) or SCAN_CR_PRESSED (█).
GRID_MSG_UP	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_UP_PRESSED (█).
GRID_MSG_DOWN	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_DOWN_PRESSED (█).
GRID_MSG_LEFT	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_LEFT_PRESSED (█).
GRID_MSG_RIGHT	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_RIGHT_PRESSED (█).
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
GridTranslateMsg(void *pObj, GOL_MSG (█) *pMsg)
```

6.2.1.14.17 GRID Structure**File**

Grid.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT numColumns;
    SHORT numRows;
    SHORT cellHeight;
    SHORT cellWidth;
    SHORT focusX;
    SHORT focusY;
    GRIDITEM * gridObjects;
} GRID;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (█)).
SHORT numColumns;	Number of columns drawn for the Grid (█).
SHORT numRows;	Number of rows drawn for the Grid (█).
SHORT cellHeight;	The height of each cell in pixels.
SHORT cellWidth;	The width of each cell in pixels.
SHORT focusX;	The x position of the cell to be focused.
SHORT focusY;	The y position of the cell to be focused.
GRIDITEM * gridObjects;	The pointer to grid items

Module

Grid (█)

Overview

Defines the parameters required for a grid Object. Clipping is not supported in grid object.

6.2.1.14.18 GRIDITEM Structure

File

Grid.h

C

```
typedef struct {
    void * data;
    WORD status;
} GRIDITEM;
```

Members

Members	Description
void * data;	Indicates if the Grid (■) Item is type GRIDITEM_IS_TEXT (■) or GRIDITEM_IS_BITMAP (■)
WORD status;	indicates the status of the Grid (■) Item

Module

Grid (■)

Overview

Defines the grid item.

6.2.1.15 Group Box

Group Box is an Object that can be used to group Objects together in the screen.

Functions

	Name	Description
✳	GbCreate (■)	This function creates a GROUPBOX (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
✳	GbDraw (■)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
✳	GbSetText (■)	This function sets the text used by passing the pointer to the static string.
✳	GbTranslateMsg (■)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Macros

Name	Description
GbGetText (■)	This macro returns the location of the text used.

Structures

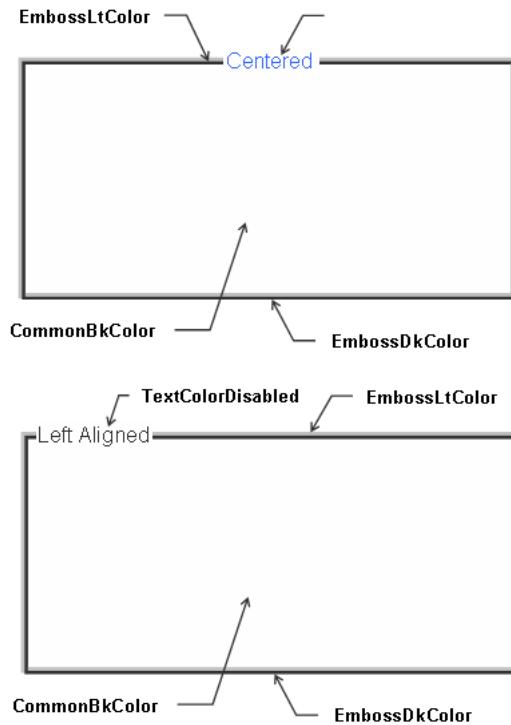
Name	Description
GROUPBOX (■)	Defines the parameters required for a group box Object. The textwidth and textHeight is not checked with the actual dimension of the object. Clipping is not supported in group box object. It is possible for the text to exceed the dimension of the Object.

Description

Group Box supports only Touchscreen inputs, replying to their events with the message:

GB_MSG_SELECTED - when the touch is within the dimension of the object.

The Group box object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



6.2.1.15.1 Group Box States

Macros

Name	Description
GB_CENTER_ALIGN (█)	Bit to indicate text is center aligned
GB_DISABLED (█)	Bit for disabled state
GB_DRAW (█)	Bit to indicate group box must be redrawn
GB_HIDE (█)	Bit to remove object from screen
GB_RIGHT_ALIGN (█)	Bit to indicate text is right aligned

Module

Group Box (█)

Description

List of Group Box bit states.

6.2.1.15.1.1 GB_CENTER_ALIGN Macro

File

GroupBox.h

C

```
#define GB_CENTER_ALIGN 0x0008 // Bit to indicate text is center aligned
```

Description

Bit to indicate text is center aligned

6.2.1.15.1.2 GB_DISABLED Macro

File

GroupBox.h

C

```
#define GB_DISABLED 0x0002 // Bit for disabled state
```

Description

Bit for disabled state

6.2.1.15.1.3 GB_DRAW Macro

File

GroupBox.h

C

```
#define GB_DRAW 0x4000 // Bit to indicate group box must be redrawn
```

Description

Bit to indicate group box must be redrawn

6.2.1.15.1.4 GB_HIDE Macro

File

GroupBox.h

C

```
#define GB_HIDE 0x8000 // Bit to remove object from screen
```

Description

Bit to remove object from screen

6.2.1.15.1.5 GB_RIGHT_ALIGN Macro

File

GroupBox.h

C

```
#define GB_RIGHT_ALIGN 0x0004 // Bit to indicate text is right aligned
```

Description

Bit to indicate text is right aligned

6.2.1.15.2 GbCreate Function

File

GroupBox.h

C

```
GROUPBOX * GbCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    GOL_SCHEME * pScheme
) ;
```

Module

Group Box (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
XCHAR * pText	The pointer to the text used for the group box. Length of string must be checked not to exceed the object's width. Clipping is not supported for the text of this object.
GOL_SCHEME * pScheme	Pointer to the style scheme used for the object. Set to NULL if default style scheme is used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
GROUPBOX *groupbox[ 2 ];
WORD state;

pScheme = GOLCreateScheme();
state = GB_DRAW | GB_RIGHT_ALIGN;
groupbox[ 0 ] = GbCreate( 10, 14,48,152,122,
                        state, "Power", scheme );
if (groupbox[ 0 ] == NULL)
    return 0;
state = GB_DRAW;
groupbox[ 1 ] = GbCreate( 11, 160,48,298,122,
                        state, "Pressure", scheme );
if (groupbox[ 1 ] == NULL)
    return 0;

while( !GbDraw(groupbox[ 0 ]));
while( !GbDraw(groupbox[ 1 ]));
return 1;
```

Overview

This function creates a GROUPBOX (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
GROUPBOX (█) *GbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR (█)
*pText, GOL_SCHEME (█) *pScheme)
```

6.2.1.15.3 GbDraw Function**File**

GroupBox.h

C

```
WORD GbDraw(
    void * pObj
);
```

Module

Group Box (█)

Input Parameters

Input Parameters	Description
pGb	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See GbCreate (█)() example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD GbDraw(void *pObj)
```

6.2.1.15.4 GbGetText Macro**File**

GroupBox.h

C

```
#define GbGetText(pB) pGb->pText
```

Module

Group Box (█)

Input Parameters

Input Parameters	Description
pGb	Pointer to the object.

Side Effects

none

Returns

Returns the address of the text string used.

Preconditions

none

Overview

This macro returns the location of the text used.

Syntax

GbGetText(pGb)

6.2.1.15.5 GbSetText Function

File

GroupBox.h

C

```
void GbSetText(
    GROUPBOX * pGb,
    XCHAR * pText
);
```

Module

Group Box (█)

Input Parameters

Input Parameters	Description
GROUPBOX * pGb	the pointer to the object whose state will be modified.
XCHAR * pText	pointer to the text that will be used.

Side Effects

Modifies the object width and height depending on the selected string width and font height.

Returns

none

Preconditions

The style scheme used for the object MUST be initialized with a valid font. If font is not valid, textWidth and textHeight parameter of GROUPBOX (█) will be undefined.

Overview

This function sets the text used by passing the pointer to the static string.

Syntax

GbSetText(GROUPBOX (█) *pGb, XCHAR (█) *pText)

6.2.1.15.6 GbTranslateMsg Function

File

GroupBox.h

C

```
WORD GbTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Group Box (█)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pGb	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- GB_MSG_SELECTED – Group Box is selected
- OBJ_MSG_INVALID – Group Box is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (█)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
GB_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the group box.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD GbTranslateMsg(void *pObj, GOL_MSG (█) *pMsg)
```

6.2.1.15.7 GROUPBOX Structure

File

GroupBox.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT textWidth;
    SHORT textHeight;
    XCHAR * pText;
} GROUPBOX;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
SHORT textWidth;	Pre-computed text width.
SHORT textHeight;	Pre-computed text height.
XCHAR * pText;	Text string used.

Module

Group Box (■)

Overview

Defines the parameters required for a group box Object. The textwidth and textHeight is not checked with the actual dimension of the object. Clipping is not supported in group box object. It is possible for the text to exceed the dimension of the Object.

6.2.1.16 List Box

List Box is an Object that defines a scrollable area where items are listed. User can select a single item or set of items.

Functions

	Name	Description
≡	LbCreate (■)	This function creates a LISTBOX (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
≡	LbDraw (■)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. The text or items drawn in the visible window of the list box is dependent on the alignment set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
≡	LbAddItem (■)	This function allocates memory for the LISTITEM (■) and adds it to the list box. The newly created LISTITEM (■) will store the location of pText, pBitmap and other parameters describing the added item.
≡	LbDellItem (■)	This function removes an item from the list box and frees the memory used.
≡	LbChangeSel (■)	This function changes the selection status of an item in the list box. If the item is currently selected, it resets the selection. If the item is currently not selected it is set to be selected.
≡	LbGetSel (■)	This function searches for selected items from the list box. A starting position can optionally be given. If starting position is set to NULL, search will begin from the first item list. It returns the pointer to the first selected item found or NULL if there are no items selected.
≡	LbGetFocusedItem (■)	This function returns the index of the focused item in the list box.
≡	LbSetFocusedItem (■)	This function sets the focus for the item with the given index.
≡	LbDellItemsList (■)	This function removes all items from the list box and frees the memory used.

	LbMsgDefault ([?])	This function performs the actual state change based on the translated message given. The following state changes are supported:
	LbTranslateMsg ([?])	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
LbGetItemList ([?])	This function returns the pointer to the current item list used in the list box.
LbSetSel ([?])	This macro sets the selection status of an item to selected.
LbGetCount ([?])	This macro returns the number of items in the list box.
LbGetVisibleCount ([?])	This macro returns the number of items visible in the list box window.
LbSetBitmap ([?])	This macro sets the bitmap used in the item.
LbGetBitmap ([?])	This macro returns the location of the currently used bitmap for the item.

Structures

Name	Description
LISTBOX ([?])	Defines the parameters required for a list box Object.
LISTITEM ([?])	Defines the parameters required for a list item used in list box.

Description

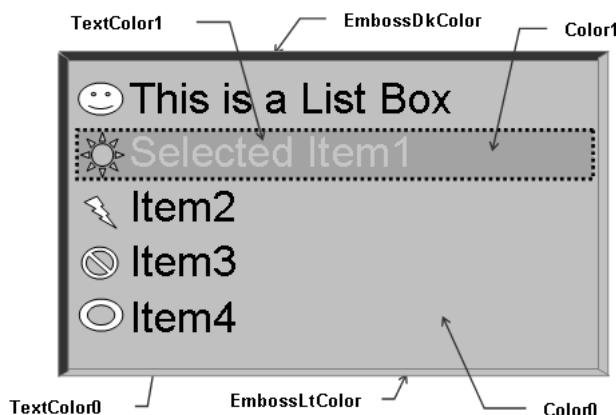
List Box supports both Touchscreen and Keyboard inputs, replying to their events with the following messages:

LB_MSG_TOUCHSCREEN – Item is selected using touch screen

LB_MSG_MOVE – Focus is moved to the next item depending on the key pressed (UP or DOWN key).

LB_MSG_SEL – Selection is set to the currently focused item.

The List Box Object is rendered using the assigned style scheme. The following figure illustrates the color assignments. Icons can be added to each item when adding items to the list using LbAddItem ([\[?\]](#)()).



6.2.1.16.1 List Box States

Macros

Name	Description
LB_RIGHT_ALIGN (图)	Bit to indicate text is left aligned
LB_SINGLE_SEL (图)	Bit to indicate the only item can be selected
LB_CENTER_ALIGN (图)	Bit to indicate text is center aligned
LB_DISABLED (图)	Bit for disabled state
LB_DRAW (图)	Bit to indicate whole edit box must be redrawn
LB_DRAW_FOCUS (图)	Bit to indicate whole edit box must be redrawn
LB_DRAW_ITEMS (图)	Bit to indicate whole edit box must be redrawn
LB_FOCUSED (图)	Bit for focused state
LB_HIDE (图)	Bit to remove object from screen

Module

List Box (图)

Description

List of List Box bit states.

6.2.1.16.1.1 LB_RIGHT_ALIGN Macro

File

ListBox.h

C

```
#define LB_RIGHT_ALIGN 0x0004 // Bit to indicate text is left aligned
```

Description

Bit to indicate text is left aligned

6.2.1.16.1.2 LB_SINGLE_SEL Macro

File

ListBox.h

C

```
#define LB_SINGLE_SEL 0x0010 // Bit to indicate the only item can be selected
```

Description

Bit to indicate the only item can be selected

6.2.1.16.1.3 LB_CENTER_ALIGN Macro

File

ListBox.h

C

```
#define LB_CENTER_ALIGN 0x0008 // Bit to indicate text is center aligned
```

Description

Bit to indicate text is center aligned

6.2.1.16.1.4 LB_DISABLED Macro

File

ListBox.h

C

```
#define LB_DISABLED 0x0002 // Bit for disabled state
```

Description

Bit for disabled state

6.2.1.16.1.5 LB_DRAW Macro

File

ListBox.h

C

```
#define LB_DRAW 0x4000 // Bit to indicate whole edit box must be redrawn
```

Description

Bit to indicate whole edit box must be redrawn

6.2.1.16.1.6 LB_DRAW_FOCUS Macro

File

ListBox.h

C

```
#define LB_DRAW_FOCUS 0x2000 // Bit to indicate whole edit box must be redrawn
```

Description

Bit to indicate whole edit box must be redrawn

6.2.1.16.1.7 LB_DRAW_ITEMS Macro

File

ListBox.h

C

```
#define LB_DRAW_ITEMS 0x1000 // Bit to indicate whole edit box must be redrawn
```

Description

Bit to indicate whole edit box must be redrawn

6.2.1.16.1.8 LB_FOCUSED Macro

File

ListBox.h

C

```
#define LB_FOCUSED 0x0001 // Bit for focused state
```

Description

Bit for focused state

6.2.1.16.1.9 LB_HIDE Macro

File

ListBox.h

C

```
#define LB_HIDE 0x8000 // Bit to remove object from screen
```

Description

Bit to remove object from screen

6.2.1.16.2 List Item Status

Macros

Name	Description
LB_STS_SELECTED (🔗)	Item is selected.
LB_STS_REDRAW (🔗)	Item is to be redrawn.

Module

List Box (🔗)

Description

List of items status.

6.2.1.16.2.1 LB_STS_SELECTED Macro

File

ListBox.h

C

```
#define LB_STS_SELECTED 0x0001 // Item is selected.
```

Description

Item is selected.

6.2.1.16.2.2 LB_STS_REDRAW Macro

File

ListBox.h

C

```
#define LB_STS_REDRAW 0x0002 // Item is to be redrawn.
```

Description

Item is to be redrawn.

6.2.1.16.3 LbCreate Function

File

ListBox.h

C

```
LISTBOX * LbCreate(
    WORD ID,
    SHORT left,
```

```

    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    GOL_SCHEME * pscheme
);

```

Module

List Box (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the Object.
WORD state	Sets the initial state of the object.
XCHAR * pText	Pointer to the initialization text for the items.
GOL_SCHEME * pScheme	Pointer to the style scheme.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```

#define LISTBOX_ID    10

const XCHAR ItemList[] = "Line1n" "Line2n";

GOL_SCHEME *pScheme;
LISTBOX *pLb;
XCHAR *pTemp;
WORD state, counter;

pScheme = GOLCreateScheme();
state = LB_DRAW;

// create an empty listbox with default style scheme
pLb = LbCreate( LISTBOX_ID,           // ID number
                10,10,150,200,      // dimension
                state,              // initial state
                NULL,               // set items to be empty
                NULL);              // use default style scheme

// check if Listbox was created
if (pLb == NULL)
    return 0;

// create the list of items to be placed in the listbox
// Add items (each line will become one item,
// lines must be separated by 'n' character)
pTemp = ItemList;
counter = 0;
while(*pTemp){
    // since each item is appended NULL is assigned to
    // LISTITEM pointer.
    if(NULL == LbAddItem(pLb, NULL, pTemp, NULL, 0, counter))
        break;
}

```

```

        while( (unsigned XCHAR)*pTemp++ > (unsigned XCHAR)31 );
        if( *(pTemp-1) == 0 )
            break;
        counter++;
    }
}

```

Overview

This function creates a LISTBOX ([\[?\]](#)) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

LISTBOX ([\[?\]](#)) *LbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR ([\[?\]](#))* pText, GOL_SCHEME ([\[?\]](#)) *pScheme)

6.2.1.16.4 LbDraw Function**File**

ListBox.h

C

```

WORD LbDraw(
    void * pObj
);

```

Module

List Box ([\[?\]](#))

Input Parameters

Input Parameters	Description
pLb	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text or items drawn in the visible window of the list box is dependent on the alignment set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

WORD LbDraw(void *pObj)

6.2.1.16.5 LbGetItemList Macro

File

ListBox.h

C

```
#define LbGetItemList(pLb) ((LISTITEM *)((LISTBOX *)pLb)->pItemList)
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box object.

Side Effects

none

Returns

Returns the pointer to the LISTITEM (■) used in the list box.

Preconditions

none

Example

See LbAddItem (■)() example.

Overview

This function returns the pointer to the current item list used in the list box.

Syntax

```
LISTITEM (■) LbGetItemList(LISTBOX (■)* pLb)
```

6.2.1.16.6 LbAddItem Function

File

ListBox.h

C

```
LISTITEM * LbAddItem(
    LISTBOX * pLb,
    LISTITEM * pPrevItem,
    XCHAR * pText,
    void * pBitmap,
    WORD status,
    WORD data
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
LISTBOX * pLb	The pointer to the list box object.
LISTITEM * pPrevItem	Pointer to the item after which a new item must be inserted, if this pointer is NULL, the item will be appended at the end of the items list.

XCHAR * pText	Pointer to the text that will be inserted. Text must persist in memory for as long as it is referenced by an item in the list box.
void * pBitmap	Pointer to the bitmap for the item. Bitmap must persist in memory for as long as it is referenced by the an item in the list box.
WORD status	This parameter specifies if the item being added will be selected or redrawn (LB_STS_SELECTED (■) or LB_STS_REDRAW (■)). Refer to LISTITEM (■) structure for details.
WORD data	User assigned data associated with the item.

Side Effects

none

Returns

Return a pointer to the item created, NULL if the operation was not successful.

Preconditions

none

Example

```

const XCHAR ItemList[] = "Line1n" "Line2n" "Line3n";

extern BITMAP_FLASH myIcon;
LISTBOX *pLb;
LISTITEM *pItem, *pItemList;
XCHAR *pTemp;

// Assume that pLb is pointing to an existing list box in memory
// that is empty (no list).

// Create the list of the list box

// Initialize this to NULL to indicate that items will be added
// at the end of the list if the list exist on the list box or
// start a new list if the list box is empty.
pItem = NULL;
pTemp = ItemList;
pItem = LbAddItem(pLb, pItem, pTemp, NULL, LB_STS_SELECTED, 1)
if(pItem == NULL)
    return 0;
LbSetBitmap(pItem, &myIcon);

// Adjust pTemp to point to the next line
while((unsigned XCHAR)*pTemp++ > (unsigned XCHAR)31);

// add the next item
pItem = LbAddItem(pLb, pItem, pTemp, NULL, 0, 2)
if(pItem == NULL)
    return 0;
LbSetBitmap(pItem, &myIcon);

// Adjust pTemp to point to the next line
while((unsigned XCHAR)*pTemp++ > (unsigned XCHAR)31);

// this time insert the next item after the first item on the list
pItem = LbGetItemList(pLb);
pItem = LbAddItem(pLb, pItem, pTemp, NULL, 0, 3)
if(pItem == NULL)
    return 0;
LbSetBitmap(pItem, &myIcon);

```

Overview

This function allocates memory for the LISTITEM (■) and adds it to the list box. The newly created LISTITEM (■) will store the location of pText, pBitmap and other parameters describing the added item.

Syntax

```
LISTITEM (■)* LbAddItem(LISTBOX (■) *pLb, LISTITEM (■) *pPrevItem, XCHAR (■) *pText, void* pBitmap, WORD status, WORD data)
```

6.2.1.16.7 LbDellItem Function**File**

ListBox.h

C

```
void LbDellItem(
    LISTBOX * pLb,
    LISTITEM * pItem
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
LISTBOX * pLb	The pointer to the list box object.
LISTITEM * pItem	The pointer to the item that will be removed.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function removes an item from the list box and frees the memory used.

Syntax

```
void LbDellItem(LISTBOX (■) *pLb, LISTITEM (■) *pItem)
```

6.2.1.16.8 LbChangeSel Function**File**

ListBox.h

C

```
void LbChangeSel(
    LISTBOX * pLb,
    LISTITEM * pItem
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
LISTBOX * pLb	The pointer to the list box object.
LISTITEM * pItem	The pointer to the item the selection status will be changed.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function changes the selection status of an item in the list box. If the item is currently selected, it resets the selection. If the item is currently not selected it is set to be selected.

Syntax

```
void LbChangeSel(LISTBOX (■) *pLb, LISTITEM (■) *pItem)
```

6.2.1.16.9 LbSetSel Macro**File**

ListBox.h

C

```
#define LbSetSel(pLb, pItem) \
    if(!(pItem->status & LB_STS_SELECTED)) \
        LbChangeSel((LISTBOX *)pLb, pItem);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box object.
pItem	The pointer to the item the selection status will be set.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro sets the selection status of an item to selected.

Syntax

```
LbSetSel(pLb, pItem)
```

6.2.1.16.10 LbGetSel Function**File**

ListBox.h

C

```
LISTITEM * LbGetSel(
```

```

LISTBOX * pLb,
LISTITEM * pFromItem
);

```

Module

List Box (■)

Input Parameters

Input Parameters	Description
LISTBOX * pLb	The pointer to the list box object.
LISTITEM * pFromItem	The pointer to the item the search must start from, if the pointer is NULL the search begins from the start of the items list.

Side Effects

none

Returns

pointer to the selected item, NULL if there are no items selected

Preconditions

none

Overview

This function searches for selected items from the list box. A starting position can optionally be given. If starting position is set to NULL, search will begin from the first item list. It returns the pointer to the first selected item found or NULL if there are no items selected.

Syntax

```
LISTITEM (■)* LbGetSel(LISTBOX (■) *pLb, LISTITEM (■) *pFromItem)
```

6.2.1.16.11 LbGetFocusedItem Function**File**

ListBox.h

C

```

SHORT LbGetFocusedItem(
    LISTBOX * pLb
);

```

Module

List Box (■)

Input Parameters

Input Parameters	Description
LISTBOX * pLb	The pointer to the list box object.

Side Effects

none

Returns

Returns the index of the focused item in the list box.

Preconditions

none

Overview

This function returns the index of the focused item in the list box.

Syntax

```
SHORT LbGetFocusedItem(LISTBOX (*) pLb)
```

6.2.1.16.12 LbSetFocusedItem Function**File**

ListBox.h

C

```
void LbSetFocusedItem(
    LISTBOX * pLb,
    SHORT index
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
LISTBOX * pLb	The pointer to the list box object.
SHORT index	The index number of the item to be focused. First item on the list is always indexed 0.

Side Effects

none

Returns

none.

Preconditions

none

Overview

This function sets the focus for the item with the given index.

Syntax

```
void LbSetFocusedItem(LISTBOX (*) pLb, SHORT index)
```

6.2.1.16.13 LbGetCount Macro**File**

ListBox.h

C

```
#define LbGetCount(pLb) ((LISTBOX *)pLb)->itemsNumber
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box object.

Side Effects

none

Returns

The number of items the list box contains.

Preconditions

none

Overview

This macro returns the number of items in the list box.

Syntax

```
LbGetCount(pLb)
```

6.2.16.14 LbGetVisibleCount Macro**File**

ListBox.h

C

```
#define LbGetVisibleCount(pLb) \
( \
    (((LISTBOX *)pLb)->hdr.bottom - ((LISTBOX *)pLb)->hdr.top - 2 * \
(GOL_EMBOSSED_SIZE + LB_INDENT)) / \
    ((LISTBOX \
*pLb)->textHeight \
) \
)
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box object.

Side Effects

none

Returns

The number of items visible in the list box window.

Preconditions

none

Overview

This macro returns the number of items visible in the list box window.

Syntax

```
LbGetVisibleCount(pLb)
```

6.2.16.15 LbSetBitmap Macro**File**

ListBox.h

C

```
#define LbSetBitmap(pItem, pBtmap) ((LISTITEM *)pItem)->pBitmap = pBtmap
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
pItem	Pointer to the item.
pBtmap	Pointer to the bitmap to be used.

Side Effects

none

Returns

none

Preconditions

none

Example

See LbAddItem (■)() example.

Overview

This macro sets the bitmap used in the item.

Syntax

LbSetBitmap(pItem, pBtmap)

6.2.1.16 LbGetBitmap Macro

File

ListBox.h

C

#define LbGetBitmap(pItem) ((LISTITEM *)pItem)->pBitmap

Module

List Box (■)

Input Parameters

Input Parameters	Description
pItem	Pointer to the list item.

Side Effects

none

Returns

Returns the pointer to the current bitmap used.

Preconditions

none

Example

```
// Assume pLb is initialized to an existing list box
LISTITEM *pItem;
void *pBitmap;

pItem = LbGetItemList(pLb);
pBitmap = LbGetBitmap(pItem);
```

Overview

This macro returns the location of the currently used bitmap for the item.

Syntax

```
LbGetBitmap(plItem)
```

6.2.1.16.17 LbDellItemsList Function**File**

ListBox.h

C

```
void LbDellItemsList(
    void * pObj
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box object.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function removes all items from the list box and frees the memory used.

Syntax

```
void LbDellItemsList(void *pObj)
```

6.2.1.16.18 LbMsgDefault Function**File**

ListBox.h

C

```
void LbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message

GOL_MSG * pMsg	The pointer to the GOL message.
pB	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
LB_MSG_TOUCHSCREEN	Touch Screen	Set LB_FOCUSED (■),	If focus is enabled, the focus state bit LB_FOCUSED (■) will be set. LB_DRAW_FOCUS (■) draw state bit will force
		Set LB_DRAW_FOCUS (■)	the List Box to be redrawn with focus.
		Set LB_DRAW_ITEMS (■)	List Box will be redrawn with selected item(s).
LB_MSG_MOVE	KeyBoard	Set LB_DRAW_ITEMS (■)	List Box will be redrawn with focus on one item.
LB_MSG_SEL	KeyBoard	Set LB_DRAW_ITEMS (■)	List Box will be redrawn with selection on the current item focused.

Syntax

```
void LbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.16.19 LbTranslateMsg Function**File**

ListBox.h

C

```
WORD LbTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

List Box (■)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pLB	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- LB_MSG_TOUCHSCREEN – Item is selected using touch screen.
- LB_MSG_MOVE – Focus is moved to the next item depending on the key pressed (UP or DOWN key).
- LB_MSG_SEL – Selection is set to the currently focused item.

Preconditions

none

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
LB_MSG_TOUCHSCREEN	Touch Screen	Any	Item is selected using touch screen.
LB_MSG_MOVE	Keyboard	EVENT_KEYSCAN	Focus is moved to the next item depending on the key pressed (UP or DOWN key).
LB_MSG_SEL	Keyboard	EVENT_KEYSCAN	LB_MSG_SEL – Selection is set to the currently focused item.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD LbTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.16.20 LISTBOX Structure**File**

ListBox.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    LISTITEM * pItemList;
    LISTITEM * pFocusItem;
    WORD itemsNumber;
    SHORT scrollY;
    SHORT textHeight;
} LISTBOX;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
LISTITEM * pItemLst;	Pointer to the list of items.
LISTITEM * pFocusItem;	Pointer to the focused item.
WORD itemsNumber;	Number of items in the list box.
SHORT scrollY;	Scroll (■) displacement for the list.
SHORT textHeight;	Pre-computed text height.

Module

List Box ([🔗](#))

Overview

Defines the parameters required for a list box Object.

6.2.1.16.21 LISTITEM Structure**File**

ListBox.h

C

```
typedef struct {
    void * pPrevItem;
    void * pNextItem;
    WORD status;
    XCHAR * pText;
    void * pBitmap;
    WORD data;
} LISTITEM;
```

Members

Members	Description
void * pPrevItem;	Pointer to the next item
void * pNextItem;	Pointer to the previous item
WORD status;	Specifies the status of the item.
XCHAR * pText;	Pointer to the text for the item
void * pBitmap;	Pointer to the bitmap
WORD data;	Some data associated with the item

Module

List Box ([🔗](#))

Overview

Defines the parameters required for a list item used in list box.

6.2.1.17 Meter

Meter is an Object that can be used to graphically display a sampled input.

Functions

	Name	Description
	MtrCreate (🔗)	This function creates a METER (🔗) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	MtrDraw (🔗)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. Depending on the defined settings, value of the meter will be displayed or hidden. Displaying the value will require a little bit more rendering time depending on the size of the meter and font used. When rendering objects of the same type, each object must be rendered completely before the rendering of the... more (🔗)

	MtrSetVal (🔗)	This function sets the value of the meter to the passed newVal. newVal is checked to be in the minValue-maxValue range inclusive. If newVal is not in the range, minValue maxValue is assigned depending on the given newVal if less than minValue or above maxValue.
	MtrMsgDefault (🔗)	This function performs the actual state change based on the translated message given. Meter value is set based on parameter 2 of the message given. The following state changes are supported:
	MtrTranslateMsg (🔗)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
MtrGetVal (🔗)	This macro returns the current value of the meter. Value is always in the minValue-maxValue range inclusive.
MtrDecVal (🔗)	This macro is used to directly decrement the value.
MtrIncVal (🔗)	This macro is used to directly increment the value.
MtrSetScaleColors (🔗)	Scale colors can be used to highlight values of the meter. User can set these colors to define the arc colors and scale colors. This also sets the color of the meter value when displayed. Limitation is that color settings are set to the following angles: Color Boundaries Type Whole Type Half Type Quarter Arc (🔗) 6 225 to 180 not used not used Arc (🔗) 5 179 to 135 179 to 135 not used Arc (🔗) 4 134 to 90 134 to 90 not used Arc (🔗) 3 89 to 45 89 to 45 89 to 45 Arc (🔗) 2 44 to 0 44... more (🔗)
MtrSetTitleFont (🔗)	This function sets the font of title.
MtrSetValueFont (🔗)	This function sets the font of value.
METER_TYPE (🔗)	<p>This is a compile time setting to select the type if meter shape. There are three types:</p> <ul style="list-style-type: none"> • MTR_WHOLE_TYPE - Meter drawn with 6 octants used. • MTR_HALF_TYPE - Meter drawn with semi circle shape. • MTR_QUARTER_TYPE - Meter drawn with quarter circle shape. <p>Set only one value at a time. This is done to save code space. User can define the colors of the arcs for each type. MTR_WHOLE_TYPE will use all the arc colors (arcColor1 - arcColor6) MTR_HALF_TYPE will use arc colors (arcColor5, arcColor4, arcColor3, arcColor2) MTR_QUARTER_TYPE will use arc colors (arcColor3, arcColor2) Set the meter type in Meter.h file and... more (🔗)</p>
MTR_ACCURACY (🔗)	Sets the meter accuracy to one decimal places when displaying the values. Application must multiply the minValue, maxValue and values passed to the widget by RESOLUTION.

Structures

Name	Description
METER (🔗)	Defines the parameters required for a meter Object. Depending on the type selected the meter is drawn with the defined shape parameters and values set on the given fields.

Description

There are three meter types that you can draw:

1. MTR_WHOLE_TYPE
2. MTR_HALF_TYPE
3. MTR_QUARTER_TYPE

It supports only System inputs, replying to the event EVENT_SET with the message: MTR_MSG_SET (see MtrTranslateMsg ([🔗](#))() for details). This action ID means that the message contains the new value of the meter on the parameter 2 of the message.

The Meter Object is rendered using the assigned style scheme, value range colors (see `MtrSetScaleColors()` for details) and compile time settings. The following figure illustrates the assignments for a MTR_HALF_TYPE meter.



Default Half Meter



Half Meter with Arc enabled

Note:
Normal, Critical and Danger Colors
are not part of the style scheme
struct. They are fields in the METER
struct.

6.2.1.17.1 Meter States

Macros

Name	Description
<code>MTR_DISABLED</code> (🔗)	Bit for disabled state.
<code>MTR_DRAW</code> (🔗)	Bit to indicate object must be redrawn.
<code>MTR_HIDE</code> (🔗)	Bit to indicate object must be removed from screen.
<code>MTR_RING</code> (🔗)	Bit for ring type, scales are drawn over the ring default is only scales drawn.
<code>MTR_DRAW_UPDATE</code> (🔗)	Bit to indicate an update only.

Module

`Meter` (🔗)

Description

List of Meter (🔗) bit states.

6.2.1.17.1.1 MTR_DISABLED Macro

File

`Meter.h`

C

```
#define MTR_DISABLED 0x0002      // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.17.1.2 MTR_DRAW Macro

File

Meter.h

C

```
#define MTR_DRAW 0x4000      // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

6.2.1.17.1.3 MTR_HIDE Macro

File

Meter.h

C

```
#define MTR_HIDE 0x8000      // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

6.2.1.17.1.4 MTR_RING Macro

File

Meter.h

C

```
#define MTR_RING 0x0004      // Bit for ring type, scales are drawn over the ring
```

Description

Bit for ring type, scales are drawn over the ring default is only scales drawn.

6.2.1.17.1.5 MTR_DRAW_UPDATE Macro

File

Meter.h

C

```
#define MTR_DRAW_UPDATE 0x1000      // Bit to indicate an update only.
```

Description

Bit to indicate an update only.

6.2.1.17.2 MtrCreate Function

File

Meter.h

C

```
METER * MtrCreate(
    WORD ID,
    SHORT left,
```

```

    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    SHORT value,
    SHORT minValue,
    SHORT maxValue,
    void * pTitleFont,
    void * pValueFont,
    XCHAR * pText,
    GOL_SCHEME * pscheme
);

```

Module

Meter (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
SHORT value	Initial value set to the meter.
SHORT minValue	The minimum value the meter will display.
SHORT maxValue	The maximum value the meter will display.
void * pTitleFont	Pointer to the font used for the Title.
XCHAR * pText	Pointer to the text label of the meter.
GOL_SCHEME * pScheme	Pointer to the font used for the Value. Pointer to the style scheme used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```

#define ID_METER 101

extern const FONT_FLASH GOLMediumFont;           // medium font
extern const FONT_FLASH GOLSsmallFont;           // small font

GOL_SCHEME *pMeterScheme;
METER *pMtr;

pMeterScheme = GOLCreateScheme();

pMtr = MtrCreate(
    ID_METER,                      // assign ID
    30, 50, 150, 180,               // set dimension
    MTR_DRAW|MTR_RING,             // draw object after creation
    0,                             // set initial value
    0, 100,                         // set minimum and maximum value
    (void*)&GOLMediumFont,          // set title font
    (void*)&GOLSsmallFont,           // set value font
    "Speed",                        // Text Label
    pMeterScheme);                 // style scheme

```

```

// check if meter was created
if (pMtr == NULL)
    return 0;

// Change range colors: Normal values to WHITE
//                                Critical values to BLUE
//                                Danger values to RED
// assume that WHITE, GREEN, YELLOW and RED have been defined.
MtrSetScaleColors(pMtr, WHITE, WHITE, WHITE, GREEN, YELLOW, RED);

// use GOLDraw() to draw the meter and all other objects you created
while(!GOLDraw());
// OR to draw the meter manually use this:
//while(!MtrDraw(pMtr));

```

Overview

This function creates a METER (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
METER (█) *MtrCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT value,
SHORT minValue, SHORT maxValue, void *pTitleFont, void *pValueFont, XCHAR (█) *pText, GOL_SCHEME (█)
*pScheme)
```

6.2.1.17.3 MtrDraw Function

File

Meter.h

C

```
WORD MtrDraw(
    void * pObj
);
```

Module

Meter (█)

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See MtrCreate (█)() example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is

determined by the style scheme set.

Depending on the defined settings, value of the meter will displayed or hidden. Displaying the value will require a little bit more rendering time depending on the size of the meter and font used.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD MtrDraw(void *pObj)
```

6.2.1.17.4 MtrSetVal Function

File

Meter.h

C

```
void MtrSetVal(
    METER * pMtr,
    SHORT newVal
);
```

Module

Meter ([¶](#))

Input Parameters

Input Parameters	Description
METER * pMtr	The pointer to the object.
SHORT newVal	New value to be set for the Meter (¶).

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the value of the meter to the passed newVal. newVal is checked to be in the minValue-maxValue range inclusive. If newVal is not in the range, minValue maxValue is assigned depending on the given newVal if less than minValue or above maxValue.

Syntax

```
MtrSetVal(METER (¶) *pMtr, SHORT newVal)
```

6.2.1.17.5 MtrGetVal Macro

File

Meter.h

C

```
#define MtrGetVal(pMtr) ((pMtr)->value)
```

Module

Meter ([¶](#))

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.

Side Effects

none

Returns

Returns current value of the meter.

Preconditions

none

Overview

This macro returns the current value of the meter. Value is always in the minValue-maxValue range inclusive.

Syntax

MtrGetVal(pMtr)

6.2.1.17.6 MtrDecVal Macro**File**

Meter.h

C

#define MtrDecVal(pMtr, deltaValue) MtrSetVal(pMtr, ((pMtr)->value - deltaValue))

ModuleMeter ([¶](#))**Input Parameters**

Input Parameters	Description
pMtr	Pointer to the object.
deltaValue	Number to be subtracted to the current Meter (¶) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly decrement the value.

Syntax

MtrDecVal(pMtr, deltaValue)

6.2.1.17.7 MtrIncVal Macro**File**

Meter.h

C

```
#define MtrIncVal(pMtr, deltaValue) MtrSetVal(pMtr, ((pMtr)->value + deltaValue))
```

Module

Meter ([🔗](#))

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.
deltaValue	Number to be added to the current Meter (🔗) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly increment the value.

Syntax

```
MtrIncVal(pMtr, deltaValue)
```

6.2.1.17.8 MtrSetScaleColors Macro

File

Meter.h

C

```
#define MtrSetScaleColors(pMtr, arc1, arc2, arc3, arc4, arc5, arc6) \
{ \
    pMtr->arcColor6 = arc6; \
    pMtr->arcColor5 = arc5; \
    pMtr->arcColor4 = arc4; \
    pMtr->arcColor3 = arc3; \
    pMtr->arcColor2 = arc2; \
    pMtr->arcColor1 = arc1; \
}
```

Module

Meter ([🔗](#))

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.
arc1	color for arc 1.
arc2	color for arc 2.
arc3	color for arc 3.
arc4	color for arc 4.
arc5	color for arc 5.
arc6	color for arc 6.

Side Effects

none

Returns

none

Preconditions

The object must be created (using MtrCreate ()) before a call to this macro is performed.

Overview

Scale colors can be used to highlight values of the meter. User can set these colors to define the arc colors and scale colors. This also sets the color of the meter value when displayed. Limitation is that color settings are set to the following angles: Color Boundaries Type Whole Type Half Type Quarter Arc () 6 225 to 180 not used not used Arc () 5 179 to 135 179 to 135 not used Arc () 4 134 to 90 134 to 90 not used Arc () 3 89 to 45 89 to 45 89 to 45 Arc () 2 44 to 0 44 to 0 44 to 0 Arc () 1 -45 to -1 not used not used As the meter is drawn colors are changed depending on the angle of the scale and label being drawn.

Syntax

```
MtrSetScaleColors(pMtr, arc1, arc2, arc3, arc4, arc5, arc6) { pMtr->arcColor6=arc6; pMtr->arcColor5=arc5;
pMtr->arcColor4=arc4; pMtr->arcColor3=arc3; pMtr->arcColor2=arc2; pMtr->arcColor1=arc1; }
```

6.2.1.17.9 MtrSetTitleFont Macro

File

Meter.h

C

```
#define MtrSetTitleFont(pMtr, pNewFont) (((METER *)pMtr)->pTitleFont = pNewFont)
```

Module

Meter ()

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.
pNewFont	Pointer to the new font used for the title.

Side Effects

none

Returns

N/A

Preconditions

Font must be created before this function is called.

Overview

This function sets the font of title.

Syntax

```
MtrSetTitleFont(pMtr, pNewFont) (((METER (*)pMtr)->pTitleFont = pNewFont)
```

6.2.1.17.10 MtrSetValueFont Macro

File

Meter.h

C

```
#define MtrSetValueFont(pMtr, pNewFont) (((METER *)pMtr)->pValueFont = pNewFont)
```

Module

Meter (

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.
pNewFont	Pointer to the new font used for the value.

Side Effects

none

Returns

N/A

Preconditions

Font must be created before this function is called.

Overview

This function sets the font of value.

Syntax

```
MtrSetValueFont(pMtr, pNewFont) (((METER ()*)pMtr)->pValueFont = pNewFont)
```

6.2.1.17.11 METER_TYPE Macro

File

Meter.h

C

```
#define METER_TYPE MTR_WHOLE_TYPE
```

Module

Meter (

Overview

This is a compile time setting to select the type if meter shape. There are three types:

- MTR_WHOLE_TYPE - Meter () drawn with 6 octants used.
- MTR_HALF_TYPE - Meter () drawn with semi circle shape.
- MTR_QUARTER_TYPE - Meter () drawn with quarter circle shape.

Set only one value at a time. This is done to save code space. User can define the colors of the arcs for each type.

MTR_WHOLE_TYPE will use all the arc colors (arcColor1 - arcColor6) MTR_HALF_TYPE will use arc colors (arcColor5, arcColor4, arcColor3, arcColor2) MTR_QUARTER_TYPE will use arc colors (arcColor3, arcColor2) Set the meter type in Meter.h file and arc colors using MtrSetScaleColors ((pMtr, arc1, arc2, arc3, arc4, arc5, arc6) macro.

6.2.1.17.12 MTR_ACCURACY Macro

File

Meter.h

C

```
#define MTR_ACCURACY 0x0008      // Sets the meter accuracy to one decimal places
```

Module

Meter ([🔗](#))

Description

Sets the meter accuracy to one decimal places when displaying the values. Application must multiply the minValue, maxValue and values passed to the widget by RESOLUTION.

6.2.1.17.13 MtrMsgDefault Function**File**

Meter.h

C

```
void MtrMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Meter ([🔗](#))

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL message.
pMtr	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function performs the actual state change based on the translated message given. Meter ([🔗](#)) value is set based on parameter 2 of the message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
MTR_MSG_SET	System	Set MTR_DRAW_UPDATE (🔗)	Meter (🔗) will be redrawn to update the needle position and value displayed.

Syntax

MtrMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG ([🔗](#))* pMsg)

6.2.1.17.14 MtrTranslateMsg Function**File**

Meter.h

C

```
WORD MtrTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Meter ([🔗](#))

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pMtr	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- MTR_MSG_SET - Meter ([🔗](#)) ID is given in parameter 1 for a TYPE_SYSTEM message.
- OBJ_MSG_INVALID - Meter ([🔗](#)) is not affected.

Preconditions

none

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
MTR_MSG_SET	System	EVENT_SET	If event set occurs and the meter ID is sent in parameter 1.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD MtrTranslateMsg(void *pObj, GOL_MSG (🔗) *pMsg)
```

6.2.1.17.15 METER Structure**File**

Meter.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    XCHAR * pText;
    SHORT value;
    SHORT minValue;
    SHORT maxValue;
    SHORT xCenter;
    SHORT yCenter;
    SHORT radius;
    SHORT xPos;
    SHORT yPos;
    GFX_COLOR arcColor6;
```

```
GFX_COLOR arcColor5;
GFX_COLOR arcColor4;
GFX_COLOR arcColor3;
GFX_COLOR arcColor2;
GFX_COLOR arcColor1;
void * pTitleFont;
void * pValueFont;
} METER;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (█)).
XCHAR * pText;	The text label of the meter.
SHORT value;	Current value of the meter.
SHORT minValue;	minimum value the meter can display
SHORT maxValue;	maximum value the meter can display (range is maxValue - minValue)
SHORT xCenter;	The x coordinate center position. This is computed automatically.
SHORT yCenter;	The y coordinate center position. This is computed automatically.
SHORT radius;	Radius of the meter, also defines the needle length.
SHORT xPos;	The current x position of the needle. This is computed automatically.
SHORT yPos;	The current y position of the needle. This is computed automatically.
GFX_COLOR arcColor6;	Arc (█) 6 color parameter.
GFX_COLOR arcColor5;	Arc (█) 5 color parameter
GFX_COLOR arcColor4;	Arc (█) 4 color parameter
GFX_COLOR arcColor3;	Arc (█) 3 color parameter
GFX_COLOR arcColor2;	Arc (█) 2 color parameter
GFX_COLOR arcColor1;	Arc (█) 1 color parameter
void * pTitleFont;	Pointer to the font used in the title of the meter
void * pValueFont;	Pointer to the font used in the current reading (if displayed) of the meter

Module

Meter (█)

Overview

Defines the parameters required for a meter Object. Depending on the type selected the meter is drawn with the defined shape parameters and values set on the given fields.

6.2.1.17.16 Files**6.2.1.18 Picture Control**

Picture is an Object that can be used to transform a bitmap to be an Object in the screen and have control on the bitmap rendering. This object can be used to create animation using a series of bitmaps.

Functions

	Name	Description
💡	PictCreate (█)	This function creates a PICTURE (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

	PictDraw (Link)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
	PictTranslateMsg (Link)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event accepted by the PICTURE (Link) Object.

Macros

Name	Description
PictSetBitmap (Link)	This macro sets the bitmap used in the object.
PictGetBitmap (Link)	This macro returns the pointer to the bitmap used in the object.
PictGetScale (Link)	This macro returns the current scale factor used to render the bitmap.
PictSetScale (Link)	This macro sets the scale factor used to render the bitmap used in the object.

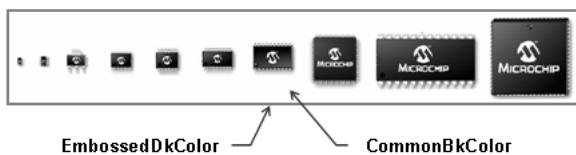
Structures

Name	Description
PICTURE (Link)	The structure contains data for picture control

Description

It supports only Keyboard inputs, replying to any touch screen events with the message: PICT_MSG_SELECTED.

The Picture Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.

**6.2.1.18.1 Picture States****Macros**

Name	Description
PICT_DISABLED (Link)	Bit to indicate Picture (Link) is in a disabled state.
PICT_DRAW (Link)	Bit to indicate Picture (Link) will be redrawn.
PICT_FRAME (Link)	Bit to indicate Picture (Link) has a frame.
PICT_HIDE (Link)	Bit to indicate Picture (Link) must be hidden.

Module

Picture Control ([Link](#))

Description

List of Picture ([Link](#)) Control bit states.

6.2.1.18.1.1 PICT_DISABLED Macro

File

Picture.h

C

```
#define PICT_DISABLED 0x0002 // Bit to indicate Picture is in a disabled state.
```

Description

Bit to indicate Picture (■) is in a disabled state.

6.2.1.18.1.2 PICT_DRAW Macro

File

Picture.h

C

```
#define PICT_DRAW 0x4000 // Bit to indicate Picture will be redrawn.
```

Description

Bit to indicate Picture (■) will be redrawn.

6.2.1.18.1.3 PICT_FRAME Macro

File

Picture.h

C

```
#define PICT_FRAME 0x0004 // Bit to indicate Picture has a frame.
```

Description

Bit to indicate Picture (■) has a frame.

6.2.1.18.1.4 PICT_HIDE Macro

File

Picture.h

C

```
#define PICT_HIDE 0x8000 // Bit to indicate Picture must be hidden.
```

Description

Bit to indicate Picture (■) must be hidden.

6.2.1.18.2 PictCreate Function

File

Picture.h

C

```
PICTURE * PictCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
```

```

    char scale,
    void * pBitmap,
    GOL_SCHEME * pScheme
);

```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
char scale	Sets the scale factor used to render the bitmap.
void * pBitmap	Pointer to the bitmap that will be used.
GOL_SCHEME * pScheme	Pointer to the style scheme
radius	Radius of the rounded edge.

Side Effects

none

Returns

Returns the pointer to the object created

Preconditions

none

Overview

This function creates a PICTURE (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
PICTURE (■) *PictCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, char scale, void
*pBitmap, GOL_SCHEME (■) *pScheme)
```

6.2.1.18.3 PictDraw Function

File

Picture.h

C

```

WORD PictDraw(
    void * pObj
);

```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
pPict	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD PictDraw(void *pObj)
```

6.2.1.18.4 PictSetBitmap Macro

File

Picture.h

C

```
#define PictSetBitmap(pPict, pBtmap) ((PICTURE*)pPict)->pBitmap = pBtmap
```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
pPict	Pointer to the object
pBtMap	Pointer to the bitmap to be used

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro sets the bitmap used in the object.

Syntax

```
PictSetBitmap(pPict,pBtMap)
```

6.2.1.18.5 PictGetBitmap Macro

File

Picture.h

C

```
#define PictGetBitmap(pPict) ((PICTURE*)pPict)->pBitmap
```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
pPict	Pointer to the object

Side Effects

none

Returns

Returns the pointer to the bitmap used.

Preconditions

none

Overview

This macro returns the pointer to the bitmap used in the object.

Syntax

```
PictGetBitmap(pPict)
```

6.2.1.18.6 PictGetScale Macro

File

Picture.h

C

```
#define PictGetScale(pPict) pPict->scale
```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
pPict	Pointer to the object

Side Effects

none

Returns

Returns the current scale factor used to display the bitmap.

Preconditions

none

Overview

This macro returns the current scale factor used to render the bitmap.

Syntax

```
PictGetScale(pPict,scl)
```

6.2.1.18.7 PictSetScale Macro

File

Picture.h

C

```
#define PictSetScale(pPict, scl) pPict->scale = scl
```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
pPict	Pointer to the object
scl	The scale factor that will be used to display the bitmap.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro sets the scale factor used to render the bitmap used in the object.

Syntax

```
PictSetScale(pPict,scl)
```

6.2.1.18.8 PictTranslateMsg Function

File

Picture.h

C

```
WORD PictTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Picture Control (■)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pPict	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- PICT_MSG_SELECTED – Picture (■) is touched.
- OBJ_MSG_INVALID – Picture (■) is not affected

Preconditions

none

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event accepted by the PICTURE (■) Object.

Translated Message	Input Source	Events	Description
PICT_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the area of the picture.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD PictTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.18.9 PICTURE Structure**File**

Picture.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    char scale;
    void * pBitmap;
    PUTIMAGE_PARAM partial;
} PICTURE;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
char scale;	Scale factor for the bitmap
void * pBitmap;	Pointer to the bitmap
PUTIMAGE_PARAM partial;	structure containing parital image data

Module

Picture Control (■)

Overview

The structure contains data for picture control

6.2.1.19 Progress Bar

Progress Bar (■) is an Object that can be used to display the progress of a task such as a file download or transfer.

Functions

	Name	Description
≡	PbCreate ()	This function creates a PROGRESSBAR () object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
≡	PbDraw ()	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
≡	PbSetRange ()	This function sets the range of the progress bar. Calling this function also resets the position equal to the new range value.
≡	PbSetPos ()	This function sets the position of the progress bar. Position should be in the given range inclusive.
≡	PbTranslateMsg ()	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Macros

Name	Description
PbGetRange ()	This macro returns the current range of the progress bar.
PbGetPos ()	This macro returns the current progress bar position.

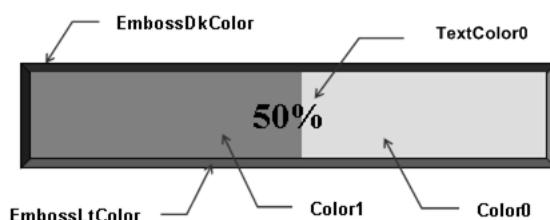
Structures

Name	Description
PROGRESSBAR ()	The structure contains data for the progress bar

Description

Progress Bar () supports only Touchscreen inputs, replying to their events with the message: PB_MSG_SELECTED.

The Progress Bar () Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



CommonBkColor – used to hide/remove the progress bar from the screen.

6.2.1.19.1 Progress Bar States

Macros

Name	Description
PB_DISABLED (█)	Bit to indicate Progress Bar (█) is in a disabled state.
PB_DRAW (█)	Bit to indicate Progress Bar (█) must be redrawn.
PB_DRAW_BAR (█)	Bit to indicate Progress Bar (█) must be redrawn.
PB_HIDE (█)	Bit to indicate Progress Bar (█) must be hidden.
PB_VERTICAL (█)	Bit for orientation (0 - horizontal, 1 - vertical)

Module

Progress Bar (█)

Description

List of Progress Bar (█) bit states.

6.2.1.19.1.1 PB_DISABLED Macro

File

ProgressBar.h

C

```
#define PB_DISABLED 0x0002 // Bit to indicate Progress Bar is in a disabled state.
```

Description

Bit to indicate Progress Bar (█) is in a disabled state.

6.2.1.19.1.2 PB_DRAW Macro

File

ProgressBar.h

C

```
#define PB_DRAW 0x4000 // Bit to indicate Progress Bar must be redrawn.
```

Description

Bit to indicate Progress Bar (█) must be redrawn.

6.2.1.19.1.3 PB_DRAW_BAR Macro

File

ProgressBar.h

C

```
#define PB_DRAW_BAR 0x2000 // Bit to indicate Progress Bar must be redrawn.
```

Description

Bit to indicate Progress Bar (█) must be redrawn.

6.2.1.19.1.4 PB_HIDE Macro

File

ProgressBar.h

C

```
#define PB_HIDE 0x8000 // Bit to indicate Progress Bar must be hidden.
```

Description

Bit to indicate Progress Bar (■) must be hidden.

6.2.1.19.1.5 PB_VERTICAL Macro**File**

ProgressBar.h

C

```
#define PB_VERTICAL 0x0004 // Bit for orientation (0 - horizontal, 1 - vertical)
```

Description

Bit for orientation (0 - horizontal, 1 - vertical)

6.2.1.19.2 PbCreate Function**File**

ProgressBar.h

C

```
PROGRESSBAR * PbCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    WORD pos,
    WORD range,
    GOL_SCHEME * pscheme
);
```

Module

Progress Bar (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the Object.
WORD state	Sets the initial state of the Object.
WORD pos	Defines the initial position of the progress.
WORD range	This specifies the maximum value of the progress bar when the progress bar is at 100% position.
GOL_SCHEME * pScheme	Pointer to the style scheme used for the object. Set to NULL if default style scheme is used.

Side Effects

none

Returns

Returns the pointer to the object created

Preconditions

none

Example

```
PROGRESSBAR *pPBar;
void CreateProgressBar(){
    pPBar = PbCreate(ID_PROGRESSBAR1,
                      50, 90, 270, 140,
                      PB_DRAW,
                      25,
                      50,
                      NULL);
    while( !PbDraw(pPBar));
}
```

Overview

This function creates a PROGRESSBAR (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
PROGRESSBAR (█) *PbCreate( WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, WORD pos, WORD range, GOL_SCHEME (█) *pScheme)
```

6.2.1.19.3 PbDraw Function

File

ProgressBar.h

C

```
WORD PbDraw(
    void * pObj
);
```

Module

Progress Bar (█)

Input Parameters

Input Parameters	Description
pPb	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See PbCreate (█)() example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is

determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD PbDraw(void *pObj)
```

6.2.1.19.4 PbSetRange Function

File

ProgressBar.h

C

```
void PbSetRange(
    PROGRESSBAR * pPb,
    WORD range
);
```

Module

Progress Bar (

Input Parameters

Input Parameters	Description
PROGRESSBAR * pPb	Pointer to the object

Side Effects

Sets the position equal to the new range.

Returns

none.

Preconditions

none

Overview

This function sets the range of the progress bar. Calling this function also resets the position equal to the new range value.

Syntax

```
PbSetRange(PROGRESSBAR () *pPb, WORD range)
```

6.2.1.19.5 PbGetRange Macro

File

ProgressBar.h

C

```
#define PbGetRange(pPb) (pPb->range)
```

Module

Progress Bar (

Input Parameters

Input Parameters	Description
pPb	Pointer to the object

Side Effects

none

Returns

Returns the range value.

Preconditions

none

Overview

This macro returns the current range of the progress bar.

Syntax

```
PbGetRange(pPb)
```

6.2.1.19.6 PbSetPos Function

File

ProgressBar.h

C

```
void PbSetPos(
    PROGRESSBAR * pPb,
    WORD position
);
```

Module

Progress Bar (■)

Input Parameters

Input Parameters	Description
PROGRESSBAR * pPb	Pointer to the object
WORD position	New position.

Side Effects

none

Returns

none

Preconditions

none

Example

```
PROGRESSBAR *pPb;
BYTE direction = 1;

// this code increments and decrements the progress bar by 1
// assume progress bar was created and initialized before
while (1) {
    if(direction) {
        if(pPb->pos == pPb->range)
            direction = 0;
        else
            PbSetPos(pPb, PbGetPos(pPb)+1);
    } else {
        if(pPb->pos == 0)
            direction = 1;
        else
            PbSetPos(pPb, PbGetPos(pPb)-1);
```

```

    }
}
```

Overview

This function sets the position of the progress bar. Position should be in the given range inclusive.

Syntax

```
void PbSetPos(PROGRESSBAR (■) *pPb, WORD position)
```

6.2.1.19.7 PbGetPos Macro**File**

ProgressBar.h

C

```
#define PbGetPos(pPb) pPb->pos
```

Module

Progress Bar (■)

Input Parameters

Input Parameters	Description
pPb	Pointer to the object

Side Effects

none

Returns

Returns the progress bar position.

Preconditions

none

Example

See PbSetPos (■)() example.

Overview

This macro returns the current progress bar position.

Syntax

```
PbGetPos(pPb)
```

6.2.1.19.8 PbTranslateMsg Function**File**

ProgressBar.h

C

```
WORD PbTranslateMsg(
    PROGRESSBAR * pPb,
    GOL_MSG * pMsg
);
```

Module

Progress Bar (■)

Input Parameters

Input Parameters	Description
PROGRESSBAR * pPb	The pointer to the object where the message will be evaluated to check if the message will affect the object.
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- PB_MSG_SELECTED – Progress Bar (█) is selected.
- OBJ_MSG_INVALID – Progress Bar (█) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (█)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
PB_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the area of the progress bar.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

WORD PbTranslateMsg(PROGRESSBAR (█) *pPb, GOL_MSG (█) *pMsg)

6.2.1.19.9 PROGRESSBAR Structure**File**

ProgressBar.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    WORD pos;
    WORD prevPos;
    WORD range;
} PROGRESSBAR;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (█)).
WORD pos;	Current progress position.
WORD prevPos;	Previous progress position.
WORD range;	Sets the range of the object.

Module

Progress Bar ([🔗](#))

Overview

The structure contains data for the progress bar

6.2.1.20 Radio Button

Radio Button ([🔗](#)) is an Object that can be used to offer set of choices to the user. Only one of the choices is selectable. Changing selection automatically removes the selection on the previous option.

Functions

Name	Description
 RbCreate (🔗)	This function creates a RADIobutton (🔗) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
 RbDraw (🔗)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
 RbGetCheck (🔗)	This function returns the ID of the currently checked Radio Button (🔗) in the group.
 RbSetCheck (🔗)	This function sets the Radio Button (🔗) with the given ID to its checked state.
 RbSetText (🔗)	This function sets the string used for the object.
 RbMsgDefault (🔗)	This function performs the actual state change based on the translated message given. The following state changes are supported:
 RbTranslateMsg (🔗)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
RbGetText (🔗)	This macro returns the address of the current text string used for the object.

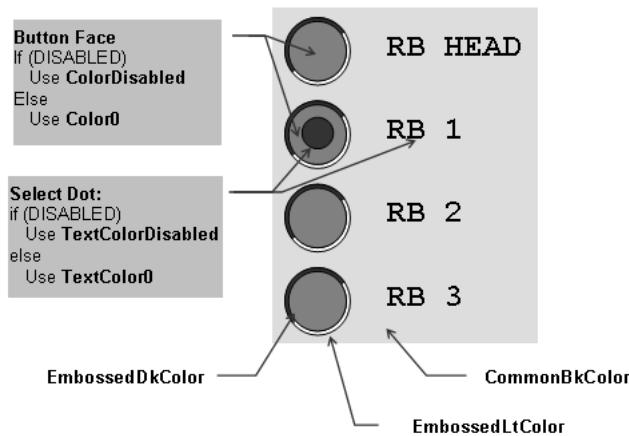
Structures

Name	Description
RADIOBUTTON (🔗)	the structure contains data for the Radio Button (🔗)

Description

Radio Button ([🔗](#)) supports both Keyboard and Touchscreen inputs, replying to their events with the message: RB_MSG_CHECKED.

The Radio Button ([🔗](#)) Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



6.2.1.20.1 Radio Button States

Macros

Name	Description
RB_CHECKED (█)	Bit to indicate Radio Button (█) is checked.
RB_DISABLED (█)	Bit for disabled state.
RB_DRAW (█)	Bit to indicate whole Radio Button (█) must be redrawn.
RB_DRAW_CHECK (█)	Bit to indicate check mark should be redrawn.
RB_DRAW_FOCUS (█)	Bit to indicate focus must be redrawn.
RB_FOCUSED (█)	Bit for focused state.
RB_GROUP (█)	Bit to indicate the first Radio Button (█) in the group.
RB_HIDE (█)	Bit to indicate that button must be removed from screen.

Module

Radio Button (█)

Description

List of Radio Button (█) bit states.

6.2.1.20.1.1 RB_CHECKED Macro

File

RadioButton.h

C

```
#define RB_CHECKED 0x0004 // Bit to indicate Radio Button is checked.
```

Description

Bit to indicate Radio Button (█) is checked.

6.2.1.20.1.2 RB_DISABLED Macro

File

RadioButton.h

C

```
#define RB_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.20.1.3 RB_DRAW Macro

File

RadioButton.h

C

```
#define RB_DRAW 0x4000 // Bit to indicate whole Radio Button must be redrawn.
```

Description

Bit to indicate whole Radio Button (▣) must be redrawn.

6.2.1.20.1.4 RB_DRAW_CHECK Macro

File

RadioButton.h

C

```
#define RB_DRAW_CHECK 0x1000 // Bit to indicate check mark should be redrawn.
```

Description

Bit to indicate check mark should be redrawn.

6.2.1.20.1.5 RB_DRAW_FOCUS Macro

File

RadioButton.h

C

```
#define RB_DRAW_FOCUS 0x2000 // Bit to indicate focus must be redrawn.
```

Description

Bit to indicate focus must be redrawn.

6.2.1.20.1.6 RB_FOCUSED Macro

File

RadioButton.h

C

```
#define RB_FOCUSED 0x0001 // Bit for focused state.
```

Description

Bit for focused state.

6.2.1.20.1.7 RB_GROUP Macro

File

RadioButton.h

C

```
#define RB_GROUP 0x0008 // Bit to indicate the first Radio Button in the group.
```

Description

Bit to indicate the first Radio Button (█) in the group.

6.2.1.20.1.8 RB_HIDE Macro**File**

RadioButton.h

C

```
#define RB_HIDE 0x8000 // Bit to indicate that button must be removed from screen.
```

Description

Bit to indicate that button must be removed from screen.

6.2.1.20.2 RbCreate Function**File**

RadioButton.h

C

```
RADIOBUTTON * RbCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    GOL_SCHEME * pScheme
);
```

Module

Radio Button (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object
SHORT bottom	Bottom most position of the object
WORD state	Sets the initial state of the object pText – The pointer to the text used for the Radio Button (█).
GOL_SCHEME * pScheme	Pointer to the style scheme used.

Side Effects

none

Returns

Returns the pointer to the object created

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
RADIOBUTTON *pRb[ 3 ];
```

```

pScheme = GOLCreateScheme();
pRb[0] = RbCreate(ID_RADIOBUTTON1,
                  255,40,310,80,
                  RB_DRAW|RB_GROUP|RB_CHECKED,
                  "RB1",
                  pScheme);                                // ID
                                                // dimension
                                                // will be displayed and
                                                // focused after creation
                                                // first button in the group
                                                // text
                                                // scheme used

pRb[1] = RbCreate(ID_RADIOBUTTON2,
                  255,85,310,125,
                  RB_DRAW,
                  "RB2",
                  pScheme);                                // ID
                                                // dimension
                                                // will be displayed and
                                                // checked after creation
                                                // text
                                                // scheme used

pRb[2] = RbCreate(ID_RADIOBUTTON3,
                  255,130,310,170,
                  RB_DRAW,
                  "RB3",
                  pScheme);                                // ID
                                                // dimension
                                                // will be displayed and
                                                // disabled after creation
                                                // text
                                                // scheme used

while( !RbDraw(pRb[0]));
while( !RbDraw(pRb[1]));
while( !RbDraw(pRb[2]));                                // draw the objects

```

Overview

This function creates a **RADIOBUTTON** (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
RADIOBUTTON (█) *RbCreate( WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR
(█) *pText, GOL_SCHEME (█) *pScheme)
```

6.2.1.20.3 RbDraw Function

File

RadioButton.h

C

```
WORD RbDraw(
    void * pObj
);
```

Module

Radio Button (█)

Input Parameters

Input Parameters	Description
pB	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See RbCreate (■)() example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD RbDraw(void *pObj)
```

6.2.1.20.4 RbGetCheck Function

File

RadioButton.h

C

```
WORD RbGetCheck(
    RADIOBUTTON * pRb
);
```

Module

Radio Button (■)

Input Parameters

Input Parameters	Description
RADIOBUTTON * pRb	Pointer to the Radio Button (■) in the group.

Side Effects

none

Returns

Returns the ID of the selected button in the group. It returns -1 if there is no object checked.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
RADIOBUTTON *pRb[3];
SHORT ID;

pScheme = GOLCreateScheme();
pRb[0] = RbCreate(ID_RADIOBUTTON1,
                  255,40,310,80,
                  RB_DRAW|RB_GROUP|RB_CHECKED,
                  "RB1",
                  pScheme); // ID
// dimension
// will be displayed and
// focused after creation
// first button in the group
// text
// scheme used

pRb[1] = RbCreate(ID_RADIOBUTTON2,
                  255,85,310,125,
                  RB_DRAW,
                  // ID
                  // dimension
                  // will be displayed and
```

```

        "RB2",
        pScheme);                                // checked after creation
                                                // text
                                                // scheme used

pRb[2] = RbCreate(ID_RADIOBUTTON3,
                   255, 130, 310, 170,
                   RB_DRAW,
                   "RB3",
                   pScheme);                                // ID
                                                // dimension
                                                // will be displayed and
                                                // disabled after creation
                                                // text
                                                // scheme used

// draw the Radio Buttons here

ID = RbGetCheck(pRb[2]);                      // can also use pRb[1] or
                                                // pRb[0] to search the
                                                // checked radio button of the
                                                // group. ID here should
                                                // be ID_RADIOBUTTON1

if (ID == ID_RADIOBUTTON1) {
    // do something here then clear the check
    ClrState(pRb[0], RB_CHECKED);
    // Change the checked object. Pointer used is any of the three.
    // the ID used will find the correct object to be checked
    RbSetCheck(pRb[3], ID_RADIOBUTTON2);
}

```

Overview

This function returns the ID of the currently checked Radio Button (▣) in the group.

Syntax

```
WORD RbGetCheck(RADIOBUTTON (▣) *pRb)
```

6.2.1.20.5 RbSetCheck Function

File

RadioButton.h

C

```
void RbSetCheck(
    RADIOBUTTON * pRb,
    WORD ID
);
```

Module

Radio Button (▣)

Input Parameters

Input Parameters	Description
RADIOBUTTON * pRb	Pointer to the Radio Button (▣) in the group.
WORD ID	ID of the object to be checked.

Side Effects

none

Returns

none

Preconditions

none

Example

See RbGetCheck (▣)() example.

Overview

This function sets the Radio Button (■) with the given ID to its checked state.

Syntax

```
void RbSetCheck(RADIOBUTTON (■) *pRb, WORD ID)
```

6.2.1.20.6 RbGetText Macro**File**

RadioButton.h

C

```
#define RbGetText(pRb) pRb->pText
```

Module

Radio Button (■)

Input Parameters

Input Parameters	Description
pRb	Pointer to the object

Side Effects

none

Returns

Returns pointer to the text string being used.

Preconditions

none

Overview

This macro returns the address of the current text string used for the object.

Syntax

```
RbGetText(pRb)
```

6.2.1.20.7 RbSetText Function**File**

RadioButton.h

C

```
void RbSetText(
    RADIOBUTTON * pRb,
    XCHAR * pText
);
```

Module

Radio Button (■)

Input Parameters

Input Parameters	Description
RADIOBUTTON * pRb	The pointer to the object whose text will be modified
XCHAR * pText	Pointer to the text that will be used

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the string used for the object.

Syntax

```
RbSetText(RADIOBUTTON (■) *pRb, XCHAR (■) *pText)
```

6.2.1.20.8 RbMsgDefault Function

File

RadioButton.h

C

```
void RbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Radio Button (■)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL message
pRb	The pointer to the object whose state will be modified

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg (■)() example.

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
RB_MSG_CHECKED	Touch Screen,	Set RB_DRAW (■),	Depending on the current value of RB_CHECKED (■) Check Box will be redrawn.

	Keyboard	Set/Clear RB_CHECKED (█)	
--	----------	--------------------------	--

Syntax

```
RbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (█)* pMsg)
```

6.2.1.20.9 RbTranslateMsg Function**File**

RadioButton.h

C

```
WORD RbTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Radio Button (█)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pRb	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- RB_MSG_CHECKED – Radio Button (█) is checked
- OBJ_MSG_INVALID – Radio Button (█) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (█)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
RB_MSG_CHECKED	Touch Screen	EVENT_PRESS	If event occurs and the x,y position falls in the area of the Radio Button (█) while the Radio Button (█) is not checked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED (█) or SCAN_SPACE_PRESSED (█) while the Radio Button (█) is not checked.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD RbTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.20.10 RADIobutton Structure**File**

RadioButton.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    OBJ_HEADER * pHead;
    OBJ_HEADER * pNext;
    SHORT textHeight;
    XCHAR * pText;
} RADIobutton;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
OBJ_HEADER * pHead;	Pointer to the first Radio Button (■) in the group
OBJ_HEADER * pNext;	Pointer to the next Radio Button (■) in the group
SHORT textHeight;	Pre-computed text height
XCHAR * pText;	Pointer to the text

Module

Radio Button (■)

Overview

the structure contains data for the Radio Button (■)

6.2.1.21 Slider/Scroll Bar

Slider or Scrollbar is an Object that can be used to display a value or scrolling location in a predefined area.

Functions

	Name	Description
■	SldCreate (■)	This function creates a SLIDER (■) object with the parameters given. Depending on the SLD_SCROLLBAR (■) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (■) is set, mode is scrollbar; if not set mode is slider. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
■	SldDraw (■)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
■	SldSetPage (■)	This sets the page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos (■)() or decremented via SldDecPos (■)(). Page size minimum value is 1. Maximum value is range/2.
■	SldSetPos (■)	This function sets the position of the slider thumb. Value should be in the set range inclusive. Object must be redrawn to reflect the change.
■	SldSetRange (■)	This sets the range of the thumb. If this field is changed Object must be completely redrawn to reflect the change.

	SldMsgDefault (🔗)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	SldTranslateMsg (🔗)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
SldGetPage (🔗)	Returns the current page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos (🔗 ()) or decremented via SldDecPos (🔗 ()). Page size minimum value is 1. Maximum value is range/2.
SldGetPos (🔗)	Returns returns the current position of the slider thumb.
SldGetRange (🔗)	Returns the current range of the thumb.
SldIncPos (🔗)	This macro increment the slider position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.
SldDecPos (🔗)	This macro decrement the slider position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Structures

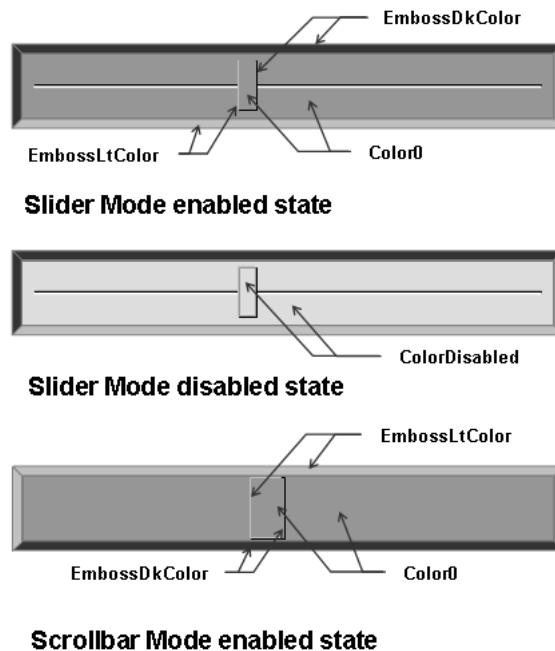
Name	Description
SLIDER (🔗)	Defines the parameters required for a slider/scrollbar Object. Depending on the SLD_SCROLLBAR (🔗) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (🔗) is set, mode is scrollbar; if not set mode is slider. For scrollbar mode, focus rectangle is not drawn.

Description

Slider or Scrollbar supports both Keyboard and Touchscreen inputs, replying to their events with the following messages:

1. SLD_MSG_INC - when the thumb is to be incremented in position
2. SLD_MSG_DEC - when the thumb is to be decremented in position

The Slider object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



6.2.1.21.1 Slider States

Macros

Name	Description
<code>SLD_DISABLED</code> (█)	Bit for disabled state
<code>SLD_DRAW</code> (█)	Bit to indicate whole slider must be redrawn
<code>SLD_DRAW_FOCUS</code> (█)	Bit to indicate that only the focus will be redrawn
<code>SLD_DRAW_THUMB</code> (█)	Bit to indicate that only thumb area must be redrawn
<code>SLD_FOCUSED</code> (█)	Bit for focus state
<code>SLD_HIDE</code> (█)	Bit to remove object from screen
<code>SLD_SCROLLBAR</code> (█)	Bit for type usage (0 - Slider (█), 1 - ScrollBar)
<code>SLD_VERTICAL</code> (█)	Bit for orientation (0 - horizontal, 1 - vertical)

Module

Slider/Scroll Bar (█)

Description

List of Slider (█) bit states.

6.2.1.21.1.1 SLD_DISABLED Macro

File

Slider.h

C

```
#define SLD_DISABLED 0x0002 // Bit for disabled state
```

Description

Bit for disabled state

6.2.1.21.1.2 SLD_DRAW Macro

File

Slider.h

C

```
#define SLD_DRAW 0x4000 // Bit to indicate whole slider must be redrawn
```

Description

Bit to indicate whole slider must be redrawn

6.2.1.21.1.3 SLD_DRAW_FOCUS Macro

File

Slider.h

C

```
#define SLD_DRAW_FOCUS 0x2000 // Bit to indicate that only the focus will be redrawn
```

Description

Bit to indicate that only the focus will be redrawn

6.2.1.21.1.4 SLD_DRAW_THUMB Macro

File

Slider.h

C

```
#define SLD_DRAW_THUMB 0x1000 // Bit to indicate that only thumb area must be redrawn
```

Description

Bit to indicate that only thumb area must be redrawn

6.2.1.21.1.5 SLD_FOCUSED Macro

File

Slider.h

C

```
#define SLD_FOCUSED 0x0001 // Bit for focus state
```

Description

Bit for focus state

6.2.1.21.1.6 SLD_HIDE Macro

File

Slider.h

C

```
#define SLD_HIDE 0x8000 // Bit to remove object from screen
```

Description

Bit to remove object from screen

6.2.1.21.1.7 SLD_SCROLLBAR Macro

File

Slider.h

C

```
#define SLD_SCROLLBAR 0x0010 // Bit for type usage (0 - Slider, 1 - ScrollBar)
```

Description

Bit for type usage (0 - Slider (█), 1 - ScrollBar)

6.2.1.21.1.8 SLD_VERTICAL Macro

File

Slider.h

C

```
#define SLD_VERTICAL 0x0004 // Bit for orientation (0 - horizontal, 1 - vertical)
```

Description

Bit for orientation (0 - horizontal, 1 - vertical)

6.2.1.21.2 SldCreate Function

File

Slider.h

C

```
SLIDER * SldCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    WORD range,
    WORD page,
    WORD pos,
    GOL_SCHEME * pscheme
);
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the Object.
WORD state	This defines the initial state of the Object.
WORD range	This specifies the maximum value of the slider when the thumb is on the rightmost position for a horizontal orientation and bottom most position for the vertical orientation. Minimum value is always at zero.

WORD page	This is the incremental change of the slider when user action requests to move the slider thumb. This value is added or subtracted to the current position of the thumb.
WORD pos	This defines the initial position of the thumb.
GOL_SCHEME * pScheme	The pointer to the style scheme used for the Object. Set to NULL if default style scheme is used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```

GOL_SCHEME *pScheme;
SLIDER *slider[3];
WORD state;

pScheme = GOLCreateScheme();

// create a slider with
//     range = [0 - 50000]
//     delta = 500
//     initial position = 25000
state = SLD_DRAW;
slider[0] = SldCreate( 5,
                      150, 145, 285, 181,
                      state,
                      50000, 500, 25000,
                      pScheme);
if (slider[0] == NULL)
    return 0;

// create a scrollbar with
//     range = [0 - 100]
//     delta = 20
//     initial position = 0

state = SLD_DRAW|SLD_SCROLLBAR;
slider[1] = SldCreate( 6,
                      150, 190, 285, 220,
                      state,
                      100, 20, 0,
                      pScheme);
if (slider[1] == NULL)
    return 0;

// create a vertical slider with
//     range = [0 - 30]
//     delta = 2
//     initial position = 20

state = SLD_DRAW|SLD_VERTICAL;
slider[2] = SldCreate( 7,
                      120, 145, 140, 220,
                      state,
                      30, 2, 20,
                      pScheme);
if (slider[2] == NULL)
    return 0;

// draw the sliders
while(!sldDraw(slider[0]));      // draw the horizontal slider
while(!sldDraw(slider[1]));      // draw the horizontal scroll bar

```

```
while(!sldDraw(slider[2]);           // draw the vertical slider
      return 1;
```

Overview

This function creates a SLIDER (█) object with the parameters given. Depending on the SLD_SCROLLBAR (█) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (█) is set, mode is scrollbar; if not set mode is slider. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
SLIDER (█) *SldCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT range,
                      SHORT page, GOL_SCHEME (█) *pScheme);
```

6.2.1.21.3 SldDraw Function**File**

Slider.h

C

```
WORD SldDraw(
    void * pObj
);
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See SldCreate (█)() example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD SldDraw(void *pObj)
```

6.2.1.21.4 SldSetPage Function

File

Slider.h

C

```
void SldSetPage(
    SLIDER * pSld,
    WORD newPage
);
```

Module

Slider/Scroll Bar ([🔗](#))

Input Parameters

Input Parameters	Description
SLIDER * pSld	Pointer to the object.
WORD newPage	Value of the new page used.

Side Effects

Position of the thumb may change when redrawn.

Returns

None.

Preconditions

none

Example

See SldIncPos ([🔗](#))() example.

Overview

This sets the page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos ([🔗](#))() or decremented via SldDecPos ([🔗](#))(). Page size minimum value is 1. Maximum value is range/2.

Syntax

```
void SldSetPage(SLIDER (🔗) *pSld, WORD newPage)
```

6.2.1.21.5 SldGetPage Macro

File

Slider.h

C

```
#define SldGetPage(pSld) (((SLIDER*)pSld)->page)
```

Module

Slider/Scroll Bar ([🔗](#))

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

Returns the current value of the slider page.

Preconditions

none

Example

```
WORD page;
SLIDER *pSld;

page = SldGetPage(pSld);
```

Overview

Returns the current page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos (█)() or decremented via SldDecPos (█)(). Page size minimum value is 1. Maximum value is range/2.

Syntax

```
SldGetPage(pSld)
```

6.2.1.21.6 SldSetPos Function

File

Slider.h

C

```
void SldSetPos(
    SLIDER * pSld,
    SHORT newPos
);
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
SLIDER * pSld	Pointer to the object.
SHORT newPos	The new position of the slider's thumb.

Side Effects

none

Returns

none

Preconditions

none

Example

```
SLIDER *pSlider;
DWORD ctr = 0;

// create slider here and initialize parameters
SetState(pSlider, SLD_DRAW);
SldDraw(pSlider);

while("some condition") {
    SldSetPos(pSlider, ctr);
    SetState(pSlider, SLD_DRAW_THUMB);
    SldDraw(pSlider);
    // update ctr here
}
```

```
    ctr = "some source of value";
}
```

Overview

This function sets the position of the slider thumb. Value should be in the set range inclusive. Object must be redrawn to reflect the change.

Syntax

```
SldSetPos(SLIDER (■) *pSld, SHORT newPos)
```

6.2.1.21.7 SldGetPos Macro

File

Slider.h

C

```
#define SldGetPos(pSld) (((SLIDER*)pSld)->pos)
```

Module

Slider/Scroll Bar (■)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

Returns the current position of the slider's thumb.

Preconditions

none

Example

```
#define MAXVALUE 100;

SLIDER *pSlider;
DWORD ctr = 0;

// create slider here and initialize parameters
SetState(pSlider, SLD_DRAW);
SldDraw(pSlider);

while("some condition") {
    SldSetPos(pSlider, ctr);
    SetState(pSlider, SLD_DRAW_THUMB);
    SldDraw(pSlider);
    // update ctr here
    ctr = "some source of value";
}

if (SldGetPos(pSlider) > (MAXVALUE - 1))
    return 0;
else
    "do something else"
```

Overview

Returns returns the current position of the slider thumb.

Syntax

```
SldGetPos(pSld)
```

6.2.1.21.8 SldSetRange Function

File

Slider.h

C

```
void SldSetRange(
    SLIDER * pSld,
    SHORT newRange
);
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
SLIDER * pSld	Pointer to the object.
SHORT newRange	Value of the new range used.

Side Effects

Position of the thumb may change when redrawn.

Returns

None.

Preconditions

none

Example

```
SLIDER *pSld;

SldSetRange(pSld, 100);
// to completely redraw the object when GOLDraw() is executed.
SetState(pSld, SLD_DRAW);
```

Overview

This sets the range of the thumb. If this field is changed Object must be completely redrawn to reflect the change.

Syntax

SldSetRange(SLIDER (█) *pSld, SHORT newRange)

6.2.1.21.9 SldGetRange Macro

File

Slider.h

C

```
#define SldGetRange(pSld) (((SLIDER*)pSld)->range)
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

Returns the current range of the slider thumb.

Preconditions

none

Example

```
WORD range;
SLIDER *pSld;

range = SldGetRange(pSld);
```

Overview

Returns the current range of the thumb.

Syntax

```
SldGetRange(pSld)
```

6.2.1.21.10 SldIncPos Macro**File**

Slider.h

C

```
#define SldIncPos(pSld) \
    SldSetPos \
    ( \
    \_ \
        pSld, \
        \
        (((DWORD) pSld->pos + (DWORD) ((SLIDER*)pSld)->page) <= (DWORD) \
        ((SLIDER*)pSld)->range) ? \
            (((SLIDER*)pSld)->pos + ((SLIDER*)pSld)->page) : \
        ((SLIDER*)pSld)->range \
    )
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

none

Preconditions

none

Example

```
void ControlSpeed(SLIDER* pSld, int setSpeed, int curSpeed)
{
    SldSetPage(pSld, 1);                                // set page size to 1
    if (setSpeed < curSpeed) {
        while(SldGetPos(pSld) < SetSpeed)
            SldIncPos(pSld);                            // increment by 1
    }
    else if (setSpeed > curSpeed) {
        while(SldGetPos(pSld) > SetSpeed)
            SldDecPos(pSld);                            // decrement by 1
    }
}
```

Overview

This macro increment the slider position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Syntax

```
SldIncPos(pSld)
```

6.2.1.21.11 SldDecPos Macro**File**

```
Slider.h
```

C

```
#define SldDecPos(pSld) \
    SldSetPos \
    \
    ( \
    \
    \      pSld, \
    \
    \      (((LONG) ((SLIDER*)pSld)->pos - (LONG) ((SLIDER*)pSld)->page) >= 0) \
?           \
?               (((SLIDER*)pSld)->pos - ((SLIDER*)pSld)->page) : \
0           \
)
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

none

Preconditions

none

Example

See SldIncPos (█)() example.

Overview

This macro decrement the slider position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Syntax

```
SldDecPos(pSld)
```

6.2.1.21.12 SldMsgDefault Function

File

Slider.h

C

```
void SldMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL message.
pSld	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (█)() example.

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
SLD_MSG_INC	Touch Keyboard	Screen, Set SLD_DRAW_THUMB (█)	Slider (█) will be redrawn with thumb in the incremented position.
SLD_MSG_DEC	Touch Keyboard	Screen, Set SLD_DRAW_THUMB (█)	Slider (█) will be redrawn with thumb in the decremented position.

Syntax

```
void SldMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (█)* pMsg)
```

6.2.1.21.13 SldTranslateMsg Function

File

Slider.h

C

```
WORD SldTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Slider/Scroll Bar (█)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pSld	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- SLD_MSG_INC – Increment slider position
- SLD_MSG_DEC – Decrement slider position
- OBJ_MSG_PASSIVE – Slider (█) is not affected
- OBJ_MSG_INVALID – Slider (█) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (█)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
SLD_MSG_INC	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the area of the slider and the slider position is to the LEFT of the x,y position for a horizontal slider or BELOW the x,y position for a vertical slider.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_UP_PRESSED (█) or SCAN_LEFT_PRESSED (█).
SLD_MSG_DEC	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the area of the slider and the slider position is to the RIGHT of the x,y position for a horizontal slider or ABOVE the x,y position for a vertical slider.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_DOWN_PRESSED (█) or SCAN_RIGHT_PRESSED (█).

OBJ_MSG_PASSIVE	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the slider.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD SldTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.21.14 SLIDER Structure**File**

Slider.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    WORD currPos;
    WORD prevPos;
    WORD range;
    WORD pos;
    WORD page;
    WORD thWidth;
    WORD thHeight;
} SLIDER;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
WORD currPos;	Position of the slider relative to minimum.
WORD prevPos;	Previous position of the slider relative to minimum.
WORD range;	User defined range of the slider. Minimum value at 0 and maximum at 0x7FFF.
WORD pos;	Position of the slider in range domain.
WORD page;	User specified resolution to incrementally change the position
WORD thWidth;	Thumb width. This is computed internally.
WORD thHeight;	Thumb width. This is computed internally. User must not change this value.

Module

Slider/Scroll Bar (■)

Overview

Defines the parameters required for a slider/scrollbar Object. Depending on the SLD_SCROLLBAR (■) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (■) is set, mode is scrollbar; if not set mode is slider. For scrollbar mode, focus rectangle is not drawn.

6.2.1.22 Static Text

Static Text is an Object that can be used to display a single or multi-line string of text in a predefined location.

Functions

	Name	Description
💡	StCreate (■)	This function creates a STATICTEXT (■) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

	StDraw ()	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
	StSetText ()	This function sets the string that will be used for the object.
	StTranslateMsg ()	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Macros

Name	Description
StGetText ()	This macro returns the address of the current text string used for the object.

Structures

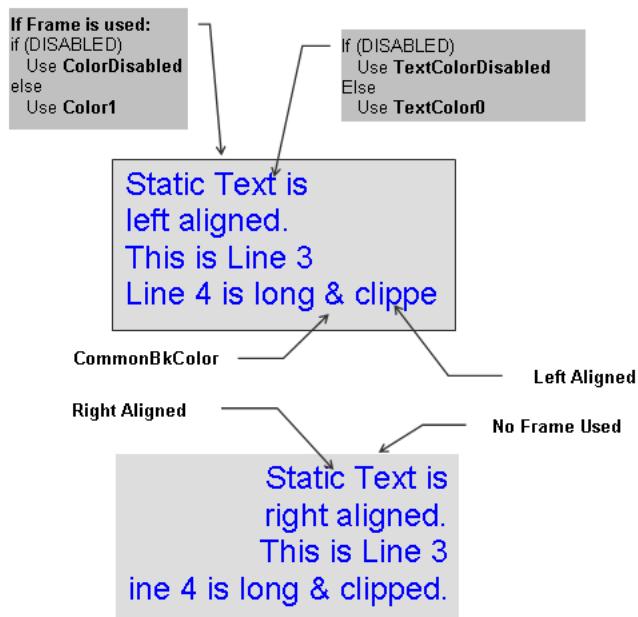
Name	Description
STATICTEXT ()	Defines the parameters required for a Static Text Object.

Description

Static Text supports only Touchscreen inputs, replying to their events with the message:

ST_MSG_SELECTED - when the touch is within the dimension of the object.

The Static Text Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



6.2.1.22.1 Static Text States

Macros

Name	Description
ST_CENTER_ALIGN (🔗)	Bit to indicate text is center aligned.
ST_DISABLED (🔗)	Bit for disabled state.
ST_DRAW (🔗)	Bit to indicate static text must be redrawn.
ST_FRAME (🔗)	Bit to indicate frame is displayed.
ST_HIDE (🔗)	Bit to remove object from screen.
ST_RIGHT_ALIGN (🔗)	Bit to indicate text is left aligned.
ST_UPDATE (🔗)	Bit to indicate that text area only is redrawn.

Module

Static Text (🔗)

Description

List of Static Text bit states.

6.2.1.22.1.1 ST_CENTER_ALIGN Macro

File

StaticText.h

C

```
#define ST_CENTER_ALIGN 0x0008 // Bit to indicate text is center aligned.
```

Description

Bit to indicate text is center aligned.

6.2.1.22.1.2 ST_DISABLED Macro

File

StaticText.h

C

```
#define ST_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.22.1.3 ST_DRAW Macro

File

StaticText.h

C

```
#define ST_DRAW 0x4000 // Bit to indicate static text must be redrawn.
```

Description

Bit to indicate static text must be redrawn.

6.2.1.22.1.4 ST_FRAME Macro

File

StaticText.h

C

```
#define ST_FRAME 0x0010 // Bit to indicate frame is displayed.
```

Description

Bit to indicate frame is displayed.

6.2.1.22.1.5 ST_HIDE Macro

File

StaticText.h

C

```
#define ST_HIDE 0x8000 // Bit to remove object from screen.
```

Description

Bit to remove object from screen.

6.2.1.22.1.6 ST_RIGHT_ALIGN Macro

File

StaticText.h

C

```
#define ST_RIGHT_ALIGN 0x0004 // Bit to indicate text is left aligned.
```

Description

Bit to indicate text is left aligned.

6.2.1.22.1.7 ST_UPDATE Macro

File

StaticText.h

C

```
#define ST_UPDATE 0x2000 // Bit to indicate that text area only is redrawn.
```

Description

Bit to indicate that text area only is redrawn.

6.2.1.22.2 StCreate Function

File

StaticText.h

C

```
STATICTEXT * StCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
```

```
XCHAR * pText,
GOL_SCHEME * pScheme
);
```

Module

Static Text (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
XCHAR * pText	Pointer to the text used in the static text.
GOL_SCHEME * pScheme	Pointer to the style scheme. Set to NULL if default style scheme is used.

Side Effects

none

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
STATICTEXT *pSt;

pScheme = GOLCreateScheme();
state = ST_DRAW | ST_FRAME | ST_CENTER_ALIGN;
StCreate(ID_STATICTEXT1,           // ID
         30,80,235,160,        // dimension
         state,                // has frame and center aligned
         "Static Textn Example", // text
         pScheme);              // use given scheme

while(!StDraw(pSt));             // draw the object
```

Overview

This function creates a STATICTEXT (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
STATICTEXT (█) *StCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state , XCHAR (█)
*pText, GOL_SCHEME (█) *pScheme)
```

6.2.1.22.3 StDraw Function**File**

StaticText.h

C

```
WORD StDraw(
    void * pObj
);
```

Module

Static Text (█)

Input Parameters

Input Parameters	Description
pSt	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See StCreate (█)() Example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD StDraw(void *pObj)
```

6.2.1.22.4 StGetText Macro**File**

StaticText.h

C

```
#define StGetText(pSt) pSt->pText
```

Module

Static Text (█)

Input Parameters

Input Parameters	Description
pSt	Pointer to the object.

Side Effects

none

Returns

Returns the pointer to the text string used.

Preconditions

none

Overview

This macro returns the address of the current text string used for the object.

Syntax

```
StGetText(pSt)
```

6.2.1.22.5 StSetText Function

File

StaticText.h

C

```
void StSetText(
    STATICTEXT * pSt,
    XCHAR * pText
);
```

Module

Static Text (

Input Parameters

Input Parameters	Description
STATICTEXT * pSt	The pointer to the object whose text string will be modified.
XCHAR * pText	The pointer to the string that will be used.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the string that will be used for the object.

Syntax

```
StSetText(STATICTEXT () *pSt, XCHAR () *pText)
```

6.2.1.22.6 StTranslateMsg Function

File

StaticText.h

C

```
WORD StTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Static Text (

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pSt	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- ST_MSG_SELECTED – Static Text is selected
- OBJ_MSG_INVALID – Static Text is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (■)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
ST_MSG_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the static text.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

WORD StTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)

6.2.1.22.7 STATICTEXT Structure**File**

StaticText.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT textHeight;
    XCHAR * pText;
} STATICTEXT;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
SHORT textHeight;	Pre-computed text height.
XCHAR * pText;	The pointer to text used.

Module

Static Text (■)

Overview

Defines the parameters required for a Static Text Object.

6.2.1.23 Text Entry

Text Entry is an Object that can be used to emulate a key pad entry with a display area for the entered characters. The Object has a feature where you can define a key to reply with a translated message that signifies a command key was pressed. A command key example can be your enter or carriage return key or an escape key. Multiple keys can be assigned command keys. Application can utilize the command key to define the behavior of the program based on a command key press.

Functions

	Name	Description
≡	TeCreate ()	This function creates a TEXTENTRY () object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
≡	TeDraw ()	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. This widget will draw the keys using the function GOLPanelDraw ()(). The number of keys will depend on the horizontal and vertical parameters given (horizontalKeys*verticalKeys).
≡	TeSetBuffer ()	This function sets the buffer used to display text. If the buffer is initialized with a string, the string must be a null terminated string. If the string length is greater than MaxSize, string will be truncated to MaxSize. pText must point to a valid memory location with size equal to MaxSize+1. The +1 is used for the string terminator.
≡	TeClearBuffer ()	This function will clear the data in the display. You must set the drawing state bit TE_UPDATE_TEXT () to update the TEXTENTRY () on the screen.
≡	TeGetKeyCommand ()	This function will return the currently used command by a key with the given index.
≡	TeSetKeyCommand ()	This function will assign a command (TE_DELETE_COM (), TE_SPACE_COM () or TE_ENTER_COM ()) to a key with the given index.
≡	TeCreateKeyMembers ()	This function will create the list of KEYMEMBERS that holds the information on each key. The number of keys is determined by the equation (verticalKeys*horizontalKeys). The object creates the information holder for each key automatically and assigns each entry in the *pText[] array with the first entry automatically assigned to the key with an index of 1. The number of entries to *pText[] must be equal or greater than (verticalKeys*horizontalKeys). The last key is assigned with an index of (verticalKeys*horizontalKeys)-1. No checking is performed on the length of *pText[] entries to match (verticalKeys*horizontalKeys).
≡	TeAddChar ()	This function will insert a character to the end of the buffer. The character inserted is dependent on the currently pressed key. Drawing states TE_UPDATE_TEXT () or TE_DRAW () must be set to see the effect of this insertion.
≡	TelsKeyPressed ()	This function will test if a key given by its index in the TextEntry object has been pressed.
≡	TeSpaceChar ()	This function will insert a space character to the end of the buffer. Drawing states TE_UPDATE_TEXT () or TE_DRAW () must be set to see the effect of this insertion.
≡	TeDelKeyMembers ()	This function will delete the KEYMEMBER () list assigned to the object from memory. Pointer to the KEYMEMBER () list is then initialized to NULL.
≡	TeSetKeyText ()	This function will set the test assigned to a key with the given index.
≡	TeMsgDefault ()	This function performs the actual state change based on the translated message given. The following state changes are supported:

	TeTranslateMsg (█)	This function evaluates the message from a user if the message will affect the object or not. If the message is valid, the keys in the Text Entry object will be scanned to detect which key was pressed. If True, the corresponding text will be displayed, the 'text' will also be stored in the TeOutput parameter of the object.
---	--------------------	--

Macros

Name	Description
TeGetBuffer (█)	This macro will return the currently used buffer in the TextEntry object.

Structures

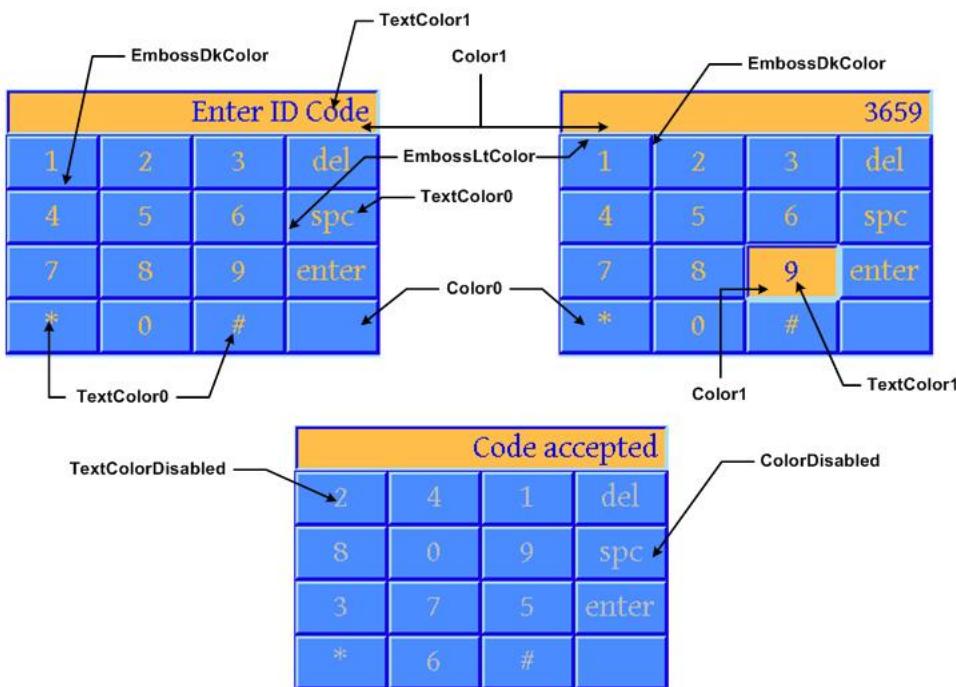
Name	Description
TEXTENTRY (█)	Defines the parameters required for a TextEntry Object.
KEYMEMBER (█)	Defines the parameters and the strings assigned for each key.

Description

Text Entry supports only Touchscreen inputs, replying to their events with the following messages:

1. TE_MSG_RELEASE – A key has been released.
2. TE_MSG_PRESS – A key is pressed.
3. TE_MSG_ADD_CHAR – A key was released with character assigned.
4. TE_MSG_DELETE – A key was released with delete command assigned.
5. TE_MSG_SPACE - A key was released with space command assigned.
6. TE_MSG_ENTER - A key was released with enter command assigned.

The Text Entry Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



6.2.1.23.1 TextEntry States

Macros

Name	Description
TE_KEY_PRESSED (🔗)	Bit for press state of one of the keys.
TE_DISABLED (🔗)	Bit for disabled state.
TE_ECHO_HIDE (🔗)	Bit to hide the entered characters and instead echo "*" characters to the display.
TE_DRAW (🔗)	Bit to indicate object must be redrawn.
TE_HIDE (🔗)	Bit to indicate object must be removed from screen.
TE_UPDATE_KEY (🔗)	Bit to indicate redraw of a key is needed.
TE_UPDATE_TEXT (🔗)	Bit to indicate redraw of the text displayed is needed.

Module

Text Entry (🔗)

Description

List of Text Entry (🔗) bit states.

6.2.1.23.1.1 TE_KEY_PRESSED Macro

File

TextEntry.h

C

```
#define TE_KEY_PRESSED 0x0004 // Bit for press state of one of the keys.
```

Description

Bit for press state of one of the keys.

6.2.1.23.1.2 TE_DISABLED Macro

File

TextEntry.h

C

```
#define TE_DISABLED 0x0002 // Bit for disabled state.
```

Description

Bit for disabled state.

6.2.1.23.1.3 TE_ECHO_HIDE Macro

File

TextEntry.h

C

```
#define TE_ECHO_HIDE 0x0008 // Bit to hide the entered characters and instead echo "*" characters to the display.
```

Description

Bit to hide the entered characters and instead echo "*" characters to the display.

6.2.1.23.1.4 TE_DRAW Macro

File

TextEntry.h

C

```
#define TE_DRAW 0x4000 // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

6.2.1.23.1.5 TE_HIDE Macro

File

TextEntry.h

C

```
#define TE_HIDE 0x8000 // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

6.2.1.23.1.6 TE_UPDATE_KEY Macro

File

TextEntry.h

C

```
#define TE_UPDATE_KEY 0x2000 // Bit to indicate redraw of a key is needed.
```

Description

Bit to indicate redraw of a key is needed.

6.2.1.23.1.7 TE_UPDATE_TEXT Macro

File

TextEntry.h

C

```
#define TE_UPDATE_TEXT 0x1000 // Bit to indicate redraw of the text displayed is needed.
```

Description

Bit to indicate redraw of the text displayed is needed.

6.2.1.23.2 Key Command Types

Macros

Name	Description
TE_DELETE_COM (☒)	This macro is used to assign a "delete" command on a key.
TE_ENTER_COM (☒)	This macro is used to assign an "enter" (carriage return) command on a key.
TE_SPACE_COM (☒)	This macro is used to assign an insert "space" command on a key.

Module

Text Entry (☒)

Description

List of available Key command types.

6.2.1.23.2.1 TE_DELETE_COM Macro**File**

TextEntry.h

C

```
#define TE_DELETE_COM 0x01      // This macro is used to assign a "delete" command on a key.
```

Description

This macro is used to assign a "delete" command on a key.

6.2.1.23.2.2 TE_ENTER_COM Macro**File**

TextEntry.h

C

```
#define TE_ENTER_COM 0x03      // This macro is used to assign an "enter" (carriage return) command on a key.
```

Description

This macro is used to assign an "enter" (carriage return) command on a key.

6.2.1.23.2.3 TE_SPACE_COM Macro**File**

TextEntry.h

C

```
#define TE_SPACE_COM 0x02      // This macro is used to assign an insert "space" command on a key.
```

Description

This macro is used to assign an insert "space" command on a key.

6.2.1.23.3 TeCreate Function**File**

TextEntry.h

C

```
TEXTENTRY * TeCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    SHORT horizontalKeys,
    SHORT verticalKeys,
    XCHAR * pText[],
    void * pBuffer,
    WORD bufferLength,
    void * pDisplayFont,
    GOL_SCHEME * pScheme
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance
SHORT left	Left most position of the object.
SHORT top	Top most position of the object.
SHORT right	Right most position of the object.
SHORT bottom	Bottom most position of the object.
WORD state	state of the widget.
SHORT horizontalKeys	Number of horizontal keys
SHORT verticalKeys	Number of vertical keys
XCHAR * pText[]	array of pointer to the custom "text" assigned by the user.
void * pBuffer	pointer to the buffer that holds the text to be displayed.
WORD bufferLength	length of the buffer assigned by the user.
void * pDisplayFont	pointer to the font image to be used on the editbox
GOL_SCHEME * pScheme	Pointer to the style scheme used. Output Returns the pointer to the object created.

Side Effects

none.

Preconditions

If the object will use customized keys, the structure CUSTOMKEYS must be populated before calling this function.

Overview

This function creates a TEXTENTRY (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

```
TEXTENTRY (█) *TeCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT
horizontalKeys, SHORT verticalKeys, XCHAR (█) *pText[], void *pBuffer, WORD bufferLength, void *pDisplayFont,
GOL_SCHEME (█) *pScheme)
```

6.2.1.23.4 TeDraw Function**File**

TextEntry.h

C

```
WORD TeDraw(
    void * pObj
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
pTe	Pointer to the object to be rendered.

Side Effects

none.

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object.

This widget will draw the keys using the function GOLPanelDraw (§). The number of keys will depend on the horizontal and vertical parameters given (horizontalKeys*verticalKeys).

Syntax

```
WORD TeDraw(void *pObj)
```

6.2.1.23.5 TeGetBuffer Macro

File

TextEntry.h

C

```
#define TeGetBuffer(pTe) (((TEXTENTRY *)pTe)->pTeOutput)
```

Module

Text Entry (§)

Input Parameters

Input Parameters	Description
pTe	pointer to the object

Side Effects

none.

Returns

It will return a pointer to the buffer used.

Preconditions

none

Overview

This macro will return the currently used buffer in the TextEntry object.

Syntax

```
TeGetBuffer(pTe)
```

6.2.1.23.6 TeSetBuffer Function

File

TextEntry.h

C

```
void TeSetBuffer(
    TEXTENTRY * pTe,
    XCHAR * pText,
    WORD MaxSize
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object
XCHAR * pText	pointer to the new text buffer to be displayed
maxSize	maximum length of the new buffer to be used.

Side Effects

none.

Returns

none.

Preconditions

none

Overview

This function sets the buffer used to display text. If the buffer is initialized with a string, the string must be a null terminated string. If the string length is greater than MaxSize, string will be truncated to MaxSize. pText must point to a valid memory location with size equal to MaxSize+1. The +1 is used for the string terminator.

Syntax

```
void TeSetBuffer(TEXTENTRY (█) *pTe, XCHAR (█) *pText, WORD MaxSize)
```

6.2.1.23.7 TeClearBuffer Function

File

TextEntry.h

C

```
void TeClearBuffer(
    TEXTENTRY * pTe
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object

Side Effects

none.

Returns

none

Preconditions

none

Overview

This function will clear the data in the display. You must set the drawing state bit TE_UPDATE_TEXT (■) to update the TEXTENTRY (■) on the screen.

Syntax

```
void TeClearBuffer (TEXTENTRY (■) *pTe)
```

6.2.1.23.8 TeGetKeyCommand Function**File**

TextEntry.h

C

```
WORD TeGetKeyCommand(
    TEXTENTRY * pTe,
    WORD index
);
```

Module

Text Entry (■)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object
WORD index	index to the key in the link list

Side Effects

none.

Returns

It will return the command ID currently set for the key. If the given index is not in the list the function returns zero. 0x00 - no command is assigned or the index given does not exist. 0x01 - TE_DELETE_COM (■) 0x02 - TE_SPACE_COM (■) 0x03 - TE_ENTER_COM (■)

Preconditions

none

Overview

This function will return the currently used command by a key with the given index.

Syntax

```
TeGetKeyCommand(pTe, index)
```

6.2.1.23.9 TeSetKeyCommand Function**File**

TextEntry.h

C

```
BOOL TeSetKeyCommand(
    TEXTENTRY * pTe,
    WORD index,
    WORD command
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object
WORD index	index to the key in the link list
WORD command	command assigned for the key

Side Effects

none.

Returns

Returns TRUE if successful and FALSE if not.

Preconditions

none

Overview

This function will assign a command (TE_DELETE_COM (█), TE_SPACE_COM (█) or TE_ENTER_COM (█)) to a key with the given index.

Syntax

```
void TeSetKeyCommand(TEXTENTRY (█) *pTe,WORD index,WORD command)
```

6.2.1.23.10 TeCreateKeyMembers Function

File

TextEntry.h

C

```
KEYMEMBER * TeCreateKeyMembers (
    TEXTENTRY * pTe,
    XCHAR * pText[]
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object
XCHAR * pText[]	pointer to the text defined by the user

Side Effects

none.

Returns

Returns the pointer to the newly created KEYMEMBER (█) list. A NULL is returned if the list is not created successfully.

Preconditions

none

Overview

This function will create the list of KEYMEMBERS that holds the information on each key. The number of keys is determined by the equation (verticalKeys*horizontalKeys). The object creates the information holder for each key automatically and

assigns each entry in the *pText[] array with the first entry automatically assigned to the key with an index of 1. The number of entries to *pText[] must be equal or greater than (verticalKeys*horizontalKeys). The last key is assigned with an index of (verticalKeys*horizontalKeys)-1. No checking is performed on the length of *pText[] entries to match (verticalKeys*horizontalKeys).

Syntax

```
KEYMEMBER (■) *TeCreateKeyMembers(TEXTENTRY (■) *pTe,XCHAR (■) *pText[])
```

6.2.1.23.11 TeAddChar Function

File

TextEntry.h

C

```
void TeAddChar(
    TEXTENTRY * pTe
);
```

Module

Text Entry (■)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object

Side Effects

none.

Preconditions

none

Overview

This function will insert a character to the end of the buffer. The character inserted is dependent on the currently pressed key. Drawing states TE_UPDATE_TEXT (■) or TE_DRAW (■) must be set to see the effect of this insertion.

Syntax

```
void TeAddChar(TEXTENTRY (■) *pTe)
```

6.2.1.23.12 TeIsKeyPressed Function

File

TextEntry.h

C

```
BOOL TeIsKeyPressed(
    TEXTENTRY * pTe,
    WORD index
);
```

Module

Text Entry (■)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object
WORD index	index to the key in the link list

Side Effects

none.

Returns

Returns a TRUE if the key is pressed. FALSE if key is not pressed or the given index does not exist in the list.

Preconditions

none

Overview

This function will test if a key given by its index in the TextEntry object has been pressed.

Syntax

```
BOOL TelsKeyPressed(TEXTENTRY (■) *pTe, WORD index)
```

6.2.1.23.13 TeSpaceChar Function

File

TextEntry.h

C

```
void TeSpaceChar(
    TEXTENTRY * pTe
);
```

Module

Text Entry (■)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object

Side Effects

none.

Returns

none.

Preconditions

none

Overview

This function will insert a space character to the end of the buffer. Drawing states TE_UPDATE_TEXT (■) or TE_DRAW (■) must be set to see the effect of this insertion.

Syntax

```
void TeSpaceChar(TEXTENTRY (■) *pTe)
```

6.2.1.23.14 TeDelKeyMembers Function

File

TextEntry.h

C

```
void TeDelKeyMembers(
    void * pObj
```

) ;

Module

Text Entry (■)

Input Parameters

Input Parameters	Description
pTe	pointer to the object

Side Effects

none.

Returns

none.

Preconditions

none

Overview

This function will delete the KEYMEMBER (■) list assigned to the object from memory. Pointer to the KEYMEMBER (■) list is then initialized to NULL.

Syntax

```
void TeDelKeyMembers(void *pObj)
```

6.2.1.23.15 TeSetKeyText Function

File

TextEntry.h

C

```
BOOL TeSetKeyText(
    TEXTENTRY * pTe,
    WORD index,
    XCHAR * pText
);
```

Module

Text Entry (■)

Input Parameters

Input Parameters	Description
TEXTENTRY * pTe	pointer to the object
WORD index	index to the key in the link list
XCHAR * pText	pointer to the new string to be used

Side Effects

none.

Returns

Returns TRUE if successful and FALSE if not.

Preconditions

none

Overview

This function will set the test assigned to a key with the given index.

Syntax

```
TeSetKeyText(TEXTENTRY ( ) *pTe, WORD index, XCHAR ( ) *pText)
```

6.2.1.23.16 TeMsgDefault Function**File**

TextEntry.h

C

```
void TeMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
) ;
```

Module

Text Entry ()

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL message.
pTe	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg ()() example.

Overview

This function performs the actual state change based on the translated message given. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
TE_MSG_ADD_CHAR	Touch Screen,	Set TE_UPDATE_TEXT (), TE_UPDATE_KEY (), Clear TE_KEY_PRESSED ()	Add a character in the buffer and update the text displayed.
TE_MSG_SPACE	Touch Screen,	Set TE_UPDATE_TEXT (), TE_UPDATE_KEY (), Clear TE_KEY_PRESSED ()	Insert a space character in the buffer and update the text displayed.
TE_MSG_DELETE	Touch Screen,	Set TE_UPDATE_TEXT (), TE_UPDATE_KEY (), Clear TE_KEY_PRESSED ()	Delete the most recent character in the buffer and update the text displayed.
TE_MSG_ENTER	Touch Screen,	Set TE_UPDATE_TEXT (), TE_UPDATE_KEY (),	User can define the use of this event in the message callback. Object will just update the key.

		Clear TE_KEY_PRESSED (█)	
TE_MSG_RELEASED	Touch Screen,	Clear TE_KEY_PRESSED (█)	A Key in the object will be redrawn in the unpressed state.
		Set Te_UPDATE_KEY	
TE_MSG_PRESSED	Touch Screen,	Set TE_KEY_PRESSED (█) TE_UPDATE_KEY (█)	A Key in the object will be redrawn in the pressed state.

Syntax

```
TeMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (█)* pMsg)
```

6.2.1.23.17 TeTranslateMsg Function**File**

TextEntry.h

C

```
WORD TeTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Text Entry (█)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pTe	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none.

Returns

Returns the translated message depending on the received GOL message:

- TE_MSG_PRESS – A key is pressed
- TE_MSG_RELEASED - A key was released (generic for keys with no commands or characters assigned)
- TE_MSG_ADD_CHAR – A key was released with character assigned
- TE_MSG_DELETE – A key was released with delete command assigned
- TE_MSG_SPACE - A key was released with space command assigned
- TE_MSG_ENTER - A key was released with enter command assigned
- OBJ_MSG_INVALID – Text Entry is not affected

Preconditions

none

Overview

This function evaluates the message from a user if the message will affect the object or not. If the message is valid, the keys in the Text Entry object will be scanned to detect which key was pressed. If True, the corresponding text will be displayed, the 'text' will also be stored in the TeOutput parameter of the object.

Translated Message	Input Source	Events	Description
TE_MSG_PRESS	Touch Screen	EVENT_PRESS, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed.
TE_MSG_RELEASED	Touch Screen	EVENT_MOVE	If the event occurs and the x,y position falls outside the face of one of the keys of the object while the key is pressed.
TE_MSG_RELEASED	Touch Screen	EVENT_RELEASE	If the event occurs and the x,y position falls does not falls inside any of the faces of the keys of the object.
TE_MSG_ADD_CHAR	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with no commands.
TE_MSG_DELETE	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with delete command.
TE_MSG_SPACE	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with space command.
TE_MSG_ENTER	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with enter command.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD TeTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.23.18 TEXTENTRY Structure**File**

TextEntry.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT horizontalKeys;
    SHORT verticalKeys;
    XCHAR * pTeOutput;
    WORD CurrentLength;
    WORD outputLenMax;
    KEYMEMBER * pActiveKey;
    KEYMEMBER * pHeadOfList;
    void * pDisplayFont;
} TEXTENTRY;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all objects (see OBJ_HEADER (■)).
SHORT horizontalKeys;	Number of horizontal keys
SHORT verticalKeys;	Number of vertical keys
XCHAR * pTeOutput;	Pointer to the buffer assigned by the user which holds the text shown in the editbox.
WORD CurrentLength;	Current length of the string in the buffer. The maximum value of this is equal to outputLenMax.
WORD outputLenMax;	Maximum expected length of output buffer pTeOutput
KEYMEMBER * pActiveKey;	Pointer to the active key KEYMEMBER (■). This is only used by the Widget. User must not change

KEYMEMBER * pHedOfList;	Pointer to head of the list
void * pDisplayFont;	Pointer to the font used in displaying the text.

Module[Text Entry \(图\)](#)**Overview**

Defines the parameters required for a TextEntry Object.

6.2.1.23.19 KEYMEMBER Structure**File**

TextEntry.h

C

```
typedef struct {
    SHORT left;
    SHORT top;
    SHORT right;
    SHORT bottom;
    SHORT index;
    WORD state;
    BOOL update;
    WORD command;
    XCHAR * pKeyName;
    SHORT textWidth;
    SHORT textHeight;
    void * pNextKey;
} KEYMEMBER;
```

Members

Members	Description
SHORT left;	Left position of the key
SHORT top;	Top position of the key
SHORT right;	Right position of the key
SHORT bottom;	Bottom position of the key
SHORT index;	Index of the key in the list
WORD state;	State of the key. Either Pressed (TE_KEY_PRESSED (图)) or Released (0)
BOOL update;	flag to indicate key is to be redrawn with the current state
WORD command;	Command of the key. Either TE_DELETE_COM (图), TE_SPACE_COM (图) or TE_ENTER_COM (图).
XCHAR * pKeyName;	Pointer to the custom text assigned to the key. This is displayed over the face of the key.
SHORT textWidth;	Computed text width, done at creation. Used to predict size and position of text on the key face.
SHORT textHeight;	Computed text height, done at creation. Used to predict size and position of text on the key face.
void * pNextKey;	Pointer to the next key parameters.

Module[Text Entry \(图\)](#)**Overview**

Defines the parameters and the strings assigned for each key.

6.2.1.24 Window

Window is an Object that can be used to encapsulate objects into a group. Unlike the Group Box Object, the Window Object has additional features such as displaying an icon or a small bitmap on its Title Bar (█). It also has additional controls for both Title Bar (█) and Client Area.

Functions

	Name	Description
≡	WndCreate (█)	This function creates a WINDOW (█) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
≡	WndDraw (█)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.
≡	WndSetText (█)	This function sets the string used for the title bar.
≡	WndTranslateMsg (█)	This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the translated messages for each event of the touch screen inputs.

Macros

Name	Description
WndGetText (█)	This macro returns the address of the current text string used for the title bar.

Structures

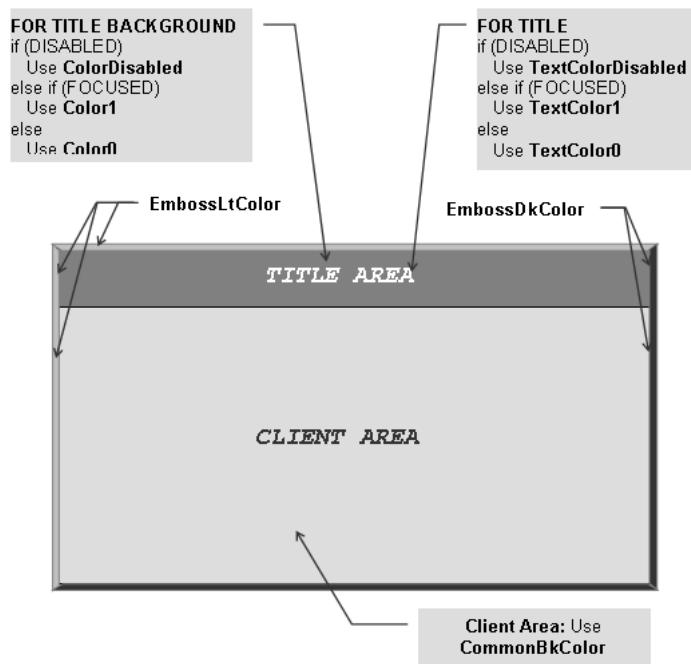
Name	Description
WINDOW (█)	The structure contains data for the window

Description

Window supports only Touchscreen inputs, replying to their events with the following messages:

1. WND_MSG_TITLE – Title area is selected.
2. WND_MSG_CLIENT – Client area is selected.

The Window Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



CommonBkColor – used to hide the window from the screen.

6.2.1.24.1 Window States

Macros

Name	Description
WND_DISABLED (█)	Bit for disabled state
WND_DRAW (█)	Bits to indicate whole window must be redrawn
WND_DRAW_CLIENT (█)	Bit to indicate client area must be redrawn
WND_DRAW_TITLE (█)	Bit to indicate title area must be redrawn
WND_FOCUSED (█)	Bit for focus state
WND_HIDE (█)	Bit to indicate window must be removed from screen
WND_TITLECENTER (█)	Bit to center the text on the Title Area

Module

Window (█)

Description

List of Window (█) bit states.

6.2.1.24.1.1 WND_DISABLED Macro

File

Window.h

C

```
#define WND_DISABLED 0x0002 // Bit for disabled state
```

Description

Bit for disabled state

6.2.1.24.1.2 WND_DRAW Macro

File

Window.h

C

```
#define WND_DRAW 0x6000 // Bits to indicate whole window must be redrawn
```

Description

Bits to indicate whole window must be redrawn

6.2.1.24.1.3 WND_DRAW_CLIENT Macro

File

Window.h

C

```
#define WND_DRAW_CLIENT 0x4000 // Bit to indicate client area must be redrawn
```

Description

Bit to indicate client area must be redrawn

6.2.1.24.1.4 WND_DRAW_TITLE Macro

File

Window.h

C

```
#define WND_DRAW_TITLE 0x2000 // Bit to indicate title area must be redrawn
```

Description

Bit to indicate title area must be redrawn

6.2.1.24.1.5 WND_FOCUSED Macro

File

Window.h

C

```
#define WND_FOCUSED 0x0001 // Bit for focus state
```

Description

Bit for focus state

6.2.1.24.1.6 WND_HIDE Macro

File

Window.h

C

```
#define WND_HIDE 0x8000 // Bit to indicate window must be removed from screen
```

Description

Bit to indicate window must be removed from screen

6.2.1.24.1.7 WND_TITLECENTER Macro

File

Window.h

C

```
#define WND_TITLECENTER 0x0004 // Bit to center the text on the Title Area
```

Description

Bit to center the text on the Title Area

6.2.1.24.2 WndCreate Function

File

Window.h

C

```
WINDOW * WndCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    void * pBitmap,
    XCHAR * pText,
    GOL_SCHEME * pScheme
);
```

Module

Window (■)

Input Parameters

Input Parameters	Description
WORD ID	Unique user defined ID for the object instance.
SHORT left	Left most position of the Object.
SHORT top	Top most position of the Object.
SHORT right	Right most position of the Object.
SHORT bottom	Bottom most position of the object.
WORD state	Sets the initial state of the object.
void * pBitmap	Pointer to the bitmap used in the title bar.
XCHAR * pText	Pointer to the text used as a title of the window.
GOL_SCHEME * pScheme	Pointer to the style scheme used.

Side Effects

none

Returns

Returns the pointer to the object created

Preconditions

none

Example

```
WINDOW *pWindow;
pWindow = WndCreate(ID_WINDOW1, // ID
                    0,0,GetMaxX(),GetMaxY(), // whole screen dimension
                    WND_DRAW, // set state to draw all
```

```

(char*)myIcon,           // icon
"Place Title Here.",   // text
NULL);                 // use default GOL scheme

if (pWindow == NULL)
    return 0;
WndDraw(pWindow);
return 1;

```

Overview

This function creates a WINDOW ([\[?\]](#)) object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

Syntax

WINDOW ([\[?\]](#)) *WndCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, void* pBitmap, XCHAR ([\[?\]](#))* pText, GOL_SCHEME ([\[?\]](#)) *pScheme)

6.2.1.24.3 WndDraw Function**File**

Window.h

C

```

WORD WndDraw(
    void * pObj
);

```

Module

Window ([\[?\]](#))

Input Parameters

Input Parameters	Description
pW	Pointer to the object to be rendered.

Side Effects

none

Returns

Returns the status of the drawing

- 1 - If the rendering was completed and
- 0 - If the rendering is not yet finished.

Next call to the function will resume the rendering on the pending drawing state.

Preconditions

Object must be created before this function is called.

Example

See WndCreate ([\[?\]](#))() example.

Overview

This function renders the object on the screen using the current parameter settings. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Syntax

```
WORD WndDraw(void *pObj)
```

6.2.1.24.4 WndGetText Macro**File**

Window.h

C

```
#define WndGetText(pW) pW->pText
```

Module

Window ([🔗](#))

Input Parameters

Input Parameters	Description
pW	Pointer to the object

Side Effects

none

Returns

Returns pointer to the text string being used.

Preconditions

none

Example

```
WINDOW *pWindow;
XCHAR textUsed = "USE THIS!";

if (WndGetText(pWindow) == NULL)
    WndSetText(&textUsed);
```

Overview

This macro returns the address of the current text string used for the title bar.

Syntax

```
WndGetText(pW)
```

6.2.1.24.5 WndSetText Function**File**

Window.h

C

```
void WndSetText(
    WINDOW * pW,
    XCHAR * pText
);
```

Module

Window ([🔗](#))

Input Parameters

Input Parameters	Description
WINDOW * pW	The pointer to the object whose text will be modified

XCHAR * pText	Pointer to the text that will be used
---------------	---------------------------------------

Side Effects

none

Returns

none

Preconditions

none

Example

See WndGetText (🔗)() example.

Overview

This function sets the string used for the title bar.

Syntax

```
WndSetText(WINDOW (🔗) *pW, XCHAR (🔗) *pText)
```

6.2.1.24.6 WndTranslateMsg Function

File

Window.h

C

```
WORD WndTranslateMsg(
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Window (🔗)

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the message struct containing the message from the user interface.
pW	The pointer to the object where the message will be evaluated to check if the message will affect the object.

Side Effects

none

Returns

Returns the translated message depending on the received GOL message:

- WND_MSG_TITLE – Title area is selected
- WND_MSG_CLIENT – Client area is selected
- OBJ_MSG_INVALID – Window (🔗) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (🔗)() example.

Overview

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the

translated messages for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
WND_MSG_TITLE	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the TITLE area of the window
WND_MSG_CLIENT	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the CLIENT area of the window
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

```
WORD WndTranslateMsg(void *pObj, GOL_MSG (■) *pMsg)
```

6.2.1.24.7 WINDOW Structure

File

Window.h

C

```
typedef struct {
    OBJ_HEADER hdr;
    SHORT textHeight;
    XCHAR * pText;
    void * pBitmap;
} WINDOW;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (■)).
SHORT textHeight;	Pre-computed text height
XCHAR * pText;	Pointer to the title text
void * pBitmap;	Pointer to the bitmap for the title bar

Module

Window (■)

Overview

The structure contains data for the window

6.2.2 Object States

Objects rendered on the display are based on their current Property States and the Drawing States.

Macros

Name	Description
GetState (■)	This macro retrieves the current value of the state bits of an object. It is possible to get several state bits.
ClrState (■)	This macro clear the state bits of an object. Object must be redrawn to display the changes. It is possible to clear several state bits with this macro.

SetState (█)	This macro sets the state bits of an object. Object must be redrawn to display the changes. It is possible to set several state bits with this macro.
--------------	---

Description

The GOL objects follow two types of states, the Property States and the Drawing States. Property States defines action and appearance of objects. Drawing States on the other hand indicate if the object needs to be hidden, partially redrawn, or fully redrawn in the display. To store the states, the field state defined in OBJ_HEADER (█) structure is used. Six most significant bits are allocated for Drawing States and the rest is allocated for the Property States. Some common Property States and Drawing States are shown in the following table.

State	Type	Bit Location	Description
OBJ_FOCUSED	P	0x0001	Object is in the focused state. This is usually used to show selection of the object. Not all objects have this feature.
OBJ_DISABLED	P	0x0002	Object is disabled and will ignore all messages.
OBJ_DRAW_FOCUS	D	0x2000	Focus for the object should be redrawn.
OBJ_DRAW	D	0x4000	Object should be redrawn completely.
OBJ_HIDE	D	0x8000	Object will be hidden by filling the area occupied by the object with the common background color. This has the highest priority over all Drawing States. When an object is set to be hidden, all other drawing states are overridden.

Where:

- OBJ – represent the prefix assigned to a GOL object.
- P – Property states, D – Drawing states

Individual Object drawing function (e.g. BtnDraw (█)(), SldDraw (█)(), etc...) does not reset the draw states instead use GOLDraw (█)() to automatically reset and manage the draw states. If the call to individual drawing function cannot be avoided, draw states must be reset manually after the drawing functions returns a 1.

6.2.2.1 Common Object States

Macros

Name	Description
FOCUSSED (█)	Focus state bit.
DISABLED (█)	Disabled state bit.
HIDE (█)	Object hide state bit. Object will be hidden from the screen by drawing over it the common background color.
DRAW (█)	Object redraw state bit. The whole Object must be redrawn.
DRAW_FOCUS (█)	Focus redraw state bit. The focus rectangle must be redrawn.
DRAW_UPDATE (█)	Partial Object redraw state bit. A part or parts of the Object must be redrawn to show updated state.

Description

List of common Object bit states.

6.2.2.1.1 FOCUSED Macro

File

GOL.h

C

```
#define FOCUSED 0x0001
```

Description

Focus state bit.

6.2.2.1.2 DISABLED Macro

File

GOL.h

C

```
#define DISABLED 0x0002
```

Description

Disabled state bit.

6.2.2.1.3 HIDE Macro

File

GOL.h

C

```
#define HIDE 0x8000
```

Description

Object hide state bit. Object will be hidden from the screen by drawing over it the common background color.

6.2.2.1.4 DRAW Macro

File

GOL.h

C

```
#define DRAW 0x7C00
```

Description

Object redraw state bit. The whole Object must be redrawn.

6.2.2.1.5 DRAW_FOCUS Macro

File

GOL.h

C

```
#define DRAW_FOCUS 0x2000
```

Description

Focus redraw state bit. The focus rectangle must be redrawn.

6.2.2.1.6 DRAW_UPDATE Macro

File

GOL.h

C

```
#define DRAW_UPDATE 0x3C00
```

Description

Partial Object redraw state bit. A part or parts of the Object must be redrawn to show updated state.

6.2.2.2 GetState Macro

File

GOL.h

C

```
#define GetState(pObj, stateBits) (((OBJ_HEADER *)pObj)->state & (stateBits))
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.
stateBits	Defines which state bits are requested. Please refer to specific objects for object state bits definition for details

Side Effects

none

Returns

Macro returns a non-zero if any bit in the stateBits mask is set.

Preconditions

none

Example

```
#define BTN_HIDE 0x8000
BUTTON *pB;
// pB is created and initialized
// do something here to set state

// Hide the button (remove from screen)
if (GetState(pB,BTN_HIDE)) {
    SetColor(pB->pGolScheme->CommonBkColor);
    Bar(pB->left,pB->top,pB->right,pB->bottom);

}
```

Overview

This macro retrieves the current value of the state bits of an object. It is possible to get several state bits.

Syntax

GetState(pObj, stateBits)

6.2.2.3 ClrState Macro

File

GOL.h

C

```
#define ClrState(pObj, stateBits) ((OBJ_HEADER *)pObj)->state &= (~(stateBits))
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.
stateBits	Defines which state bits are to be cleared. Please refer to specific objects for object state bits definition for details

Side Effects

none

Returns

none

Preconditions

none

Example

See example for SetState (■)().

Overview

This macro clear the state bits of an object. Object must be redrawn to display the changes. It is possible to clear several state bits with this macro.

Syntax

ClrState(pObj, stateBits)

6.2.2.4 SetState Macro

File

GOL.h

C

```
#define SetState(pObj, stateBits) ((OBJ_HEADER *)pObj)->state |= (stateBits)
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.
stateBits	Defines which state bits are to be set. Please refer to specific objects for object state bits definition for details

Side Effects

none

Returns

none

Preconditions

none

Example

```
void BtnMsgDefault(WORD msg, BUTTON* pB){
    switch(msg){
        case BTN_MSG_PRESSED:
            // set pressed and redraw
            SetState(pB, BTN_PRESSED|BTN_DRAW);
            break;
        case BTN_MSG_RELEASED:
            ClrState(pB, BTN_PRESSED);           // reset pressed
    }
}
```

```

        SetState(pB, BTN_DRAW);           // redraw
        break;
    }
}

```

Overview

This macro sets the state bits of an object. Object must be redrawn to display the changes. It is possible to set several state bits with this macro.

Syntax

```
SetState(pObj, stateBits)
```

6.2.3 Object Management

Functions

	Name	Description
💡	GOLAddObject (🔗)	This function adds an object to the tail of the active list pointed to by _pGolObjects (🔗). The new list tail is set to point to NULL.
💡	GOLFFindObject (🔗)	This function finds an object in the active list pointed to by _pGolObjects (🔗) using the given object ID.
💡	GOLRedrawRec (🔗)	This function marks all objects in the active list intersected by the given rectangular area to be redrawn.
💡	GOLDraw (🔗)	This function loops through the active list and redraws objects that need to be redrawn. Partial redrawing or full redraw is performed depending on the drawing states of the objects. GOLDrawCallback (🔗)() function is called by GOLDraw() when drawing of objects in the active list is completed.
💡	GOLDrawCallback (🔗)	GOLDrawCallback() function MUST BE implemented by the user. This is called inside the GOLDraw (🔗)() function when the drawing of objects in the active list is completed. User drawing must be done here. Drawing color, line type, clipping region, graphic cursor position and current font will not be changed by GOL if this function returns a zero. To pass drawing control to GOL this function must return a non-zero value. If GOL messaging is not using the active link list, it is safe to modify the list here.
💡	GOLFree (🔗)	This function frees all the memory used by objects in the active list and initializes _pGolObjects (🔗) pointer to NULL to start a new empty list. This function must be called only inside the GOLDrawCallback (🔗)() function when using GOLDraw (🔗)() and GOLMsg (🔗)() functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.
💡	GOLDeleteObject (🔗)	Deletes an object to the linked list objects for the current screen.
💡	GOLDeleteObjectByID (🔗)	Deletes an object in the current active linked list of objects using the ID parameter of the object.
💡	GOLSetFocus (🔗)	This function sets the keyboard input focus to the object. If the object cannot accept keyboard messages focus will not be changed. This function resets FOCUSED (🔗) state for the object was in focus previously, set FOCUSED (🔗) state for the required object and marks the objects to be redrawn.
💡	GOLInit (🔗)	This function initializes the graphics library and creates a default style scheme with default settings referenced by the global scheme pointer. GOLInit() function must be called before GOL functions can be used. It is not necessary to call GraphInit() function if this function is used.
💡	GOLCanBeFocused (🔗)	This function returns non-zero if the object can be focused. Only button, check box, radio button, slider, edit box, list box, scroll bar can accept focus. If the object is disabled it cannot be set to focused state.
💡	GOLGetFocusNext (🔗)	This function returns the pointer to the next object in the active linked list which is able to receive keyboard input.

	GOLGetFocusPrev (🔗)	This function returns the pointer to the previous object in the active linked list which is able to receive keyboard input.
	GOLPanelDrawTsk (🔗)	This function draws a panel on the screen with parameters set by GOLPanelDraw (🔗 ()) macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.
	GOLTTwoTonePanelDrawTsk (🔗)	This function draws a two tone panel on the screen with parameters set by GOLPanelDraw (🔗 ()) macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Macros

Name	Description
GOLRedraw (🔗)	This macro sets the object to be redrawn. For the redraw to be effective, the object must be in the current active list. If not, the redraw action will not be performed until the list where the object is currently inserted will be set to be the active list.
GOLDrawComplete (🔗)	This macro resets the drawing states of the object (6 MSBits of the object's state).
GetObjType (🔗)	This macro returns the object type.
GetObjID (🔗)	This macro returns the object ID.
GetObjNext (🔗)	This macro returns the next object after the specified object.
GOLNewList (🔗)	This macro starts a new linked list of objects and resets the keyboard focus to none. This macro assigns the current active list _pGolObjects (🔗) and current receiving keyboard input _pObjectFocused (🔗) object pointers to NULL. Any keyboard inputs at this point will be ignored. Previous active list must be saved in another pointer if to be referenced later. If not needed anymore memory used by that list should be freed by GOLFree (🔗 ()) function.
GOLGetList (🔗)	This macro gets the current active list.
GOLSetList (🔗)	This macro sets the given object list as the active list and resets the keyboard focus to none. This macro assigns the receiving keyboard input object _pObjectFocused (🔗) pointer to NULL. If the new active list has an object's state set to focus, the _pObjectFocused (🔗) pointer must be set to this object or the object's state must be change to unfocused. This is to avoid two objects displaying a focused state when only one object in the active list must be set to a focused state at anytime.
IsObjUpdated (🔗)	This macro tests if the object is pending to be redrawn. This is done by testing the 6 MSBits of object's state to detect if the object must be redrawn.
GOLGetFocus (🔗)	This macro returns the pointer to the current object receiving keyboard input.
GOLPanelDraw (🔗)	This is macro GOLPanelDraw.

Description

This section describes the API functions and macros that are used to create, maintain and render individual and list of objects.

6.2.3.1 GOLAddObject Function

File

GOL.h

C

```
void GOLAddObject(
    OBJ_HEADER * object
);
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object to be added on the current active list.

Side Effects

none

Returns

none

Preconditions

none

Example

```
void MoveObject(OBJ_HEADER *pSrcList, OBJ_HEADER *pDstList,
    OBJ_HEADER *pObjtoMove) {
    OBJ_HEADER *pTemp = pSrcList;

    if(pTemp != pObjtoMove) {
        while(pTemp->pNxtObj != pObjtoMove)
            pTemp = pTemp->pNxtObj;
    }

    pTemp->pNxtObj = pObjtoMove ->pNxt; // remove object from list
    GOLSetList(pDstList); // destination as active list
    GOLAddObject(pObjtoMove); // add object to active list
}
```

Overview

This function adds an object to the tail of the active list pointed to by `_pGolObjects` (■). The new list tail is set to point to NULL.

Syntax

```
void GOLAddObject(OBJ_HEADER (■)* object)
```

6.2.3.2 GOLFFindObject Function

File

GOL.h

C

```
OBJ_HEADER * GOLFFindObject(
    WORD ID
);
```

Input Parameters

Input Parameters	Description
WORD ID	User assigned value set during the creation of the object.

Side Effects

none

Returns

Pointer to the object with the given ID.

Preconditions

none

Example

```
void CopyObject(OBJ_HEADER *pSrcList, OBJ_HEADER *pDstList, WORD ID)
{
    OBJ_HEADER *pTemp;

    pTemp = GOLFFindObject(ID); // find the object
```

```

if (pTemp != NULL) {
    GOLSetList(pDstList);           // destination as active list
    GOLAddObject(pObj);            // add object to active list
}
}

```

Overview

This function finds an object in the active list pointed to by `_pGolObjects` (■) using the given object ID.

Syntax

`OBJ_HEADER (*) GOLFFindObject(WORD ID)`

6.2.3.3 GOLRedraw Macro

File

`GOL.h`

C

```
#define GOLRedraw(pObj) ((OBJ_HEADER *)pObj)->state |= 0x7c00;
```

Input Parameters

Input Parameters	Description
<code>pObj</code>	Pointer to the object to be redrawn.

Side Effects

none

Returns

none

Preconditions

none

Example

```

void GOLRedrawRec(SHORT left, SHORT top, SHORT right, SHORT bottom) {
    // set all objects encompassed by the rectangle to be redrawn
    OBJ_HEADER *pCurrentObj;

    pCurrentObj = GOLGetList();
    while(pCurrentObj != NULL){
        if (
            ((pCurrentObj->left >= left) && (pCurrentObj->left <= right)) ||
            ((pCurrentObj->right >= left) && (pCurrentObj->right <= right)) ||
            ((pCurrentObj->top >= top) && (pCurrentObj->top <= bottom)) ||
            ((pCurrentObj->bottom >= top) && (pCurrentObj->bottom <= bottom)))
            GOLRedraw(pCurrentObj);
    }
    pCurrentObj = pCurrentObj->pNxtObj;
} //end of while
}

```

Overview

This macro sets the object to be redrawn. For the redraw to be effective, the object must be in the current active list. If not, the redraw action will not be performed until the list where the object is currently inserted will be set to be the active list.

Syntax

`GOLRedraw(pObj)`

6.2.3.4 GOLRedrawRec Function

File

GOL.h

C

```
void GOLRedrawRec(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom
);
```

Input Parameters

Input Parameters	Description
SHORT left	Defines the left most border of the rectangle area.
SHORT top	Defines the top most border of the rectangle area.
SHORT right	Defines the right most border of the rectangle area.
SHORT bottom	Defines the bottom most border of the rectangle area.

Side Effects

none

Returns

none

Preconditions

none

Example

```
OBJ_HEADER *pTemp;
OBJ_HEADER *pAllObjects;

// assume *pAllObjects points to a list of all existing objects
// created and initialized

// mark all objects inside the rectangle to be redrawn
GOLRedrawRec(10,10,100,100);

pTemp = pAllObjects;
GOLStartNewList();                                // reset active list
while(pTemp->pNxtObj != NULL) {
    if (pTemp->state&0x7C00)                      // add only objects to be
        GOLAddObject(pTemp);                         // redrawn to the active list
    pTemp = pTemp->pNxtObj;
}
GOLDraw();                                         // redraw active list
```

Overview

This function marks all objects in the active list intersected by the given rectangular area to be redrawn.

Syntax

```
void GOLRedrawRec(SHORT left, SHORT top, SHORT right, SHORT bottom)
```

6.2.3.5 GOLDraw Function

File

GOL.h

C

```
WORD GOLDraw( );
```

Side Effects

none

Returns

Non-zero if the active link list drawing is completed.

Preconditions

none

Example

```
// Assume objects are created & states are set to draw objects
while(1){
    if(GOLDraw()){           // parse active list and redraw objects that needs to be
    redrawn

        // here GOL drawing is completed
        // it is safe to modify objects states and linked list

        TouchGetMsg(&msg);      // evaluate messages from touch screen device

        GOLMsg(&msg);          // evaluate each object is affected by the message
    }
}
```

Overview

This function loops through the active list and redraws objects that need to be redrawn. Partial redrawing or full redraw is performed depending on the drawing states of the objects. GOLDrawCallback (画)() function is called by GOLDraw() when drawing of objects in the active list is completed.

Syntax

```
WORD GOLDraw()
```

6.2.3.6 GOLDrawComplete Macro

File

GOL.h

C

```
#define GOLDrawComplete(pObj) ((OBJ_HEADER *)pObj)->state &= 0x03ff
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.

Side Effects

none

Returns

none

Preconditions

none

Example

```
// This function should be called again whenever an incomplete
// rendering (done = 0) of an object occurs.
```

```

// internal states in the BtnDraw() or WndDraw() should pickup
// on the state where it left off to continue rendering.
void GOLDraw() {
    static OBJ_HEADER *pCurrentObj = NULL;
    SHORT done;

    if(pCurrentObj == NULL) {
        if(GOLDrawCallback()) {
            // If it's last object jump to head
            pCurrentObj = GOLGetList();
        } else {
            return;
        }
    }
    done = 0;

    while(pCurrentObj != NULL) {
        if(IsObjUpdated(pCurrentObj)) {
            done = pCurrentObj->draw(pCurrentObj);

            if(done){
                GOLDrawComplete(pCurrentObj);
            } else{
                return;
            }
        }
        pCurrentObj = pCurrentObj->pNxtObj;
    }
}

```

Overview

This macro resets the drawing states of the object (6 MSBits of the object's state).

Syntax

```
GOLDrawComplete(pObj)
```

6.2.3.7 GOLDrawCallback Function

File

GOL.h

C

```
WORD GOLDrawCallback();
```

Side Effects

none

Returns

Return a one if GOLDraw (■) will have drawing control on the active list. Return a zero if user wants to keep the drawing control.

Preconditions

none

Example

```

#define SIG_STATE_SET    0
#define SIG_STATE_DRAW   1
WORD GOLDrawCallback(){
    static BYTE state = SIG_STATE_SET;
    if(state == SIG_STATE_SET){
        // Draw the button with disabled colors
        GOLPanelDraw(SIG_PANEL_LEFT,SIG_PANEL_TOP,
                     SIG_PANEL_RIGHT,SIG_PANEL_BOTTOM, 0,
                     WHITE, altScheme->EmbossLtColor,

```

```

        altScheme->EmbossDkColor,
        NULL, GOL_EMBOSS_SIZE);

    state = SIG_STATE_DRAW;
}

if( !GOLPanelDrawTsk() ){
    // do not return drawing control to GOL
    // drawing is not complete
    return 0;
}else{
    state = SIG_STATE_SET;
    // return drawing control to GOL, drawing is complete
    return 1;
}
}
}

```

Overview

GOLDrawCallback() function MUST BE implemented by the user. This is called inside the GOLDraw (■)() function when the drawing of objects in the active list is completed. User drawing must be done here. Drawing color, line type, clipping region, graphic cursor position and current font will not be changed by GOL if this function returns a zero. To pass drawing control to GOL this function must return a non-zero value. If GOL messaging is not using the active link list, it is safe to modify the list here.

Syntax

WORD GOLDrawCallback()

6.2.3.8 GOLFfree Function

File

GOL.h

C

```
void GOLFfree( );
```

Side Effects

All objects in the active list are deleted from memory.

Returns

none

Preconditions

none

Example

```

void DeletePage(OBJ_HEADER *pPage) {
    OBJ_HEADER *pTemp;

    // assuming pPage is different from the current active list
    pTemp = GOLGetList();           // save the active list
    GOLSetList(pPage);             // set list as active list
    GOLFfree();                   // pPage objects are deleted

    GOLSetList(pTemp);            // restore the active list
}

```

Overview

This function frees all the memory used by objects in the active list and initializes _pGolObjects (■) pointer to NULL to start a new empty list. This function must be called only inside the GOLDrawCallback (■)()function when using GOLDraw (■)() and GOLMsg (■)() functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.

Syntax

```
void GOLFree()
```

6.2.3.9 GetObjType Macro**File**

GOL.h

C

```
#define GetObjType(pObj) ((OBJ_HEADER *)pObj)->type
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.

Side Effects

none

Returns

Returns the OBJ_TYPE of the object.

Preconditions

none

Overview

This macro returns the object type.

Syntax

```
GetObjType(pObj)
```

6.2.3.10 GetObjID Macro**File**

GOL.h

C

```
#define GetObjID(pObj) ((OBJ_HEADER *)pObj)->ID
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.

Side Effects

none

Returns

Returns the ID of the object.

Preconditions

none

Example

```
void UseOfGetObjID(OBJ_HEADER *pObj) {
    WORD id;
    switch(id = GetObjID(pObj)) {
```

```

        case ID_WINDOW1:
            // do something
        case ID_WINDOW2:
            // do something else
        case ID_WINDOW3:
            // do something else
        default:
            // do something else
    }
}

```

Overview

This macro returns the object ID.

Syntax

GetObjID(pObj)

6.2.3.11 GetObjNext Macro

File

GOL.h

C

```
#define GetObjNext(pObj) ((OBJ_HEADER *)pObj)->pNxtObj
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.

Side Effects

none

Returns

Returns the pointer of the next object.

Preconditions

none

Example

```

// This is the same example for the GetObjType() macro
// We just replaced one line
void RedrawButtons(void) {
    OBJ_HEADER *pCurr;

    pCurr = GOLGetList();           // get active list
    while(pCurr->pNxtObj != NULL) {
        if (GetObjType(pCurr) == BUTTON)
            pCurr->state = BTN_DRAW; // set button to be redrawn
        pCurr = GetObjNext(pCurr);   // Use of GetObjNext() macro
        // replaces the old line
    }
    GolDraw();                     // redraw all buttons in the
                                    // active list
}

```

Overview

This macro returns the next object after the specified object.

Syntax

GetObjNext(pObj)

6.2.3.12 GOLDeleteObject Function

File

GOL.h

C

```
BOOL GOLDeleteObject(
    OBJ_HEADER * object
);
```

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

Deletes an object to the linked list objects for the current screen.

Syntax

```
BOOL GOLDeleteObject(OBJ_HEADER (■) * object)
```

6.2.3.13 GOLDeleteObjectByID Function

File

GOL.h

C

```
BOOL GOLDeleteObjectByID(
    WORD ID
);
```

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

Deletes an object in the current active linked list of objects using the ID parameter of the object.

Syntax

```
BOOL GOLDeleteObjectByID(WORD ID)
```

6.2.3.14 GOLNewList Macro

File

GOL.h

C

```
#define GOLNewList \
    _pGolObjects = NULL; \
    _pObjectFocused = NULL
```

Side Effects

This macro sets the focused object pointer (`_pObjectFocused`) to NULL.

Returns

none

Preconditions

none

Example

```
OBJ_HEADER *pSave;

pSave = GOLGetList();           // save current list
GOLNewList();                  // start the new list
                                // current list is now NULL

// assume that objects are already created
// you can now add objects to the new list
GOLAddObject(pButton);
GOLAddObject(pWindow);
GOLAddObject(pSlider);
```

Overview

This macro starts a new linked list of objects and resets the keyboard focus to none. This macro assigns the current active list `_pGolObjects` (■) and current receiving keyboard input `_pObjectFocused` (■) object pointers to NULL. Any keyboard inputs at this point will be ignored. Previous active list must be saved in another pointer if to be referenced later. If not needed anymore memory used by that list should be freed by `GOLFRelease()` function.

Syntax

```
void GOLNewList()
```

6.2.3.15 GOLGetList Macro

File

GOL.h

C

```
#define GOLGetList _pGolObjects
```

Side Effects

none

Returns

Returns the pointer to the current active list.

Preconditions

none

Example

See GOLNewList (🔗)() example.

Overview

This macro gets the current active list.

Syntax

GOLGetList()

6.2.3.16 GOLSetList Macro

File

GOL.h

C

```
#define GOLSetList(objsList) \
    _pGolObjects = objsList; \
    _pObjectFocused = NULL;
```

Input Parameters

Input Parameters	Description
objsList	The pointer to the new active list.

Side Effects

This macro sets the focused object pointer (_pObjectFocused (🔗)) to NULL. Previous active list should be saved if needed to be referenced later. If not, use GOLFfree (🔗)() function to free the memory used by the objects before calling GOLSetList().

Returns

none

Preconditions

none

Example

```
OBJ_HEADER *pSave;
pSave = GOLGetList();           // save current list
GOLNewList();                  // start the new list
                                // current list is now NULL

// you can now add objects to the current list
// assume that objects are already created
GOLAddObject(pButton);
GOLAddObject(pWindow);
GOLAddObject(pSlider);

// do something here on the new list
// return the old list
GOLSetList(pSave);
```

Overview

This macro sets the given object list as the active list and resets the keyboard focus to none. This macro assigns the receiving keyboard input object _pObjectFocused (🔗) pointer to NULL. If the new active list has an object's state set to focus, the _pObjectFocused (🔗) pointer must be set to this object or the object's state must be change to unfocused. This is to avoid two objects displaying a focused state when only one object in the active list must be set to a focused state at anytime.

Syntax

GOLSetList(objsList)

6.2.3.17 GOLSetFocus Function

File

GOL.h

C

```
void GOLSetFocus(
    OBJ_HEADER * object
);
```

Input Parameters

Input Parameters	Description
OBJ_HEADER * object	Pointer to the object of interest.

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the keyboard input focus to the object. If the object cannot accept keyboard messages focus will not be changed. This function resets FOCUSED (■) state for the object was in focus previously, set FOCUSED (■) state for the required object and marks the objects to be redrawn.

Syntax

```
void GOLSetFocus(OBJ_HEADER (■)* object)
```

6.2.3.18 IsObjUpdated Macro

File

GOL.h

C

```
#define IsObjUpdated(pObj) (((OBJ_HEADER *)pObj)->state & 0xfc00)
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.

Side Effects

none

Returns

Returns a nonzero value if the object needs to be redrawn. Zero if not.

Preconditions

none

Example

```
int DrawButtonWindowOnly() {
    static OBJ_HEADER *pCurrentObj = NULL;
```

```

SHORT done = 0;

if (pCurrentObj == NULL)
    pCurrentObj = GOLGetList();           // get current list

while(pCurrentObj != NULL){
    if(IsObjUpdated(pCurrentObj)){
        done = pCurrentObj->draw(pCurrentObj);

        // reset state of object if done
        if (done)
            GOLDrawComplete(pCurrentObj)
        // Return if not done. This means that BtnDraw()
        // was terminated prematurely by device busy status
        // and must be recalled to finish rendering of
        // objects in the list that have new states.
        else
            return 0;
    }
    // go to the next object in the list
    pCurrentObj = pCurrentObj->pNxtObj;
}
return 1;
}

```

Overview

This macro tests if the object is pending to be redrawn. This is done by testing the 6 MSBits of object's state to detect if the object must be redrawn.

Syntax

```
IsObjUpdated(pObj)
```

6.2.3.19 GOLInit Function

File

GOL.h

C

```
void GOLInit();
```

Side Effects

This sets the line type to SOLID_LINE (█), sets the screen to all BLACK (█), sets the current drawing color to WHITE (█), sets the graphic cursor position to upper left corner of the screen, sets active and visual pages to page #0, clears the active page and disables clipping. This also creates a default style scheme.

Returns

none

Preconditions

none

Overview

This function initializes the graphics library and creates a default style scheme with default settings referenced by the global scheme pointer. GOLInit() function must be called before GOL functions can be used. It is not necessary to call GraphInit() function if this function is used.

Syntax

```
void GOLInit()
```

6.2.3.20 GOLGetFocus Macro

File

GOL.h

C

```
#define GOLGetFocus _pObjectFocused
```

Side Effects

none

Returns

Returns the pointer to the object receiving keyboard input. If there is no object in focus NULL is returned.

Preconditions

none

Overview

This macro returns the pointer to the current object receiving keyboard input.

Syntax

```
GOLGetFocus()
```

6.2.3.21 GOLCanBeFocused Function

File

GOL.h

C

```
WORD GOLCanBeFocused(
    OBJ_HEADER * object
);
```

Input Parameters

Input Parameters	Description
OBJ_HEADER * object	Pointer to the object of interest.

Side Effects

none

Returns

This returns a non-zero if the object can be focused and zero if not.

Preconditions

none

Overview

This function returns non-zero if the object can be focused. Only button, check box, radio button, slider, edit box, list box, scroll bar can accept focus. If the object is disabled it cannot be set to focused state.

Syntax

```
WORD GOLCanBeFocused(OBJ_HEADER (■)* object)
```

6.2.3.22 GOLGetFocusNext Function

File

GOL.h

C

```
OBJ_HEADER * GOLGetFocusNext();
```

Side Effects

none

Returns

This returns the pointer of the next object in the active list capable of receiving keyboard input. If there is no object capable of receiving keyboard inputs (i.e. none can be focused) NULL is returned.

Preconditions

none

Overview

This function returns the pointer to the next object in the active linked list which is able to receive keyboard input.

Syntax

```
OBJ_HEADER (¶) *GOLGetFocusNext()
```

6.2.3.23 GOLGetFocusPrev Function

File

GOL.h

C

```
OBJ_HEADER * GOLGetFocusPrev();
```

Side Effects

none

Returns

This returns the pointer of the previous object in the active list capable of receiving keyboard input. If there is no object capable of receiving keyboard inputs (i.e. none can be focused) NULL is returned.

Preconditions

none

Overview

This function returns the pointer to the previous object in the active linked list which is able to receive keyboard input.

Syntax

```
OBJ_HEADER (¶) *GOLGetFocusPrev (¶)()
```

6.2.3.24 GOLPanelDraw Macro

File

GOL.h

C

```
#define GOLPanelDraw(left, top, right, bottom, radius, faceClr, embossLtClr, embossDkClr,
pBitmap, embossSize) \
    _rpn1X1 = \
    left; \
    _rpn1Y1 = \
    top; \
    _rpn1X2 = \
    right; \
    _rpn1Y2 = \
    bottom; \
    _rpn1R = \
    radius; \
    _rpn1FaceColor = \
    faceClr; \
    _rpn1EmbossLtColor = \
    embossLtClr; \
    _rpn1EmbossDkColor = \
    embossDkClr; \
    _pRpn1Bitmap = \
    pBitmap; \
    _rpn1EmbossSize = \
    embossSize;
```

Description

This is macro GOLPanelDraw.

6.2.3.25 GOLPanelDrawTsk Function

File

GOL.h

C

```
WORD GOLPanelDrawTsk();
```

Side Effects

none

Returns

Returns the status of the panel rendering

```
0 => Rendering of the panel is not yet finished.  
1 => Rendering of the panel is finished.
```

Preconditions

Parameters of the panel must be set by GOLPanelDraw (■)() macro.

Example

See GOLPanelDraw (■)() example.

Overview

This function draws a panel on the screen with parameters set by GOLPanelDraw (■)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Syntax

WORD GOLPanelDrawTsk()

6.2.3.26 GOLTTwoTonePanelDrawTsk Function

File

GOL.h

C

```
WORD GOLTTwoTonePanelDrawTsk();
```

Side Effects

none

Returns

Returns the status of the panel rendering

```
0 => Rendering of the panel is not yet finished.  
1 => Rendering of the panel is finished.
```

Preconditions

Parameters of the panel must be set by GOLPanelDraw ([\[?\]](#))() macro.

Example

Usage is similar to GOLPanelDraw ([\[?\]](#))() example.

Overview

This function draws a two tone panel on the screen with parameters set by GOLPanelDraw ([\[?\]](#))() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Syntax

```
WORD GOLTTwoTonePanelDrawTsk()
```

6.2.4 GOL Messages

The library provides an interface to accept messages from the input devices.

Enumerations

Name	Description
TRANS_MSG ([?])	This structure defines the list of translated messages for GOL Objects used in the library.
INPUT_DEVICE_EVENT ([?])	This structure defines the types of GOL message events used in the library.
INPUT_DEVICE_TYPE ([?])	This structure defines the types of input devices used in the library.

Functions

	Name	Description
	GOLMsg ([?])	<p>This function receives a GOL message from user and loops through the active list of objects to check which object is affected by the message. For affected objects the message is translated and GOLMsgCallback ([?])() is called. In the call back function, user has the ability to implement action for the message. If the call back function returns non-zero OBJMsgDefault() is called to process message for the object by default. If zero is returned OBJMsgDefault() is not called. Please refer to GOL Messages section for details.</p> <p>This function should be called when GOL drawing is completed. It can be done... more ([?])</p>

	GOLMsgCallback (🔗)	The user MUST implement this function. GOLMsg (🔗 ()) calls this function when a valid message for an object in the active list is received. User action for the message should be implemented here. If this function returns non-zero, the message for the object will be processed by default. If zero is returned, GOL will not perform any action.
---	--------------------------------------	--

Structures

Name	Description
GOL_MSG (🔗)	<p>This structure defines the GOL message used in the library.</p> <ul style="list-style-type: none"> • The types must be one of the INPUT_DEVICE_TYPE (🔗): • TYPE_UNKNOWN • TYPE_KEYBOARD • TYPE_TOUCHSCREEN • TYPE_MOUSE • uiEvent must be one of the INPUT_DEVICE_EVENT (🔗). • for touch screen: • EVENT_INVALID • EVENT_MOVE • EVENT_PRESS • EVENT_STILLPRESS • EVENT_RELEASE • for keyboard: • EVENT_KEYSCAN (param2 contains scan code) • EVENT_KEYCODE (param2 contains character code) • param1: • for touch screen is the x position • for keyboard ID of object receiving the message • param2 • for touch screen y position • for keyboard scan or key code

Description

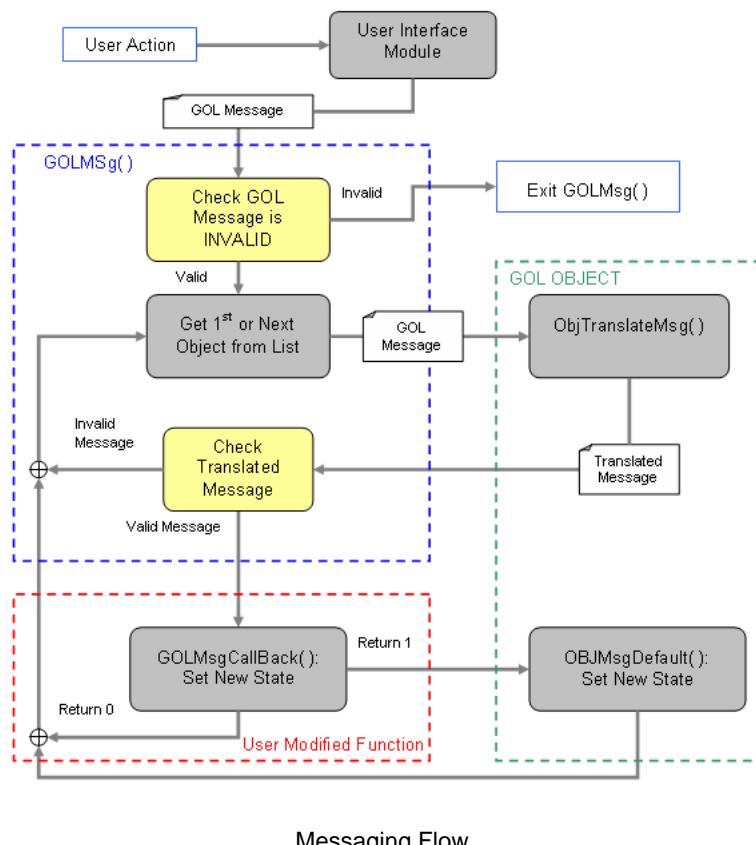
To facilitate the processing of user actions on the objects, messaging are used.

User passes messages from the input devices to GOL using the GOL message structure. The structure is described by the following table.

```
typedef struct {
    BYTE      type;
    BYTE      uiEvent;
    int       param1;
    int       param2;
} GOL_MSG;
```

Field	Description
type	Defines the type of device where the message was created. These are the devices implemented in the User Interface Layer. Possible device types are the following: TYPE_UNKNOWN TYPE_KEYBOARD TYPE_TOUCHSCREEN TYPE_MOUSE
uiEvent	Event ID of the user or device type action on the object. Possible event IDs are the following: EVENT_INVALID EVENT_MOVE EVENT_PRESS EVENT_STILLPRESS EVENT_RELEASE EVENT_KEYSCAN EVENT_CHARCODE
param1 param2	Parameters 1 and 2 definition varies from device types. For example, param1 and param2 are defined as x-coordinate position and y-coordinate position respectively for TYPE_TOUCHSCREEN. For TYPE_KEYBOARD, param1 is defined as the ID of the receiving object and param2 is defined as the keyboard scan or character code.

GOLMsg (✉)() function accepts this structure and processes the message for all objects in the active list.



Messaging Flow

The messaging mechanism follows this flow:

1. User Interface Module (touch screen, keypad) sends a GOL message (the GOL_MSG (✉) structure).
2. A loop evaluates which object is affected by the message. This is done inside GOLMsg (✉)() function.
3. Affected object returns the translated message based on the GOL message parameters.

4. User can change default action with the callback function. If the call back function returns a non-zero value message will be processed by default.
5. Object should be redrawn to reflect new state.

The translated message is a set of actions unique to each object type. Please refer to each object translated message ID for details.

Objects that are disabled will not accept any messages. GOLMsg (█)() function must be called when GOL drawing is completed. In this case all objects have been drawn and it is safe to change objects states. GOLMsg (█)() call can be done if GOLDraw (█)() function returns non-zero or inside GOLDrawCallback (█)() function.

6.2.4.1 GOLMsg Function

File

GOL.h

C

```
void GOLMsg(
    GOL_MSG * pMsg
);
```

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the GOL message from user.

Side Effects

none

Returns

none

Preconditions

none

Example

```
// Assume objects are created & states are set to draw objects
while(1){
    if(GOLDraw()){
        // GOL drawing is completed here
        // it is safe to change objects
        TouchGetMsg(&msg);           // from user interface module
        GOLMsg(&msg);
    }
}
```

Overview

This function receives a GOL message from user and loops through the active list of objects to check which object is affected by the message. For affected objects the message is translated and GOLMsgCallback (█)() is called. In the call back function, user has the ability to implement action for the message. If the call back function returns non-zero OBJMsgDefault() is called to process message for the object by default. If zero is returned OBJMsgDefault() is not called. Please refer to GOL Messages section for details.

This function should be called when GOL drawing is completed. It can be done when GOLDraw (█)() returns non-zero value or inside GOLDrawCallback (█)() function.

Syntax

```
void GOLMsg(GOL_MSG (█) *pMsg)
```

6.2.4.2 GOLMsgCallback Function

File

GOL.h

C

```
WORD GOLMsgCallback(
    WORD objMsg,
    OBJ_HEADER * pObj,
    GOL_MSG * pMsg
);
```

Input Parameters

Input Parameters	Description
WORD objMsg	Translated message for the object or the action ID response from the object.
OBJ_HEADER * pObj	Pointer to the object that processed the message.
GOL_MSG * pMsg	Pointer to the GOL message from user.

Side Effects

none

Returns

Return a non-zero if the message will be processed by default. If a zero is returned, the message will not be processed by GOL.

Preconditions

none

Example

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG *pMsg){
    static char focusSwitch = 1;

    switch(GetObjID(pObj)){
        case ID_BUTTON1:
            // Change text and focus state
            if(objMsg == BTN_MSG_RELEASED){
                focusSwitch ^= 1;
                if(focusSwitch){
                    BtnSetText((BUTTON*)pObj, "Focused");
                    SetState(pObj, BTN_FOCUSED);
                }else{
                    BtnSetText((BUTTON*)pObj, "Unfocused");
                    ClrState(pObj, BTN_FOCUSED);
                }
            }
            // Process by default
            return 1;
        case ID_BUTTON2:
            // Change text
            if(objMsg == BTN_MSG_PRESSED){
                BtnSetText((BUTTON*)pObj, "Pressed");
            }
            if(objMsg == BTN_MSG_RELEASED){
                BtnSetText((BUTTON*)pObj, "Released");
            }
            // Process by default
            return 1;
        case ID_BUTTON3:
            // Change face picture
            if(objMsg == BTN_MSG_PRESSED){
                BtnSetBitmap(pObj, arrowLeft);
            }
            if(objMsg == BTN_MSG_RELEASED){
```

```

        BtnSetBitmap(pObj,(char*)arrowRight);
    }
    // Process by default
    return 1;
case ID_BUTTON_NEXT:
    if(objMsg == BTN_MSG_RELEASED){
        screenState = CREATE_CHECKBOXES;
    }
    // Process by default
    return 1;
case ID_BUTTON_BACK:
    return 1;
default:
    return 1;
}
}

```

Overview

The user MUST implement this function. GOLMsg (■)() calls this function when a valid message for an object in the active list is received. User action for the message should be implemented here. If this function returns non-zero, the message for the object will be processed by default. If zero is returned, GOL will not perform any action.

Syntax

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER (■)* pObj, GOL_MSG (■)* pMsg)
```

6.2.4.3 GOL_MSG Structure

File

GOL.h

C

```

typedef struct {
    BYTE type;
    BYTE uiEvent;
    SHORT param1;
    SHORT param2;
} GOL_MSG;

```

Members

Members	Description
BYTE type;	Type of input device.
BYTE uiEvent;	The generic events for input device.
SHORT param1;	Parameter 1 meaning is dependent on the type of input device.
SHORT param2;	Parameter 2 meaning is dependent on the type of input device.

Overview

This structure defines the GOL message used in the library.

- The types must be one of the INPUT_DEVICE_TYPE (■):
- TYPE_UNKNOWN
- TYPE_KEYBOARD
- TYPE_TOUCHSCREEN
- TYPE_MOUSE
- uiEvent must be one of the INPUT_DEVICE_EVENT (■).
- for touch screen:
- EVENT_INVALID
- EVENT_MOVE

- EVENT_PRESS
- EVENT_STILLPRESS
- EVENT_RELEASE
- for keyboard:
- EVENT_KEYSCAN (param2 contains scan code)
- EVENT_KEYCODE (param2 contains character code)
- param1:
 - for touch screen is the x position
 - for keyboard ID of object receiving the message
- param2
 - for touch screen y position
 - for keyboard scan or key code

6.2.4.4 TRANS_MSG Enumeration

File

GOL.h

C

```
typedef enum {
    OBJ_MSG_INVALID = 0,
    CB_MSG_CHECKED,
    CB_MSG_UNCHECKED,
    RB_MSG_CHECKED,
    WND_MSG_CLIENT,
    WND_MSG_TITLE,
    BTN_MSG_PRESSED,
    BTN_MSG_STILLPRESSED,
    BTN_MSG_RELEASED,
    BTN_MSG_CANCELPRESS,
    PICT_MSG_SELECTED,
    GB_MSG_SELECTED,
    CC_MSG_SELECTED,
    SLD_MSG_INC,
    SLD_MSG_DEC,
    ST_MSG_SELECTED,
    DM_MSG_SELECTED,
    PB_MSG_SELECTED,
    RD_MSG_CLOCKWISE,
    RD_MSG_CTR_CLOCKWISE,
    MTR_MSG_SET,
    EB_MSG_CHAR,
    EB_MSG_DEL,
    EB_MSG_TOUCHSCREEN,
    LB_MSG_SEL,
    LB_MSG_MOVE,
    LB_MSG_TOUCHSCREEN,
    GRID_MSG_TOUCHED,
    GRID_MSG_ITEM_SELECTED,
    GRID_MSG_UP,
    GRID_MSG_DOWN,
    GRID_MSG_LEFT,
    GRID_MSG_RIGHT,
    CH_MSG_SELECTED,
    TE_MSG_RELEASED,
    TE_MSG_PRESSED,
    TE_MSG_ADD_CHAR,
    TE_MSG_DELETE,
    TE_MSG_SPACE,
    TE_MSG_ENTER,
```

```

AC_MSG_PRESSED,
AC_MSG_RELEASED,
OBJ_MSG_PASSIVE
} TRANS_MSG;

```

Members

Members	Description
OBJ_MSG_INVALID = 0	Invalid message response.
CB_MSG_CHECKED	Check Box check action ID.
CB_MSG_UNCHECKED	Check Box un-check action ID.
RB_MSG_CHECKED	Radio Button (■) check action ID.
WND_MSG_CLIENT	Window (■) client area selected action ID.
WND_MSG_TITLE	Window (■) title bar selected action ID.
BTN_MSG_PRESSED	Button (■) pressed action ID.
BTN_MSG_STILLPRESSED	Button (■) is continuously pressed ID.
BTN_MSG_RELEASED	Button (■) released action ID.
BTN_MSG_CANCELPRESS	Button (■) released action ID with button press canceled.
PICT_MSG_SELECTED	Picture (■) selected action ID.
GB_MSG_SELECTED	Group Box selected action ID.
CC_MSG_SELECTED	Custom Control selected action ID.
SLD_MSG_INC	Slider (■) or Scroll (■) Bar (■) increment action ID.
SLD_MSG_DEC	Slider (■) or Scroll (■) Bar (■) decrement action ID.
ST_MSG_SELECTED	Static Text selected action ID.
DM_MSG_SELECTED	Digital Meter (■) selected action ID.
PB_MSG_SELECTED	Progress Bar (■) selected action ID.
RD_MSG_CLOCKWISE	Dial (■) move clockwise action ID.
RD_MSG_CTR_CLOCKWISE	Dial (■) move counter clockwise action ID.
MTR_MSG_SET	Meter (■) set value action ID.
EB_MSG_CHAR	Edit Box insert character action ID.
EB_MSG_DEL	Edit Box remove character action ID.
EB_MSG_TOUCHSCREEN	Edit Box touchscreen selected action ID.
LB_MSG_SEL	List Box item select action ID.
LB_MSG_MOVE	List Box item move action ID.
LB_MSG_TOUCHSCREEN	List Box touchscreen selected action ID.
GRID_MSG_TOUCHED	Grid (■) item touched action ID.
GRID_MSG_ITEM_SELECTED	Grid (■) item selected action ID.
GRID_MSG_UP	Grid (■) up action ID.
GRID_MSG_DOWN	Grid (■) down action ID.
GRID_MSG_LEFT	Grid (■) left action ID.
GRID_MSG_RIGHT	Grid (■) right action ID.
CH_MSG_SELECTED	Chart (■) selected action ID
TE_MSG_RELEASED	TextEntry released action ID
TE_MSG_PRESSED	TextEntry pressed action ID
TE_MSG_ADD_CHAR	TextEntry add character action ID
TE_MSG_DELETE	TextEntry delete character action ID
TE_MSG_SPACE	TextEntry add space character action ID
TE_MSG_ENTER	TextEntry enter action ID
AC_MSG_PRESSED	Analog Clock Pressed Action
AC_MSG_RELEASED	Analog Clock Released Action
OBJ_MSG_PASSIVE	Passive message response. No change in object needed.

Overview

This structure defines the list of translated messages for GOL Objects used in the library.

6.2.4.5 INPUT_DEVICE_EVENT Enumeration

File

GOL.h

C

```
typedef enum {
    EVENT_INVALID = 0,
    EVENT_MOVE,
    EVENT_PRESS,
    EVENT_STILLPRESS,
    EVENT_RELEASE,
    EVENT_KEYSCAN,
    EVENT_CHARCODE,
    EVENT_SET,
    EVENT_SET_STATE,
    EVENT_CLR_STATE
} INPUT_DEVICE_EVENT;
```

Members

Members	Description
EVENT_INVALID = 0	An invalid event.
EVENT_MOVE	A move event.
EVENT_PRESS	A press event.
EVENT_STILLPRESS	A continuous press event.
EVENT_RELEASE	A release event.
EVENT_KEYSCAN	A keyscan event, parameters has the object ID keyboard scan code.
EVENT_CHARCODE	Character code is presented at the parameters.
EVENT_SET	A generic set event.
EVENT_SET_STATE	A set state event.
EVENT_CLR_STATE	A clear state event.

Overview

This structure defines the types of GOL message events used in the library.

6.2.4.6 INPUT_DEVICE_TYPE Enumeration

File

GOL.h

C

```
typedef enum {
    TYPE_UNKNOWN = 0,
    TYPE_KEYBOARD,
    TYPE_TOUCHSCREEN,
    TYPE_MOUSE,
    TYPE_TIMER,
    TYPE_SYSTEM
} INPUT_DEVICE_TYPE;
```

Members

Members	Description
TYPE_UNKNOWN = 0	Unknown device.
TYPE_KEYBOARD	Keyboard.
TYPE_TOUCHSCREEN	Touchscreen.
TYPE_MOUSE	Mouse.
TYPE_TIMER	Timer.
TYPE_SYSTEM	System Messages.

Overview

This structure defines the types of input devices used in the library.

6.2.4.7 Scan Key Codes

Macros

Name	Description
SCAN_BS_PRESSED (█)	Back space key pressed.
SCAN_BS_RELEASED (█)	Back space key released.
SCAN_CR_PRESSED (█)	Carriage return pressed.
SCAN_CR_RELEASED (█)	Carriage return released.
SCAN_DEL_PRESSED (█)	Delete key pressed.
SCAN_DEL_RELEASED (█)	Delete key released.
SCAN_DOWN_PRESSED (█)	Down key pressed.
SCAN_DOWN_RELEASED (█)	Down key released.
SCAN_END_PRESSED (█)	End key pressed.
SCAN_END_RELEASED (█)	End key released.
SCAN_HOME_PRESSED (█)	Home key pressed.
SCAN_HOME_RELEASED (█)	Home key released.
SCAN_LEFT_PRESSED (█)	Left key pressed.
SCAN_LEFT_RELEASED (█)	Left key released.
SCAN_PGDOWN_PRESSED (█)	Page down key pressed.
SCAN_PGDOWN_RELEASED (█)	Page down key released.
SCAN_PGUP_PRESSED (█)	Page up key pressed.
SCAN_PGUP_RELEASED (█)	Page up key released.
SCAN_RIGHT_PRESSED (█)	Right key pressed.
SCAN_RIGHT_RELEASED (█)	Right key released.
SCAN_SPACE_PRESSED (█)	Space key pressed.
SCAN_SPACE_RELEASED (█)	Space key released.
SCAN_TAB_PRESSED (█)	Tab key pressed.
SCAN_TAB_RELEASED (█)	Tab key released.
SCAN_UP_PRESSED (█)	Up key pressed.
SCAN_UP_RELEASED (█)	Up key released.

Description

The defined scan codes for AT keyboard.

6.2.4.7.1 SCAN_BS_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_BS_PRESSED 0x0E
```

Description

Back space key pressed.

6.2.4.7.2 SCAN_BS_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_BS_RELEASED 0x8E
```

Description

Back space key released.

6.2.4.7.3 SCAN_CR_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_CR_PRESSED 0x1C
```

Description

Carriage return pressed.

6.2.4.7.4 SCAN_CR_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_CR_RELEASED 0x9C
```

Description

Carriage return released.

6.2.4.7.5 SCAN_DEL_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_DEL_PRESSED 0x53
```

Description

Delete key pressed.

6.2.4.7.6 SCAN_DEL_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_DEL_RELEASED 0xD3
```

Description

Delete key released.

6.2.4.7.7 SCAN_DOWN_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_DOWN_PRESSED 0x50
```

Description

Down key pressed.

6.2.4.7.8 SCAN_DOWN_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_DOWN_RELEASED 0xD0
```

Description

Down key released.

6.2.4.7.9 SCAN_END_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_END_PRESSED 0x4F
```

Description

End key pressed.

6.2.4.7.10 SCAN_END_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_END_RELEASED 0xCF
```

Description

End key released.

6.2.4.7.11 SCAN_HOME_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_HOME_PRESSED 0x47
```

Description

Home key pressed.

6.2.4.7.12 SCAN_HOME_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_HOME_RELEASED 0xC7
```

Description

Home key released.

6.2.4.7.13 SCAN_LEFT_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_LEFT_PRESSED 0x4B
```

Description

Left key pressed.

6.2.4.7.14 SCAN_LEFT_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_LEFT_RELEASED 0xCB
```

Description

Left key released.

6.2.4.7.15 SCAN_PGDOWN_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_PGDOWN_PRESSED 0x51
```

Description

Page down key pressed.

6.2.4.7.16 SCAN_PGDOWN_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_PGDOWN_RELEASED 0xD1
```

Description

Page down key released.

6.2.4.7.17 SCAN_PGUP_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_PGUP_PRESSED 0x49
```

Description

Page up key pressed.

6.2.4.7.18 SCAN_PGUP_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_PGUP_RELEASED 0xC9
```

Description

Page up key released.

6.2.4.7.19 SCAN_RIGHT_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_RIGHT_PRESSED 0x4D
```

Description

Right key pressed.

6.2.4.7.20 SCAN_RIGHT_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_RIGHT_RELEASED 0xCD
```

Description

Right key released.

6.2.4.7.21 SCAN_SPACE_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_SPACE_PRESSED 0x39
```

Description

Space key pressed.

6.2.4.7.22 SCAN_SPACE_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_SPACE_RELEASED 0xB9
```

Description

Space key released.

6.2.4.7.23 SCAN_TAB_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_TAB_PRESSED 0x0F
```

Description

Tab key pressed.

6.2.4.7.24 SCAN_TAB_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_TAB_RELEASED 0x8F
```

Description

Tab key released.

6.2.4.7.25 SCAN_UP_PRESSED Macro

File

ScanCodes.h

C

```
#define SCAN_UP_PRESSED 0x48
```

Description

Up key pressed.

6.2.4.7.26 SCAN_UP_RELEASED Macro

File

ScanCodes.h

C

```
#define SCAN_UP_RELEASED 0xC8
```

Description

Up key released.

6.2.5 Style Scheme

All objects uses a style scheme structure that defines the font and colors used.

Functions

	Name	Description
!	GOLCreateScheme (🔗)	This function creates a new style scheme object and initializes the parameters to default values. Default values are based on the GOLSchemeDefault (🔗) defined in GOLSchemeDefault.c file. Application code can override this initialization, See GOLSchemeDefault (🔗).

Macros

Name	Description
GOLSetScheme (🔗)	This macro sets the GOL scheme to be used for the object.
GOLGetScheme (🔗)	This macro gets the GOL scheme used by the given object.
GOLGetSchemeDefault (🔗)	This macro returns the default GOL scheme pointer.
GOL_EMBOSS_SIZE (🔗)	This option defines the 3-D effect emboss size for objects. The default value of this is 3 set in GOL.h. If it is not defined in GraphicsConfig.h file then the default value is used.
RGBConvert (🔗)	This macro converts the color data to the selected COLOR_DEPTH (🔗). This macro is only valid when COLOR_DEPTH (🔗) is 8, 16, or 24.

Structures

Name	Description
GOL_SCHEME (🔗)	GOL scheme defines the style scheme to be used by an object.

Variables

Name	Description
GOLFondDefault (🔗)	This is variable GOLFondDefault.
GOLSchemeDefault (🔗)	This defines a default GOL scheme that gets populated when an application calls the GOLCreateScheme (🔗 ()). The application can override this definition by defining the macro GFX_SCHEMEDEFAULT in the GraphicsConfig.h header file and defining GOLSchemeDefault structure in the application code. It is important to use the same structure name since the library assumes that this object exists and assigns the default style scheme pointer to this object.

Description

All objects uses a style scheme structure that defines the font and colors used. Upon the object's creation a user defined style scheme can be assigned to the object. In the absence of the user defined scheme, the default scheme is used.

```
typedef struct {
    WORD             EmbossDkColor;
    WORD             EmbossLtColor;
```

```

WORD           TextColor0;
WORD           TextColor1;
WORD           TextColorDisabled;
WORD           Color0;
WORD           Color1;
WORD           ColorDisabled;
WORD           CommonBkColor;
BYTE          *pFont;
BYTE          AlphaValue;
GFX_GRADIENT_STYLE gradientScheme;
} GOL_SCHEME;

```

Field	Description
EmbossDkColor	Dark emboss color used for the 3-D effect of the object.
EmbossLtColor	Light emboss color used for the 3-D effect of the object.
TextColor0	Generic text colors used by the objects. Usage may vary from one object type to another.
TextColor1	
TextColorDisabled	Text color used for objects that are disabled.
Color0	Generic colors used to render objects. Usage may vary from one object type to another.
Color1	
ColorDisabled	Color used to render objects that are disabled.
CommonBkColor	A common background color of objects. Typically used to hide objects from the screen.
pFont	Pointer to the font table used by the object.
AlphaValue	Alpha value used for alpha blending, this is only available only when USE_ALPHABLEND (■) is defined in the GraphicsConfig.h.
gradientScheme	Gradient Scheme for supported widgets, this is available only when USE_GRADIENT (■) is defined in the GraphicsConfig.h.

TextColorDisabled and ColorDisabled are used when the object is in the disabled state. Otherwise, TextColor0, TextColor1, Color0 and Color1 are used. When object Draw state is set to HIDE (■), the CommonBkColor is used to fill area occupied by object.

Style scheme can be created with GOLCreateScheme (■)() function that returns a pointer to the newly created GOL_SCHEME (■) structure with default values automatically assigned. The default settings of the style scheme for a 16bpp setup are shown below:

Style Parameter	Default Value
EmbossDkColor	EMBOSSDKCOLORDEFAULT
EmbossLtColor	EMBOSSLTCOLORDEFAULT
TextColor0	TEXTCOLOR0DEFAULT
TextColor1	TEXTCOLOR1DEFAULT
TextColorDisabled	TEXTCOLORDISABLEDDEFAULT
Color0	COLOR0DEFAULT
Color1	COLOR1DEFAULT
ColorDisabled	COLORDISABLEDDEFAULT
CommonBkColor	COMMONBACKGROUNDCOLORDEFAULT
pFont	FONTDEFAULT (■)
AlphaValue	0
gradientScheme	{ GRAD_NONE, RGBConvert (■)(0xA9, 0xDB, 0xEF), RGBConvert (■)(0x26, 0xC7, 0xF2), 50 }

The default values can be changed in the GOLSchemeDefault.c file.

The application code can define its own default style scheme by defining the macro GFX_SCHEMEDEFAULT in the GraphicsConfig.h.

Then GOL_SCHEME (■) GOLSchemeDefault (■) must be defined in the application code with each structure member initialized to the desired values. See GOLSchemeDefault.c file for an example on how to initialize the style scheme.

6.2.5.1 GOLCreateScheme Function

File

GOL.h

C

```
GOL_SCHEME * GOLCreateScheme();
```

Side Effects

none

Returns

Pointer to the new GOL_SCHEME (■) created.

Preconditions

none

Example

```
extern const char Font22[] __attribute__((aligned(2)));
extern const char Font16[] __attribute__((aligned(2)));

GOL_SCHEME *pScheme1, *pScheme2;
pScheme1 = GOLCreateScheme();
pScheme2 = GOLCreateScheme();

pScheme1->pFont = (BYTE*)Font22;
pScheme2->pFont = (BYTE*)Font16;
```

Overview

This function creates a new style scheme object and initializes the parameters to default values. Default values are based on the GOLSchemeDefault (■) defined in GOLSchemeDefault.c file. Application code can override this initialization, See GOLSchemeDefault (■).

Syntax

GOL_SCHEME (■) *GOLCreateScheme()

6.2.5.2 GOLSetScheme Macro

File

GOL.h

C

```
#define GOLSetScheme(pObj, pScheme) ((OBJ_HEADER *)pObj)->pGolScheme = pScheme
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.
pScheme	Pointer to the style scheme to be used.

Side Effects

none

Returns

none

Preconditions

none

Example

```
extern FONT_FLASH Gentium12;
GOL_SCHEME *pScheme1;
BUTTON *pButton;

pScheme1 = GOLCreateScheme();
pScheme1->pFont = &Gentium12;

// assume button is created and initialized

// reassign the scheme used by pButton to pScheme1
GOLSetScheme(pButton, pScheme1);
```

Overview

This macro sets the GOL scheme to be used for the object.

Syntax

GOLSetScheme(pObj, pScheme)

6.2.5.3 GOLGetScheme Macro

File

GOL.h

C

```
#define GOLGetScheme(pObj) ((OBJ_HEADER *)pObj)->pGolScheme
```

Input Parameters

Input Parameters	Description
pObj	Pointer to the object of interest.

Side Effects

none

Returns

Returns the style scheme used by the given object.

Preconditions

none

Example

```
GOL_SCHEME *pScheme2;
BUTTON *pButton;

// assume button is created and initialized
// get the scheme assigned to pButton
pScheme2 = GOLGetScheme(pButton);
```

Overview

This macro gets the GOL scheme used by the given object.

Syntax

```
GOLGetScheme(pObj)
```

6.2.5.4 GOLGetSchemeDefault Macro**File**

GOL.h

C

```
#define GOLGetSchemeDefault _pDefaultGolScheme
```

Side Effects

none

Returns

Returns the pointer to the default style scheme.

Preconditions

none

Overview

This macro returns the default GOL scheme pointer.

Syntax

```
GOLGetSchemeDefault()
```

6.2.5.5 GOL_SCHEME Structure**File**

GOL.h

C

```
typedef struct {
    GFX_COLOR EmbossDkColor;
    GFX_COLOR EmbossLtColor;
    GFX_COLOR TextColor0;
    GFX_COLOR TextColor1;
    GFX_COLOR TextColorDisabled;
    GFX_COLOR Color0;
    GFX_COLOR Color1;
    GFX_COLOR ColorDisabled;
    GFX_COLOR CommonBkColor;
    void * pFont;
    WORD AlphaValue;
    GFX_GRADIENT_STYLE gradientScheme;
} GOL_SCHEME;
```

Members

Members	Description
GFX_COLOR EmbossDkColor;	Emboss dark color used for 3d effect.
GFX_COLOR EmbossLtColor;	Emboss light color used for 3d effect.
GFX_COLOR TextColor0;	Character color 0 used for objects that supports text.
GFX_COLOR TextColor1;	Character color 1 used for objects that supports text.
GFX_COLOR TextColorDisabled;	Character color used when object is in a disabled state.
GFX_COLOR Color0;	Color 0 usually assigned to an Object state.

GFX_COLOR Color1;	Color 1 usually assigned to an Object state.
GFX_COLOR ColorDisabled;	Color used when an Object is in a disabled state.
GFX_COLOR CommonBkColor;	Background color used to hide Objects.
void * pFont;	Font selected for the scheme.
WORD AlphaValue;	Alpha value used for alpha blending, this is available only when USE_ALPHABLEND (■) is defined in the GraphicsConfig.h.
GFX_GRADIENT_STYLE gradientScheme;	Gradient Scheme for widgets, this is available only when USE_GRADIENT (■) is defined in the GraphicsConfig.h.

Overview

GOL scheme defines the style scheme to be used by an object.

6.2.5.6 Default Style Scheme Settings

Variables

Name	Description
FONTDEFAULT (■)	Default GOL font.

Description

Lists the default settings for the style scheme.

6.2.5.6.1 FONTDEFAULT Variable

File

GOL.h

C

```
const FONT_FLASH FONTDEFAULT;
```

Description

Default GOL font.

6.2.5.7 GOLFondDefault Variable

File

GOLFondDefault.c

C

```
const FONT_FLASH GOLFondDefault = { (FLASH | COMP_NONE), (GFX_FONT_SPACE char *)__GOLFondDefault };
```

Description

This is variable GOLFondDefault.

6.2.5.8 GOL_EMBOSS_SIZE Macro

File

GOL.h

C

```
#define GOL_EMBOSS_SIZE 3
```

Overview

This option defines the 3-D effect emboss size for objects. The default value of this is 3 set in GOL.h. If it is not defined in GraphicsConfig.h file then the default value is used.

6.2.5.9 GOLSchemeDefault Variable

File

GOLSchemeDefault.c

C

```
const GOL_SCHEME GOLSchemeDefault = { BLACK, WHITE, WHITE, BLACK, WHITE, BLACK, WHITE,
BLACK, BLACK, GRAY006, GRAY010, WHITE, BLACK, GRAY012, GRAY008, GRAY004, GRAY014, BLACK,
RGBConvert(0x2B, 0x55, 0x87), RGBConvert(0xD4, 0xE4, 0xF7), RGBConvert(0x07, 0x1E, 0x48),
RGBConvert(0xFF, 0xFF, 0xFF), RGBConvert(245, 245, 220), RGBConvert(0xA9, 0xDB, 0xEF),
RGBConvert(0x26, 0xC7, 0xF2), RGBConvert(0xB6, 0xD2, 0xFB), RGBConvert(0xD4, 0xED, 0xF7),
(void *)&FONTDEFAULT, 100, #error "The USE_GRADIENT feature is not currently supported when
USE_PALETTE is enabled." { GRAD_NONE, RGBConvert(0xA9, 0xDB, 0xEF), RGBConvert(0x26, 0xC7,
0xF2), 50 } };
```

Overview

This defines a default GOL scheme that gets populated when an application calls the GOLCreateScheme (). The application can override this definition by defining the macro GFX_SCHEMEDEFAULT in the GraphicsConfig.h header file and defining GOLSchemeDefault structure in the application code. It is important to use the same structure name since the library assumes that this object exists and assigns the default style scheme pointer to this object.

6.2.5.10 RGBConvert Macro

File

gfxcolors.h

C

```
#define RGBConvert(red, green, blue) ((GFX_COLOR) (((GFX_COLOR)(red) << 16) |
((GFX_COLOR)(green) << 8) | (GFX_COLOR)(blue)))
```

Input Parameters

Input Parameters	Description
red	red component of the color.
green	green component of the color.
blue	blue component of the color.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro converts the color data to the selected COLOR_DEPTH (). This macro is only valid when COLOR_DEPTH () is 8, 16, or 24.

COLOR_DEPTH (🔗)	Conversion
8	8-8-8 to 3-3-2 conversion
16	8-8-8 to 5-6-5 conversion
24	8-8-8 to 8-8-8 conversion or no conversion

Syntax

```
RGBConvert(red, green, blue)
```

6.2.6 GOL Global Variables

Variables

Name	Description
_pDefaultGolScheme (🔗)	Pointer to the GOL default scheme (GOL_SCHEME (🔗)). This scheme is created in GOLInit (🔗 ()) function. GOL scheme defines the style scheme to be used by an object. Use GOLGetSchemeDefault (🔗 ()) to get this pointer.
_pGolObjects (🔗)	Pointer to the current linked list of objects displayed and receiving messages. GOLDraw (🔗 ()) and GOLMsg (🔗 ()) process objects referenced by this pointer.
_pObjectFocused (🔗)	Pointer to the object receiving keyboard input. This pointer is used or modified by the following APIs: <ul style="list-style-type: none"> • GOLSetFocus (🔗()) • GOLGetFocus (🔗()) • GOLGetFocusNext (🔗()) • GOLGetFocusPrev (🔗()) • GOLCanBeFocused (🔗())

Description

Graphics Object Layer global variables.

6.2.6.1 _pDefaultGolScheme Variable

File

GOL.h

C

```
GOL_SCHEME * _pDefaultGolScheme;
```

Overview

Pointer to the GOL default scheme (GOL_SCHEME ([🔗](#))). This scheme is created in GOLInit ([🔗](#)()) function. GOL scheme defines the style scheme to be used by an object. Use GOLGetSchemeDefault ([🔗](#)()) to get this pointer.

6.2.6.2 _pGolObjects Variable

File

GOL.h

C

```
OBJ_HEADER * _pGolObjects;
```

Overview

Pointer to the current linked list of objects displayed and receiving messages. GOLDraw (🔗)() and GOLMsg (🔗)() process objects referenced by this pointer.

6.2.6.3 _pObjectFocused Variable

File

GOL.h

C

```
OBJ_HEADER * _pObjectFocused;
```

Overview

Pointer to the object receiving keyboard input. This pointer is used or modified by the following APIs:

- GOLSetFocus (🔗)()
- GOLGetFocus (🔗)()
- GOLGetFocusNext (🔗)()
- GOLGetFocusPrev (🔗)()
- GOLCanBeFocused (🔗)()

6.3 Graphics Primitive Layer API

Description of Graphics Primitive Layer API.

Enumerations

Name	Description
GFX_RESOURCE (🔗)	Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.

Structures

Name	Description
GFX_IMAGE_HEADER (🔗)	Structure for images stored in various system memory (Flash, External Memory (SPI, Parallel Flash, or memory in EPMP).
IMAGE_FLASH (🔗)	Structure for images stored in FLASH memory.
IMAGE_RAM (🔗)	Structure for images stored in RAM memory.
GFX_EXTDATA (🔗)	This structure is used to describe external memory.

6.3.1 Text Functions

Functions

	Name	Description
💡	SetFont (🔗)	This function sets the current font used in OutTextXY (🔗)(), OutText (🔗)() and OutChar (🔗)() functions.

	OutChar ([?])	This function outputs a character from the current graphic cursor position. OutChar() uses the current active font set with SetFont ([?] ()).
	OutText ([?])	This function outputs a string of characters starting at the current graphic cursor position. The string must be terminated by a line feed or zero. For Non-Blocking configuration, OutText() may return control to the program due to display device busy status. When this happens zero is returned and OutText() must be called again to continue the outputting of the string. For Blocking configuration, this function always returns a 1. OutText() uses the current active font set with SetFont ([?] ()).
	OutTextXY ([?])	This function outputs a string of characters starting at the given x, y position. The string must be terminated by a line feed or zero. For Non-Blocking configuration, OutTextXY() may return control to the program due to display device busy status. When this happens zero is returned and OutTextXY() must be called again to continue the outputting of the string. For Blocking configuration, this function always returns a 1. OutTextXY() uses the current active font set with SetFont ([?] ()).
	GetTextHeight ([?])	This macro returns the height of the specified font. All characters in a given font table have a constant height.
	GetTextWidth ([?])	This function returns the width of the specified string for the specified font. The string must be terminated by a line feed or zero.

Macros

Name	Description
GetFontOrientation ([?])	Returns font orientation.
SetFontOrientation ([?])	Sets font orientation vertical or horizontal.
GFX_Font_GetAntiAliasType ([?])	Returns the font anti-alias type.
GFX_Font_SetAntiAliasType ([?])	Sets font anti-alias type to either Translucent or opaque.
XCHAR ([?])	This macro sets the data type for the strings and characters. There are three types used for XCHAR and the type is selected by adding one of the macros in GraphicsConfig.h.

Structures

Name	Description
FONT_HEADER ([?])	Structure describing the font header.
FONT_FLASH ([?])	Structure for font stored in FLASH memory.

Types

Name	Description
FONT_EXTERNAL ([?])	Structure for font stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Description

This lists the Primitive level text functions.

6.3.1.1 FONT_HEADER Structure

File

Primitive.h

C

```
typedef struct {
    BYTE fontID;
    BYTE extendedGlyphEntry : 1;
    BYTE res1 : 1;
    BYTE bpp : 2;
    BYTE orientation : 2;
    BYTE res2 : 2;
```

```
WORD firstChar;
WORD lastChar;
WORD height;
} FONT_HEADER;
```

Members

Members	Description
BYTE fontID;	User assigned value
BYTE extendedGlyphEntry : 1;	Extended Glyph entry flag. When set font has extended glyph feature enabled.
BYTE res1 : 1;	Reserved for future use (must be set to 0)
BYTE bpp : 2;	Actual BPP = 2^{bpp} <ul style="list-style-type: none"> • 0 - 1 BPP • 1 - 2 BPP • 2 - 4 BPP • 3 - 8 BPP
BYTE orientation : 2;	Orientation of the character glyphs (0,90,180,270 degrees) <ul style="list-style-type: none"> • 00 - Normal • 01 - Characters rotated 270 degrees clockwise • 10 - Characters rotated 180 degrees • 11 - Characters rotated 90 degrees clockwise
BYTE res2 : 2;	Reserved for future use (must be set to 0).
WORD firstChar;	Character code of first character (e.g. 32).
WORD lastChar;	Character code of last character in font (e.g. 3006).
WORD height;	Font characters height in pixels.

Overview

Structure describing the font header.

6.3.1.2 FONT_FLASH Structure**File**

Primitive.h

C

```
typedef struct {
    GFX_RESOURCE type;
    GFX_FONT_SPACE char * address;
} FONT_FLASH;
```

Members

Members	Description
GFX_RESOURCE type;	must be FLASH
GFX_FONT_SPACE char * address;	font image address in FLASH

Overview

Structure for font stored in FLASH memory.

6.3.1.3 FONT_EXTERNAL Type

File

Primitive.h

C

```
typedef GFX_EXTDATA FONT_EXTERNAL;
```

Overview

Structure for font stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

6.3.1.4 SetFont Function

File

Primitive.h

C

```
void SetFont(
    void * pFont
);
```

Input Parameters

Input Parameters	Description
void * pFont	Pointer to the new font image to be used.

Side Effects

none

Returns

none

Example

See OutTextXY (■)() example.

Overview

This function sets the current font used in OutTextXY (■)(), OutText (■)() and OutChar (■)() functions.

Syntax

```
void SetFont(void* pFont)
```

6.3.1.5 GetFontOrientation Macro

File

Primitive.h

C

```
#define GetFontOrientation _fontOrientation
```

Returns

Return the current font orientation.

- 1 when font orientation is vertical
- 0 when font orientation is horizontal

Preconditions

none

Example

```
void PlaceText(SHORT x, SHORT y, WORD space, XCHAR *pString)
{
    SHORT width;

    SetColor(BRIGHTRED);           // set color
    SetFont(pMyFont);             // set to use global font

    // get string width
    width = GetTextWidth(pString, pMyFont);

    // check if it fits
    if (space < width)
    {
        if (GetFontOrientation() == 0)
            // reset the orientation to vertical
            SetFontOrientation(1);
    }
    else
    {
        if (GetFontOrientation() == 1)
            // reset the orientation to horizontal
            SetFontOrientation(0);
    }
    // place string in the middle of the screen
    OutTextXY(x, y, pString);
}
```

Overview

Returns font orientation.

Syntax

```
GetFontOrientation()
```

6.3.1.6 SetFontOrientation Macro

File

Primitive.h

C

```
#define SetFontOrientation(orient) _fontOrientation = orient;
```

Input Parameters

Input Parameters	Description
orient	sets font orientation when rendering characters and strings. <ul style="list-style-type: none"> • 1 when font orientation is vertical • 0 when font orientation is horizontal

Returns

none

Preconditions

none

Example

See GetFontOrientation ([¶](#))() example.

Overview

Sets font orientation vertical or horizontal.

Syntax

```
SetFontOrientation(orient)
```

6.3.1.7 GFX_Font_GetAntiAliasType Macro**File**

Primitive.h

C

```
#define GFX_Font_GetAntiAliasType
```

Returns

Return the current font anti-alias type.

- ANTIALIAS_TRANSLUCENT (■) - (or 1) when font anti-alias is type translucent
- ANTIALIAS_OPAQUE (■) - (or 0) when font anti-alias is type opaque

Preconditions

Compiler switch USE_ANTIALIASED_FONTS (■) must be enabled

Overview

Returns the font anti-alias type.

Syntax

```
GFX_Font_GetAntiAliasType()
```

6.3.1.8 GFX_Font_SetAntiAliasType Macro**File**

Primitive.h

C

```
#define GFX_Font_SetAntiAliasType(transparency)
```

Input Parameters

Input Parameters	Description
transparency	<p>sets font font anti-alias type</p> <ul style="list-style-type: none"> • ANTIALIAS_TRANSLUCENT (■) - (or 1) when font anti-alias type is translucent • ANTIALIAS_OPAQUE (■) - (or 0) when font anti-alias type is opaque

Returns

none

Preconditions

Compiler switch USE_ANTIALIASED_FONTS (■) must be enabled

Overview

Sets font anti-alias type to either Translucent or opaque.

Syntax

```
GFX_Font_SetAntiAliasType(transparency)
```

6.3.1.9 OutChar Function

File

Primitive.h

C

```
WORD OutChar(
    XCHAR ch
);
```

Input Parameters

Input Parameters	Description
XCHAR ch	The character code to be displayed.

Side Effects

After the function is completed, the graphic cursor position is moved in the horizontal direction by the character width. Vertical position of the graphic cursor is not changed.

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the character is not yet completely drawn.
- Returns 1 when the character is completely drawn.

For Blocking configuration:

- Always return 1.

Preconditions

none

Example

```
static WORD counter = 0;
XCHAR ch;

// render characters until null character
while((XCHAR)(ch = *(textString + counter)) != 0)
{
    if(OutChar(ch) == 0)
        return (0);
    counter++;
}
```

Overview

This function outputs a character from the current graphic cursor position. OutChar() uses the current active font set with SetFont (█)().

Syntax

```
WORD OutChar(XCHAR █ ch)
```

6.3.1.10 OutText Function

File

Primitive.h

C

```
WORD OutText(
    XCHAR * textString
);
```

Input Parameters

Input Parameters	Description
XCHAR * textString	Pointer to the string to be displayed.

Side Effects

Current horizontal graphic cursor position will be moved to the end of the text. The vertical graphic cursor position will not be changed.

Returns

For NON-Blocking configuration:

- Returns 0 when string is not yet outputted completely.
- Returns 1 when string is outputted completely.

For Blocking configuration:

- Always return 1.

Example

```
SetFont(pMyFont);
SetColor(WHITE);
// place the string at the upper left corner of the screen
MoveTo(0, 0);
OutText("Test String!");
```

Overview

This function outputs a string of characters starting at the current graphic cursor position. The string must be terminated by a line feed or zero. For Non-Blocking configuration, OutText() may return control to the program due to display device busy status. When this happens zero is returned and OutText() must be called again to continue the outputting of the string. For Blocking configuration, this function always returns a 1. OutText() uses the current active font set with SetFont (█).

Syntax

```
WORD OutText(XCHAR █* textString)
```

6.3.1.11 OutTextXY Function

File

Primitive.h

C

```
WORD OutTextXY(
    SHORT x,
    SHORT y,
    XCHAR * textString
);
```

Input Parameters

Input Parameters	Description
SHORT x	Defines the x starting position of the string.
SHORT y	Defines the y starting position of the string.
XCHAR * textString	Pointer to the string to be displayed.

Side Effects

Current horizontal graphic cursor position will be moved to the end of the text. The vertical graphic cursor position will not be

changed.

Returns

For NON-Blocking configuration:

- Returns 0 when string is not yet outputted completely.
- Returns 1 when string is outputted completely.

For Blocking configuration:

- Always return 1.

Example

```
void PlaceText(void)
{
    SHORT width, height;
    static const XCHAR text[] = "Touch screen to continue";

    SetColor(BRIGHTRED);           // set color
    SetFont(pMyFont);             // set font to my font

    // get string width & height
    width = GetTextWidth(text, pMyFont);
    height = GetTextHeight(pMyFont);

    // place string in the middle of the screen
    OutTextXY( (GetMaxX() - width) >> 1,
               (GetMaxY() - height) >> 1,
               (char*)text);
}
```

Overview

This function outputs a string of characters starting at the given x, y position. The string must be terminated by a line feed or zero. For Non-Blocking configuration, OutTextXY() may return control to the program due to display device busy status. When this happens zero is returned and OutTextXY() must be called again to continue the outputting of the string. For Blocking configuration, this function always returns a 1. OutTextXY() uses the current active font set with SetFont (■)().

Syntax

WORD OutTextXY(SHORT x, SHORT y, XCHAR (■)* textString)

6.3.1.12 GetTextHeight Function

File

Primitive.h

C

```
SHORT GetTextHeight(
    void * pFont
);
```

Input Parameters

Input Parameters	Description
void * pFont	Pointer to the font image.

Side Effects

none

Returns

Returns the font height.

Example

See OutTextXY (■)() example.

Overview

This macro returns the height of the specified font. All characters in a given font table have a constant height.

Syntax

```
SHORT GetTextHeight(void* pFont)
```

6.3.1.13 GetTextWidth Function

File

Primitive.h

C

```
SHORT GetTextWidth(
    XCHAR * textString,
    void * pFont
);
```

Input Parameters

Input Parameters	Description
XCHAR * textString	Pointer to the string.
void * pFont	Pointer to the font image.

Side Effects

none

Returns

Returns the string width in the specified font.

Example

See OutTextXY (🔗)() example.

Overview

This function returns the width of the specified string for the specified font. The string must be terminated by a line feed or zero.

Syntax

```
SHORT GetTextWidth(XCHAR (🔗)* textString, void* pFont)
```

6.3.1.14 XCHAR Macro

File

Primitive.h

C

```
#define XCHAR char
```

Notes

Only one of the two or none at all are defined in GraphicsConfig.h.

- #define USE_MULTIBYTECHAR (🔗)
- #define USE_UNSIGNED_XCHAR (🔗)
- when none are defined, XCHAR defaults to type char.

Overview

This macro sets the data type for the strings and characters. There are three types used for XCHAR and the type is selected

by adding one of the macros in GraphicsConfig.h.

In GraphicsConfig.h	XCHAR	Description
#define USE_MULTIBYTECHAR (█)	#define XCHAR unsigned short	Use multibyte characters (0-2^16 range).
#define USE_UNSIGNED_XCHAR (█)	#define XCHAR unsigned char	Use unsigned char (0-255 range).
none of the two defined	#define XCHAR char	Use signed char (0-127 range).

6.3.1.15 Anti-Alias Type

Macros

Name	Description
ANTIALIAS_OPAQUE (█)	Mid colors are calculated only once while rendering each character. This is ideal for rendering text over a constant background.
ANTIALIAS_TRANSLUCENT (█)	Mid values are calculated for every necessary pixel. This feature is useful when rendering text over an image or when the background is not one flat color.

Description

Anti-alias type definitions.

6.3.1.15.1 ANTIALIAS_OPAQUE Macro

File

Primitive.h

C

```
#define ANTIALIAS_OPAQUE 0
```

Description

Mid colors are calculated only once while rendering each character. This is ideal for rendering text over a constant background.

6.3.1.15.2 ANTIALIAS_TRANSLUCENT Macro

File

Primitive.h

C

```
#define ANTIALIAS_TRANSLUCENT 1
```

Description

Mid values are calculated for every necessary pixel. This feature is useful when rendering text over an image or when the background is not one flat color.

6.3.2 Gradient

Gradients can be drawn dynamically with the Microchip Graphics Library.

Enumerations

Name	Description
GFX_GRADIENT_TYPE (🔗)	Enumeration for gradient type

Functions

	Name	Description
💡	BarGradient (🔗)	This renders a bar onto the screen, but instead of one color, a gradient is drawn depending on the direction (GFX_GRADIENT_TYPE (🔗)), length, and colors chosen. This function is a blocking call.
💡	BevelGradient (🔗)	This renders a filled bevel with gradient color on the fill. It works the same as the fillbevel function, except a gradient out of color1 and color2 is drawn depending on the direction (GFX_GRADIENT_TYPE (🔗)). This function is a blocking call.

Structures

Name	Description
GFX_GRADIENT_STYLE (🔗)	This structure is used to describe the gradient style.

6.3.2.1 BarGradient Function

File

Primitive.h

C

```
WORD BarGradient(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    GFX_COLOR color1,
    GFX_COLOR color2,
    DWORD length,
    BYTE direction
);
```

Input Parameters

Input Parameters	Description
SHORT left	x position of the left top corner.
SHORT top	y position of the left top corner.
SHORT right	x position of the right bottom corner.
SHORT bottom	y position of the right bottom corner.
GFX_COLOR color1	start color for the gradient
GFX_COLOR color2	end color for the gradient
DWORD length	From 0-100%. How much of a gradient is wanted
BYTE direction	Gradient Direction

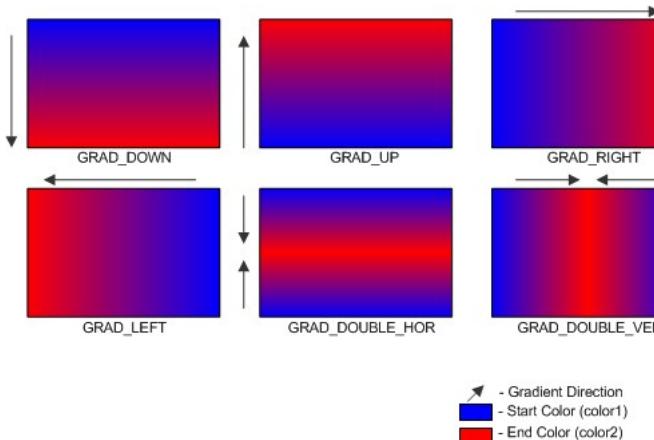
Side Effects

none

Returns

Always returns a 1 since it is a blocking function.

Description



Notes

none

Preconditions

USE_GRADIENT (█) macro must be defined (in GraphicsConfig.h)

Example

```
// draw a full screen gradient background
// with color transitioning from BRIGHTRED to
// BLACK in the upward direction.

GFX_GRADIENT_STYLE gradScheme;

gradScheme.gradientType      = GRAD_UP;
gradScheme.gradientStartColor = BRIGHTRED;
gradScheme.gradientEndColor  = BLACK;

BarGradient(0,
            0,
            GetMaxX(),
            GetMaxY(),
            gradScheme.gradientStartColor,
            gradScheme.gradientEndColor,
            50,
            //left position
            //top position
            //right position
            //bottom position
            the rectangular area
            parameters (full screen),
            // at the halfway point (50%) of
            // defined by the first 4
            // the color becomes BLACK and
            // the rectangle defined is filled
            // see GFX_GRADIENT_TYPE
            BLACK color is used until
            up
            gradScheme.gradientType);
```

Overview

This renders a bar onto the screen, but instead of one color, a gradient is drawn depending on the direction (GFX_GRADIENT_TYPE (█)), length, and colors chosen. This function is a blocking call.

Syntax

```
WORD BarGradient(SHORT left, SHORT top, SHORT right, SHORT bottom, GFX_COLOR color1, GFX_COLOR color2,
DWORD length, BYTE direction);
```

6.3.2.2 BevelGradient Function

File

Primitive.h

C

```
WORD BevelGradient(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    SHORT rad,
    GFX_COLOR color1,
    GFX_COLOR color2,
    DWORD length,
    BYTE direction
);
```

Input Parameters

Input Parameters	Description
SHORT left	x coordinate position of the upper left center of the circle that draws the rounded corners.
SHORT top	y coordinate position of the upper left center of the circle that draws the rounded corners.
SHORT right	x coordinate position of the lower right center of the circle that draws the rounded corners.
SHORT bottom	y coordinate position of the lower right center of the circle that draws the rounded corners.
SHORT rad	defines the radius of the circle, that draws the rounded corners. When rad = 0, the object drawn is a rectangular gradient.
GFX_COLOR color1	start color for the gradient
GFX_COLOR color2	end color for the gradient
DWORD length	From 0-100%. How much of a gradient is wanted
BYTE direction	see GFX_GRADIENT_TYPE (■)

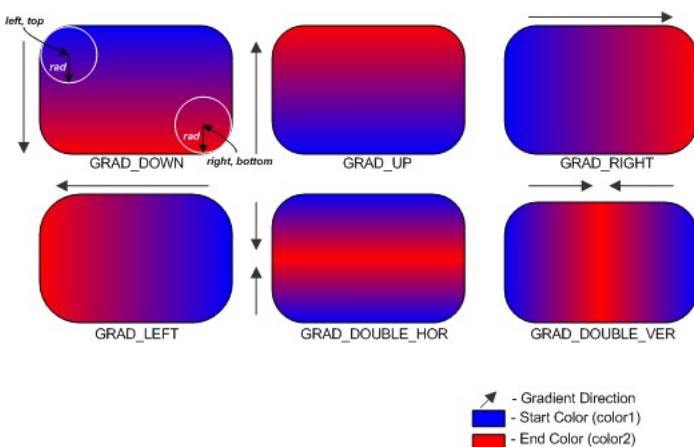
Side Effects

none

Returns

Always returns a 1 since it is a blocking function.

Description



Notes

none

Preconditions

USE_GRADIENT (■) macro must be defined (in GraphicsConfig.h)

Overview

This renders a filled bevel with gradient color on the fill. It works the same as the fillbevel function, except a gradient out of color1 and color2 is drawn depending on the direction (GFX_GRADIENT_TYPE (■)). This function is a blocking call.

Syntax

```
BevelGradient(SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT rad, GFX_COLOR color1, GFX_COLOR color2, DWORD length, BYTE direction);
```

6.3.2.3 GFX_GRADIENT_TYPE Enumeration

File

Primitive.h

C

```
typedef enum {
    GRAD_NONE = 0,
    GRAD_DOWN,
    GRAD_RIGHT,
    GRAD_UP,
    GRAD_LEFT,
    GRAD_DOUBLE_VER,
    GRAD_DOUBLE_HOR
} GFX_GRADIENT_TYPE;
```

Members

Members	Description
GRAD_NONE = 0	No Gradients (■) to be drawn
GRAD_DOWN	gradient changes in the vertical direction
GRAD_RIGHT	gradient change in the horizontal direction
GRAD_UP	gradient changes in the vertical direction
GRAD_LEFT	gradient change in the horizontal direction
GRAD_DOUBLE_VER	two gradient transitions in the vertical direction
GRAD_DOUBLE_HOR	two gradient transitions in the horizontal direction

Overview

Enumeration for gradient type

6.3.2.4 GFX_GRADIENT_STYLE Structure

File

Primitive.h

C

```
typedef struct {
    GFX_GRADIENT_TYPE gradientType;
    DWORD gradientStartColor;
    DWORD gradientEndColor;
    DWORD gradientLength;
} GFX_GRADIENT_STYLE;
```

Members

Members	Description
GFX_GRADIENT_TYPE gradientType;	selected the gradient type
DWORD gradientStartColor;	sets the starting color of gradient transition
DWORD gradientEndColor;	sets the ending color of gradient transition
DWORD gradientLength;	defines the length of the gradient transition in pixels

Overview

This structure is used to describe the gradient style.

6.3.3 Line Functions

Functions

	Name	Description
	Line (■)	This function draws a line with the current line type from the start point to the end point.

Macros

Name	Description
LineRel (■)	This macro draws a line with the current line type from the current graphic cursor position to the position defined by displacement.
LineTo (■)	This macro draws a line with the current line type from the current graphic cursor position to the given x, y position.
SetLineThickness (■)	This macro sets sets line thickness to 1 pixel or 3 pixels.
SetLineType (■)	This macro sets the line type to draw.

Description

This lists the Primitive line text functions.

6.3.3.1 Line Function

File

Primitive.h

C

```
WORD Line(
    SHORT x1,
    SHORT y1,
    SHORT x2,
    SHORT y2
);
```

Input Parameters

Input Parameters	Description
SHORT x1	x coordinate of the start point.
SHORT y1	y coordinate of the start point.
SHORT x2	x coordinate of the end point.
SHORT y2	y coordinate of the end point.

Side Effects

The graphic cursor position is moved to the end point of the line.

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This function draws a line with the current line type from the start point to the end point.

Syntax

```
WORD Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
```

6.3.3.2 LineRel Macro

File

Primitive.h

C

```
#define LineRel(dx, dy) Line(GetX(), GetY(), GetX() + dx, GetY() + dy)
```

Input Parameters

Input Parameters	Description
dX	Displacement from the current x position.
dY	Displacement from the current y position.

Side Effects

The graphic cursor position is moved to the end point of the line.

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This macro draws a line with the current line type from the current graphic cursor position to the position defined by displacement.

Syntax

```
LineRel(dx, dY)
```

6.3.3.3 LineTo Macro

File

Primitive.h

C

```
#define LineTo(x, y) Line(_cursorX, _cursorY, x, y)
```

Input Parameters

Input Parameters	Description
x	End point x position.
y	End point y position.

Side Effects

The graphic cursor position is moved to the end point of the line.

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This macro draws a line with the current line type from the current graphic cursor position to the given x, y position.

Syntax

LineTo(x,y)

6.3.3.4 SetLineThickness Macro

File

Primitive.h

C

```
#define SetLineThickness(lnThickness) _lineThickness = lnThickness;
```

Input Parameters

Input Parameters	Description
InThickness	Line (■) thickness code <ul style="list-style-type: none"> • NORMAL_LINE (■) : 1 pixel • THICK_LINE (■) : 3 pixels

Side Effects

none

Returns

none

Overview

This macro sets line thickness to 1 pixel or 3 pixels.

Syntax

SetLineThickness(InThickness)

6.3.3.5 SetLineType Macro

File

Primitive.h

C

```
#define SetLineType(lnType) _lineType = lnType;
```

Input Parameters

Input Parameters	Description
InType	The type of line to be used. Supported line types: <ul style="list-style-type: none"> • SOLID_LINE ( • DOTTED_LINE ( • DASHED_LINE (

Side Effects

none

Returns

none

Overview

This macro sets the line type to draw.

Syntax

```
SetLineType(InType)
```

6.3.3.6 Line Types

Macros

Name	Description
SOLID_LINE (	Solid Line () Style
DASHED_LINE (	Dashed Line () Style
DOTTED_LINE (	Dotted Line () Style

Description

Line type definitions.

6.3.3.6.1 SOLID_LINE Macro

File

Primitive.h

C

```
#define SOLID_LINE 0
```

Description

Solid Line () Style

6.3.3.6.2 DASHED_LINE Macro

File

Primitive.h

C

```
#define DASHED_LINE 4
```

Description

Dashed Line (■) Style

6.3.3.6.3 DOTTED_LINE Macro

File

Primitive.h

C

```
#define DOTTED_LINE 1
```

Description

Dotted Line (■) Style

6.3.3.7 Line Size

Macros

Name	Description
NORMAL_LINE (■)	Normal Line (■) (thickness is 1 pixel)
THICK_LINE (■)	Thick Line (■) (thickness is 3 pixels)

Description

Line sizes definition.

6.3.3.7.1 NORMAL_LINE Macro

File

Primitive.h

C

```
#define NORMAL_LINE 0
```

Description

Normal Line (■) (thickness is 1 pixel)

6.3.3.7.2 THICK_LINE Macro

File

Primitive.h

C

```
#define THICK_LINE 1
```

Description

Thick Line (■) (thickness is 3 pixels)

6.3.4 Rectangle Functions

Functions

	Name	Description
➊	Bar (➊)	This function draws a bar given the left, top and right, bottom corners with the current set color (SetColor (➋)()). When alpha blending is enabled the bar is alpha blended with the existing pixels specified by the parameters. The alpha percentage used is the last value set by SetAlpha (⌂)().
➋	DrawPoly (⌂)	This function draws a polygon with the current line type using the given number of points. The polygon points (polyPoints) are stored in an array arranged in the following order:

Macros

Name	Description
Rectangle (⌂)	This macro draws a rectangle with the given left, top and right, bottom corners. Current line type is used.

Description

This lists the Primitive level rectangle functions.

6.3.4.1 Bar Function

File

Primitive.h

C

```
WORD Bar(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom
);
```

Input Parameters

Input Parameters	Description
SHORT left	x position of the left top corner.
SHORT top	y position of the left top corner.
SHORT right	x position of the right bottom corner.
SHORT bottom	y position of the right bottom corner.

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This function draws a bar given the left, top and right, bottom corners with the current set color (SetColor (⌂)()). When alpha

blending is enabled the bar is alpha blended with the existing pixels specified by the parameters. The alpha percentage used is the last value set by SetAlpha (█)().

Syntax

```
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
```

6.3.4.2 Rectangle Macro

File

Primitive.h

C

```
#define Rectangle(left, top, right, bottom) Bevel(left, top, right, bottom, 0)
```

Input Parameters

Input Parameters	Description
left	x position of the left top corner.
top	y position of the left top corner.
right	x position of the right bottom corner.
bottom	y position of the right bottom corner.

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This macro draws a rectangle with the given left, top and right, bottom corners. Current line type is used.

Syntax

```
Rectangle(left, top, right, bottom)
```

6.3.4.3 DrawPoly Function

File

Primitive.h

C

```
WORD DrawPoly(
    SHORT numPoints,
    SHORT * polyPoints
);
```

Input Parameters

Input Parameters	Description
SHORT numPoints	Defines the number of x,y points in the polygon.

SHORT * polyPoints	Pointer to the array of polygon points. The array defines the x,y points of the polygon. The sequence should be x0, y0, x1, y1, x2, y2, ... xn, yn where n is the # of polygon sides.
--------------------	---

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Example

```

SHORT OpenShapeXYPoints[6] = {10, 10, 20, 10, 20, 20};
SHORT ClosedShapeXYPoints[8] = {10, 10, 20, 10, 20, 20, 10, 10};

SetColor(WHITE);                                // set color to WHITE
SetLineType(SOLID_LINE);                        // set line to solid line
SetLineThickness(THICK_LINE);                   // set line to thick line
DrawPoly(6, OpenShapeXYPoints);                 // draw the open shape
DrawPoly(8, ClosedShapeXYPoints);               // draw the closed shape

```

Overview

This function draws a polygon with the current line type using the given number of points. The polygon points (polyPoints) are stored in an array arranged in the following order:

```

SHORT polyPoints[size] = {x0, y0, x1, y1, x2, y2 ... xn, yn};
Where n = # of polygon sides
size = numPoints * 2

```

DrawPoly() draws any shape defined by the polyPoints. The function will just draw the lines connecting all the x,y points enumerated by polyPoints[].

Syntax

```
WORD DrawPoly(SHORT numPoints, SHORT* polyPoints)
```

6.3.5 Circle Functions

Functions

	Name	Description
!	Arc (█)	Draws the octant arc of the beveled figure with the given centers, radii and octant mask. When octant = 0xFF and the following are true: <ol style="list-style-type: none"> 1. xL = xR, yT = yB , r1 = 0 and r2 = z, a filled circle is drawn with a radius of z. 2. radii have values (where r1 < r2), a full ring with thickness of (r2-r1) is drawn. 3. xL != xR, yT != yB , r1 = 0 and r2 = 0 (where xR > xL and yB > yT) a rectangle is drawn. xL, yT specifies the left top corner and xR,... more (█)
!	DrawArc (█)	This renders an arc with from startAngle to endAngle with the thickness of r2-r1. The function returns 1 when the arc is rendered successfully and returns a 0 when it is not yet finished. The next call to the function will continue the rendering.

	Bevel (█)	Draws a beveled figure on the screen. When $x1 = x2$ and $y1 = y2$, a circular object is drawn. When $x1 < x2$ and $y1 < y2$ and $rad (radius) = 0$, a rectangular object is drawn.
	FillBevel (█)	Draws a filled beveled figure on the screen. For a filled circular object $x1 = x2$ and $y1 = y2$. For a filled rectangular object $radius = 0$.

Macros

Name	Description
Circle (█)	This macro draws a circle with the given center and radius.
FillCircle (█)	This macro draws a filled circle. Uses the FillBevel (█)() function.
SetBevelDrawType (█)	This macro sets the fill bevel type to be drawn.

Description

This lists the Primitive level circle functions.

6.3.5.1 Circle Macro

File

Primitive.h

C

```
#define Circle(x, y, radius) Bevel(x, y, x, y, radius)
```

Input Parameters

Input Parameters	Description
x	Center x position.
y	Center y position.
radius	the radius of the circle.

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This macro draws a circle with the given center and radius.

Syntax

Circle(x, y, radius)

6.3.5.2 FillCircle Macro

File

Primitive.h

C

```
#define FillCircle(x1, y1, rad) FillBevel(x1, y1, x1, y1, rad)
```

Input Parameters

Input Parameters	Description
x1	x coordinate position of the center of the circle.
y1	y coordinate position of the center of the circle.
rad	defines the radius of the circle.

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This macro draws a filled circle. Uses the FillBevel (█)() function.

Syntax

```
FillCircle(SHORT x1, SHORT y1, SHORT rad)
```

6.3.5.3 Arc Function

File

Primitive.h

C

```
WORD Arc(
    SHORT xL,
    SHORT yT,
    SHORT xR,
    SHORT yB,
    SHORT r1,
    SHORT r2,
    BYTE octant
);
```

Input Parameters

Input Parameters	Description
SHORT xL	x location of the upper left center in the x,y coordinate.
SHORT yT	y location of the upper left center in the x,y coordinate.
SHORT xR	x location of the lower right center in the x,y coordinate.
SHORT yB	y location of the lower right center in the x,y coordinate.
SHORT r1	The smaller radius of the two concentric circles that defines the thickness of the object.
SHORT r2	The larger of radius the two concentric circles that defines the thickness of the object.

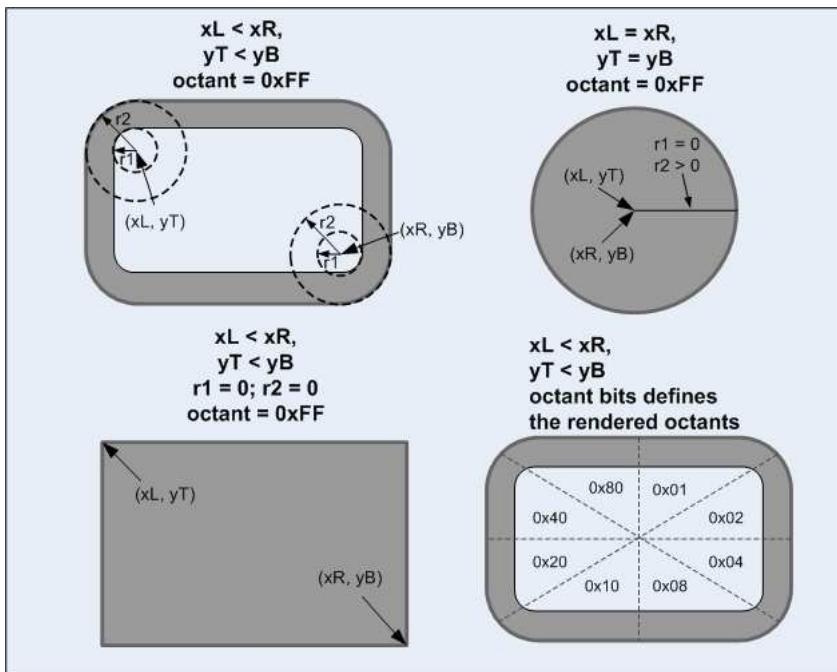
BYTE octant	Bitmask of the octant that will be drawn. Moving in a clockwise direction from $x = 0, y = +\text{radius}$
	<ul style="list-style-type: none"> bit0 : first octant bit1 : second octant bit2 : third octant bit3 : fourth octant bit4 : fifth octant bit5 : sixth octant bit6 : seventh octant bit7 : eighth octant

Side Effects

none

Returns

Returns the rendering status. 1 - If the rendering was completed and 0 - If the rendering is not yet finished.

Description**Preconditions**

none

Overview

Draws the octant arc of the beveled figure with the given centers, radii and octant mask. When $\text{octant} = 0xFF$ and the following are true:

1. $xL = xR, yT = yB, r1 = 0$ and $r2 = z$, a filled circle is drawn with a radius of z .
2. radii have values (where $r1 < r2$), a full ring with thickness of $(r2-r1)$ is drawn.
3. $xL \neq xR, yT \neq yB, r1 = 0$ and $r2 = 0$ (where $xR > xL$ and $yB > yT$) a rectangle is drawn. xL, yT specifies the left top corner and xR, yB specifies the right bottom corner.

When $\text{octant} \neq 0xFF$ the figure drawn is the subsection of the 8 section figure where each non-zero bit of the octant value specifies the octants that will be drawn.

Syntax

```
WORD Arc(SHORT xL, SHORT yT, SHORT xR, SHORT yB, SHORT r1, SHORT r2, BYTE octant)
```

6.3.5.4 DrawArc Function**File**

Primitive.h

C

```
WORD DrawArc(  
    SHORT cx,  
    SHORT cy,  
    SHORT r1,  
    SHORT r2,  
    SHORT startAngle,  
    SHORT endAngle  
) ;
```

Input Parameters

Input Parameters	Description
SHORT cx	the location of the center of the arc in the x direction.
SHORT cy	the location of the center of the arc in the y direction.
SHORT r1	the smaller radius of the arc.
SHORT r2	the larger radius of the arc.
SHORT startAngle	start angle of the arc.
SHORT endAngle	end angle of the arc.

Side Effects

none

Returns

Returns 1 if the rendering is done, 0 if not yet done.

Notes

none

Preconditions

none

Overview

This renders an arc with from startAngle to endAngle with the thickness of r2-r1. The function returns 1 when the arc is rendered successfully and returns a 0 when it is not yet finished. The next call to the function will continue the rendering.

Syntax

```
WORD DrawArc(SHORT cx, SHORT cy, SHORT r1, SHORT r2, SHORT startAngle, SHORT endAngle)
```

6.3.5.5 Bevel Function**File**

Primitive.h

C

```
WORD Bevel(  
    SHORT x1,  
    SHORT y1,
```

```

    SHORT x2,
    SHORT y2,
    SHORT rad
);

```

Input Parameters

Input Parameters	Description
SHORT x1	x coordinate position of the upper left center of the circle that draws the rounded corners.
SHORT y1	y coordinate position of the upper left center of the circle that draws the rounded corners.
SHORT x2	x coordinate position of the lower right center of the circle that draws the rounded corners.
SHORT y2	y coordinate position of the lower right center of the circle that draws the rounded corners.
SHORT rad	defines the radius of the circle, that draws the rounded corners.

Side Effects

none

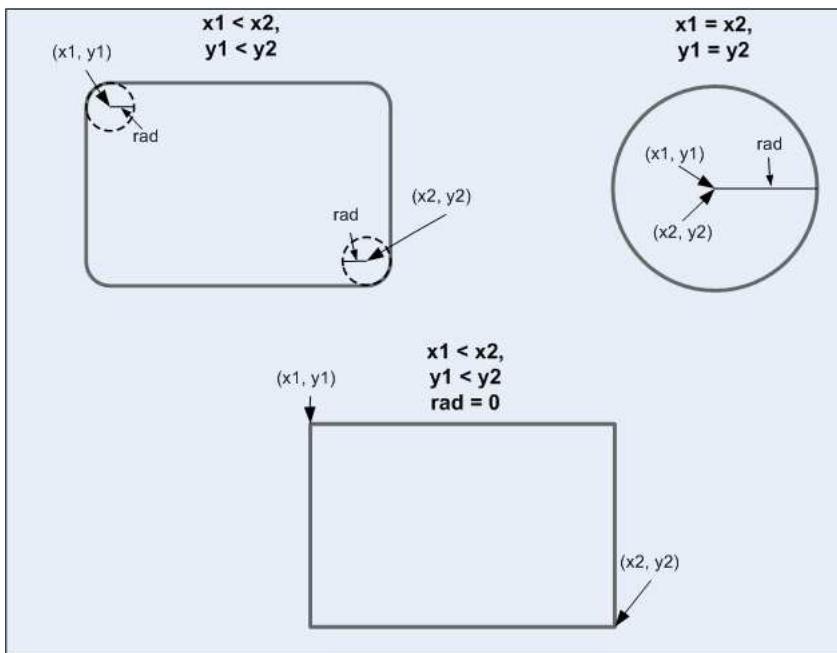
Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Description**Overview**

Draws a beveled figure on the screen. When $x_1 = x_2$ and $y_1 = y_2$, a circular object is drawn. When $x_1 < x_2$ and $y_1 < y_2$ and rad (radius) = 0, a rectangular object is drawn.

Syntax

WORD Bevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)

6.3.5.6 FillBevel Function

File

Primitive.h

C

```
WORD FillBevel(  
    SHORT x1,  
    SHORT y1,  
    SHORT x2,  
    SHORT y2,  
    SHORT rad  
) ;
```

Input Parameters

Input Parameters	Description
SHORT x1	x coordinate position of the upper left center of the circle that draws the rounded corners.
SHORT y1	y coordinate position of the upper left center of the circle that draws the rounded corners.
SHORT x2	x coordinate position of the lower right center of the circle that draws the rounded corners.
SHORT y2	y coordinate position of the lower right center of the circle that draws the rounded corners.
SHORT rad	defines the radius of the circle, that draws the rounded corners.

Side Effects

none

Returns

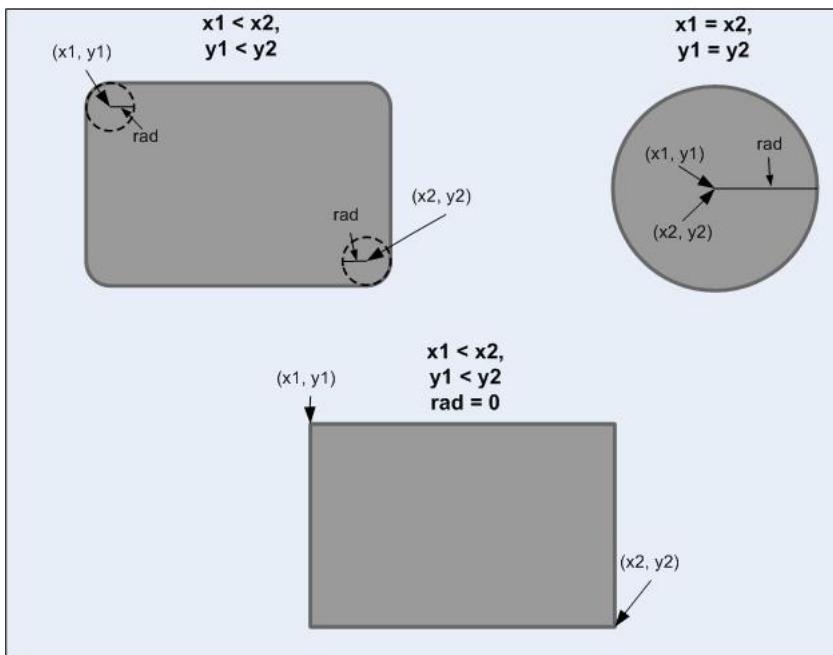
For NON-Blocking configuration:

- Returns 0 when device is busy and the shape is not yet completely drawn.
- Returns 1 when the shape is completely drawn.

For Blocking configuration:

- Always return 1.

Description



Overview

Draws a filled beveled figure on the screen. For a filled circular object $x_1 = x_2$ and $y_1 = y_2$. For a filled rectangular object $radius = 0$.

Syntax

```
WORD FillBevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
```

6.3.5.7 SetBevelDrawType Macro

File

Primitive.h

C

```
#define SetBevelDrawType(type) (_bevelDrawType = type)
```

Input Parameters

Input Parameters	Description
type	<p>is set using the following.</p> <ul style="list-style-type: none"> DRAWFULLBEVEL to draw the full shape DRAWTOPBEVEL to draw the upper half portion DRAWBOTTOMBEVEL to draw the lower half portion

Side Effects

none

Returns

none

Overview

This macro sets the fill bevel type to be drawn.

Syntax

```
SetBevelDrawType(type)
```

6.3.6 Graphic Cursor

Macros

Name	Description
GetX (█)	This macro returns the current graphic cursor x-coordinate.
GetY (█)	This macro returns the current graphic cursor y-coordinate.
MoveRel (█)	This macro moves the graphic cursor relative to the current location. The given dX and dY displacement can be positive or negative numbers.
MoveTo (█)	This macro moves the graphic cursor to new x,y position.

Description

This lists the functions to control the graphics cursor.

6.3.6.1 GetX Macro

File

Primitive.h

C

```
#define GetX _cursorX
```

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro returns the current graphic cursor x-coordinate.

Syntax

```
GetX()
```

6.3.6.2 GetY Macro

File

Primitive.h

C

```
#define GetY _cursorY
```

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro returns the current graphic cursor y-coordinate.

Syntax

GetX (█)()

6.3.6.3 MoveRel Macro

File

Primitive.h

C

```
#define MoveRel(dx, dy) \
    _cursorX += dx;      \
    _cursorY += dy;
```

Input Parameters

Input Parameters	Description
dX	Specifies the displacement of the graphic cursor for the horizontal direction.
dY	Specifies the displacement of the graphic cursor for the vertical direction.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro moves the graphic cursor relative to the current location. The given dX and dY displacement can be positive or negative numbers.

Syntax

MoveRel(dX,dY)

6.3.6.4 MoveTo Macro

File

Primitive.h

C

```
#define MoveTo(x, y) \
    _cursorX = x;      \
    _cursorY = y;
```

Input Parameters

Input Parameters	Description
x	Specifies the new x position of the graphic cursor.
y	Specifies the new y position of the graphic cursor.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro moves the graphic cursor to new x,y position.

Syntax

```
MoveTo(x,y)
```

6.3.7 Alpha Blending Functions

This lists the functions to control Alpha Blending. This feature is enabled only in selected drivers.

Functions

	Name	Description
	AlphaBlendWindow (🔗)	This Alpha-Blends a foreground and a background stored in frames to a destination window. A frame is a memory area that contain array of pixels information. An example would be a display buffer. This operation can be performed on a single frame (where foregroundArea, backgroundArea and destinationArea all points to the same frame), 2 frames (where two of the three areas are pointing to the same frame and one is another frame), or 3 frames (where each area is a separate frame). The Alpha-Blending is performed on the windows inside the specified frames. These windows are defined by the offsets... more (🔗)

Macros

Name	Description
SetAlpha (🔗)	This macro sets the alpha value. Enabling this feature requires the macros USE_ALPHABLEND_LITE (🔗) defined in the GraphicsConfig.h. See USE_ALPHABLEND_LITE (🔗) for information on supported primitive rendering functions.
GetAlpha (🔗)	This macro returns the current alpha value. Enabling this feature requires the macros USE_ALPHABLEND_LITE (🔗) defined in the GraphicsConfig.h.

Description

Alpha-Blend support are two levels. USE_ALPHABLEND_LITE ([🔗](#)) and USE_ALPHA_BLEND.

USE_ALPHABLEND_LITE ([🔗](#))

When this macro is enabled in GraphicsConfig.h, Primitive Layer support for an Alpha-Blended Bar ([🔗](#))() is enabled.

A rectangular area can be Alpha-Blended with a specific color by doing the following:

1. SetAlpha ([🔗](#))() - this will set the Alpha-Blend value

2. SetColor (§) - this will define the color that will be Alpha-Blended to the rectangular area
3. Bar (§)(left, top, right, bottom) - this will perform the Alpha-Blending on the rectangular area defined by left, top, right and bottom parameters.

USE_ALPHABLEND (§)

This is full Alpha-Blend support but is mainly performed by the display driver used. Windows can be Alpha-Blended using the driver level routines with or without the hardware acceleration. Refer to the specific display driver for support.

6.3.7.1 SetAlpha Macro

File

Primitive.h

C

```
#define SetAlpha(alpha) (_alpha = alpha)
```

Input Parameters

Input Parameters	Description
alpha	<p>Defines the alpha blending percentage of the new color set by SetColor (§) to the existing pixel color. Valid values for alpha for pure primitive layer implementation are:</p> <ul style="list-style-type: none"> • 100 : no alpha blending, color set by last SetColor (§) call will replace the pixels. • 75 : alpha blending with new color set by last SetColor (§) call will be alpha blended with 75% to the existing pixel colors. • 50 : alpha blending with new color set by last SetColor (§) call will be alpha blended with 50% to the existing pixel colors. • 25 : alpha blending with new color set by last SetColor (§) call will be alpha blended with 25% to the existing pixel colors.

Side Effects

none

Returns

None

Example

```
SetAlpha(50);           // set alpha level
SetColor(BLUE);        // set color to use
Bar(5,10,30,50);      // render an alpha blended Bar
```

Overview

This macro sets the alpha value. Enabling this feature requires the macros USE_ALPHABLEND_LITE (§) defined in the GraphicsConfig.h. See USE_ALPHABLEND_LITE (§) for information on supported primitive rendering functions.

Syntax

SetAlpha(alpha)

6.3.7.2 GetAlpha Macro

File

Primitive.h

C

```
#define GetAlpha (_alpha)
```

Side Effects

none

Returns

Returns the current alpha value set by the last call to SetAlpha (█)().

Overview

This macro returns the current alpha value. Enabling this feature requires the macros USE_ALPHABLEND_LITE (█) defined in the GraphicsConfig.h.

Syntax

GetAlpha(alpha)

6.3.7.3 AlphaBlendWindow Function

File

Primitive.h

C

```
WORD AlphaBlendWindow(
    DWORD foregroundArea,
    SHORT foregroundLeft,
    SHORT foregroundTop,
    DWORD backgroundArea,
    SHORT backgroundLeft,
    SHORT backgroundTop,
    DWORD destinationArea,
    SHORT destinationLeft,
    SHORT destinationTop,
    WORD width,
    WORD height,
    BYTE alphaPercentage
);
```

Input Parameters

Input Parameters	Description
DWORD foregroundArea	Defines the starting address/page of the foreground window.
SHORT foregroundLeft	Defines the foreground horizontal offset in pixels starting from the starting address/page defined by foregroundArea.
SHORT foregroundTop	Defines the foreground vertical offset in pixels starting from the starting address/page defined by foregroundArea.
DWORD backgroundArea	Defines the starting address/page of the background window.
SHORT backgroundLeft	Defines the background horizontal offset in pixels starting from the starting address/page defined by backgroundArea.
SHORT backgroundTop	Defines the background vertical offset in pixels starting from the starting address/page defined by backgroundArea.
DWORD destinationArea	Defines the starting address/page of the destination window.
SHORT destinationLeft	Defines the destination horizontal offset in pixels starting from the starting address/page defined by destinationArea.
SHORT destinationTop	Defines the destination vertical offset in pixels starting from the starting address/page defined by destinationArea.
WORD width	Defines the width of the window to be alpha blended.
WORD height	Defines the height of the window to be alpha blended.
BYTE alphaPercentage	This defines the amount of transparency to give the foreground Window (█). Valid range is 0-100. Actual allowed values may be limited by the driver used. Refer to the specific driver for allowed values.

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

This Alpha-Blends a foreground and a background stored in frames to a destination window. A frame is a memory area that contain array of pixels information. An example would be a display buffer. This operation can be performed on a single frame (where foregroundArea, backgroundArea and destinationArea all points to the same frame), 2 frames (where two of the three areas are pointing to the same frame and one is another frame), or 3 frames (where each area is a separate frame). The Alpha-Blending is performed on the windows inside the specified frames. These windows are defined by the offsets for each frame and the given width and height. The Alpha-Blended windows are always equal in sizes. This function is only available when it is supported by the display driver used. Enabling this feature requires the macros USE_ALPHABLEND_LITE ([🔗](#)) or USE_ALPHABLEND ([🔗](#)) defined in the GraphicsConfig.h.

Syntax

```
void AlphaBlendWindow(DWORD foregroundArea, SHORT foregroundLeft, SHORT foregroundTop, DWORD
backgroundArea, SHORT backgroundLeft, SHORT backgroundTop, DWORD destinationArea, SHORT destinationLeft,
SHORT destinationTop, WORD width, WORD height, BYTE alphaPercentage)
```

6.3.8 Bitmap Functions

Functions

	Name	Description
	PutImagePartial (🔗)	This function outputs a full or a partial image starting from left,top coordinates. The partial image starts at xoffset and yoffset. Size is specified by the given width and height parameters.
	GetImageHeight (🔗)	This function returns the image height.
	GetImageWidth (🔗)	This function returns the image width.

Macros

Name	Description
PutImage (🔗)	This renders the image pointed to by "image" starting from left, top coordinates.

Structures

Name	Description
BITMAP_HEADER (🔗)	Structure describing the bitmap header.

Description

This lists the functions to display bitmaps.

6.3.8.1 PutImage Macro

File

Primitive.h

C

```
#define PutImage(left, top, image, stretch) PutImagePartial(left, top, image, stretch, 0,  
0, 0, 0)
```

Input Parameters

Input Parameters	Description
left	horizontal starting position of the full image on the screen
top	vertical starting position of the full image on the screen
image	pointer to the image location.
stretch	The image stretch factor. <ul style="list-style-type: none"> • IMAGE_NORMAL (■) : no stretch • IMAGE_X2 (■) : image is stretched to twice its width and height

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and the image is not yet completely drawn.
- Returns 1 when the image is completely drawn.

For Blocking configuration:

- Always return 1.

Overview

This renders the image pointed to by "image" starting from left, top coordinates.

Syntax

```
WORD PutImage(SHORT left, SHORT top, void* image, BYTE stretch)
```

6.3.8.2 PutImagePartial Function

File

Primitive.h

C

```
WORD PutImagePartial(  
    SHORT left,  
    SHORT top,  
    void * image,  
    BYTE stretch,  
    SHORT xoffset,  
    SHORT yoffset,  
    WORD width,  
    WORD height  
) ;
```

Input Parameters

Input Parameters	Description
SHORT left	horizontal starting position of full or partial image on the screen
SHORT top	vertical starting position of full or partial image on the screen,
void * image	pointer to the image location.
BYTE stretch	The image stretch factor. <ul style="list-style-type: none"> • IMAGE_NORMAL (■) : no stretch • IMAGE_X2 (■) : image is stretched to twice its width and height
SHORT xoffset	Specifies the horizontal offset in pixels of the selected partial image from the left most pixel of the full image.
SHORT yoffset	Specifies the vertical offset in pixels of the selected partial image from the top most pixel of the full image.
WORD width	width of the partial image to be rendered. xoffset + width must not exceed the full image width.
WORD height	height of the partial image to be rendered. yoffset + height must not exceed the full image height.

Side Effects

none

Returns

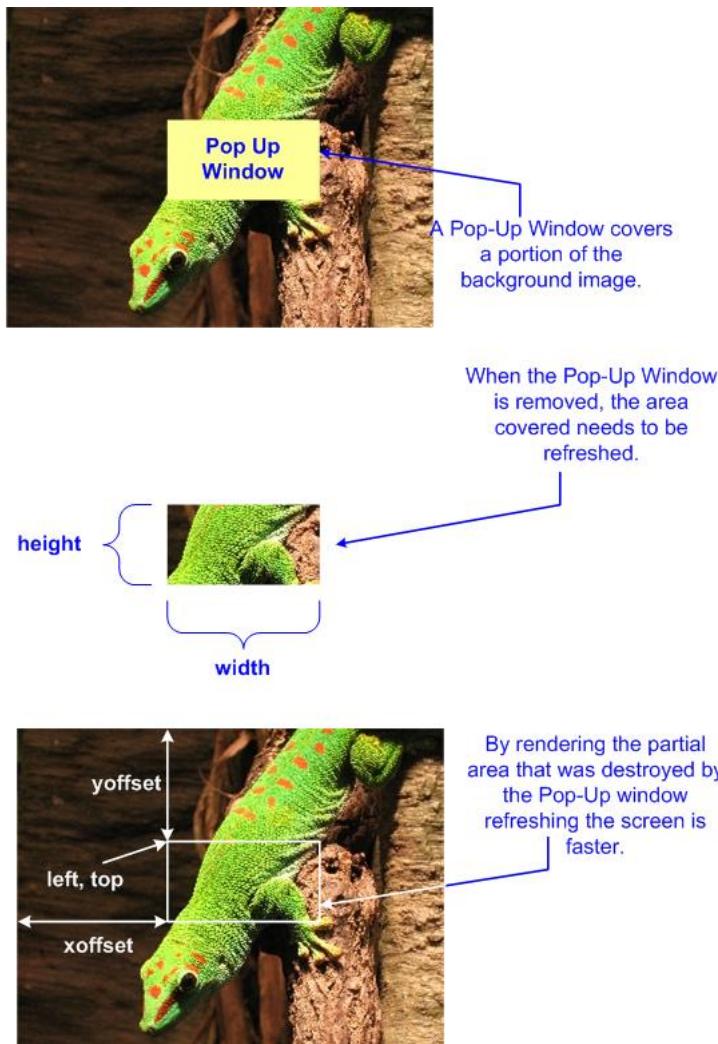
For NON-Blocking configuration:

- Returns 0 when device is busy and the image is not yet completely drawn.
- Returns 1 when the image is completely drawn.

For Blocking configuration:

- Always return 1.

Description



Overview

This function outputs a full or a partial image starting from left,top coordinates. The partial image starts at xoffset and yoffset. Size is specified by the given width and height parameters.

Syntax

```
WORD PutImagePartial(SHORT left, SHORT top, void* image, BYTE stretch, SHORT xoffset, SHORT yoffset, WORD width,
WORD height)
```

6.3.8.3 GetImageHeight Function

File

Primitive.h

C

```
SHORT GetImageHeight(
    void * bitmap
);
```

Input Parameters

Input Parameters	Description
void * bitmap	Pointer to the bitmap.

Side Effects

none

Returns

Returns the image height in pixels.

Overview

This function returns the image height.

Syntax

SHORT GetImageHeight(void* bitmap)

6.3.8.4 GetImageWidth Function

File

Primitive.h

C

```
SHORT GetImageWidth(
    void * bitmap
);
```

Input Parameters

Input Parameters	Description
void * bitmap	Pointer to the bitmap.

Side Effects

none

Returns

Returns the image width in pixels.

Overview

This function returns the image width.

Syntax

SHORT GetImageWidth(void* bitmap)

6.3.8.5 BITMAP_HEADER Structure

File

Primitive.h

C

```
typedef struct {
    BYTE compression;
    BYTE colorDepth;
    SHORT height;
    SHORT width;
} BITMAP_HEADER;
```

Members

Members	Description
BYTE compression;	Compression setting
BYTE colorDepth;	Color depth used
SHORT height;	Image height
SHORT width;	Image width

Overview

Structure describing the bitmap header.

6.3.8.6 Bitmap Settings

Macros

Name	Description
IMAGE_NORMAL (🔗)	Normal image stretch code
IMAGE_X2 (🔗)	Stretched image stretch code

Description

Bitmap rendering settings.

6.3.8.6.1 IMAGE_NORMAL Macro

File

Primitive.h

C

```
#define IMAGE_NORMAL 1
```

Description

Normal image stretch code

6.3.8.6.2 IMAGE_X2 Macro

File

Primitive.h

C

```
#define IMAGE_X2 2
```

Description

Stretched image stretch code

6.3.8.7 Bitmap Source

Bitmap data structure is dependent on the location.

6.3.9 External Memory

This lists the external memory access functions and descriptions.

Functions

	Name	Description
💡	ExternalMemoryCallback (🔗)	This function must be implemented in the application. The library will call this function each time when the external memory data will be required. The application must copy requested bytes quantity into the buffer provided. Data start address in external memory is a sum of the address in GFX_EXTDATA (🔗) structure and offset.

Macros

Name	Description
EXTERNAL_FONT_BUFFER_SIZE (🔗)	This defines the size of the buffer used by font functions to retrieve font data from the external memory. The buffer size can be increased to accommodate large font sizes. The user must be aware of the expected glyph sizes of the characters stored in the font table. To modify the size used, declare this macro in the GraphicsConfig.h file with the desired size.

Description

Bitmaps and Fonts can be located in external memory. To refer the data, EXTDATA structure is used:

```
typedef struct _EXTDATA_
{
    TYPE_MEMORY type;           // must be set to EXTERNAL
    WORD ID;                  // memory ID
    DWORD address;             // bitmap or font image address
} EXTDATA;
```

where:

- type – shows type of memory used.
- 0 – internal
- 1 - external.
- ID – unique number must be assigned by application to have a way distinguishing memory chips if the application has several of them.
- address – start address of the bitmap or font image in the external memory.

To use the bitmap or font the pointer to EXTDATA structure must be passed into corresponding function (PutImage (🔗)() or SetFont (🔗)()). Each time the library will need data it will call special call back function ExternalMemoryCallback (🔗)().

This function must be implemented in the application. Inside, the application must copy requested bytes quantity into the buffer provided. Data start address can be calculated as a sum of the start image address specified in EXTDATA structure and offset provided.

6.3.9.1 ExternalMemoryCallback Function

File

Primitive.h

C

```
WORD ExternalMemoryCallback(
    GFX_EXTDATA * memory,
    LONG offset,
    WORD nCount,
    void * buffer
);
```

Input Parameters

Input Parameters	Description
GFX_EXTDATA * memory	Pointer to the external memory bitmap or font structures (FONT_EXTERNAL (🔗) or BITMAP_EXTERNAL).

LONG offset	Data offset.
WORD nCount	Number of bytes to be transferred into the buffer.
void * buffer	Pointer to the buffer.

Side Effects

none

Returns

Returns the number of bytes were transferred.

Example

```
// If there are several memories in the system they can be selected by IDs.
// In this example, ID for memory device used is assumed to be 0.
#define X_MEMORY 0

WORD ExternalMemoryCallback(GFX_EXTDATA* memory, LONG offset, WORD nCount, void* buffer) {
    int i;
    long address;

    // Address of the requested data is a start address of the object referred by
    GFX_EXTDATA structure plus offset
    address = memory->address+offset;

    if(memory->ID == X_MEMORY){
        // MemoryXReadByte() is some function implemented to access external memory.
        // Implementation will be specific to the memory used. In this example
        // it reads byte each time it is called.
        i = 0;
        while (i < nCount) {
            (BYTE*)buffer = MemoryXReadByte(address++);
            i++;
        }
    }
    // return the actual number of bytes retrieved
    return (i);
}
```

Overview

This function must be implemented in the application. The library will call this function each time when the external memory data will be required. The application must copy requested bytes quantity into the buffer provided. Data start address in external memory is a sum of the address in GFX_EXTDATA (■) structure and offset.

Syntax

WORD ExternalMemoryCallback(GFX_EXTDATA (■)* memory, LONG offset, WORD nCount, void* buffer)

6.3.9.2 EXTERNAL_FONT_BUFFER_SIZE Macro

File

Primitive.h

C

```
#define EXTERNAL_FONT_BUFFER_SIZE 600
```

Overview

This defines the size of the buffer used by font functions to retrieve font data from the external memory. The buffer size can be increased to accommodate large font sizes. The user must be aware of the expected glyph sizes of the characters stored in the font table. To modify the size used, declare this macro in the GraphicsConfig.h file with the desired size.

6.3.9.3 Memory Type

Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.

6.3.10 Set Up Functions

Functions

	Name	Description
◆	ClearDevice ()	This function clears the screen with the current color and sets the graphic cursor position to (0, 0). Clipping is NOT supported by ClearDevice().
◆	InitGraph ()	This function initializes the display controller, sets the line type to SOLID_LINE (), sets the screen to all BLACK (), sets the current drawing color to WHITE (), sets the graphic cursor position to upper left corner of the screen, sets active and visual pages to page #0, clears the active page and disables clipping. This function should be called before using the Graphics Primitive Layer.

Description

This lists the Primitive set up and initialization functions.

6.3.10.1 ClearDevice Function

File

Primitive.h

C

```
void ClearDevice();
```

Side Effects

none

Returns

none

Example

```
void ClearScreen(void)
{
    SetColor(WHITE);           // set color to WHITE
    ClearDevice();             // set screen to all WHITE
}
```

Overview

This function clears the screen with the current color and sets the graphic cursor position to (0, 0). Clipping is NOT supported by ClearDevice().

Syntax

```
void ClearDevice(void)
```

6.3.10.2 InitGraph Function

File

Primitive.h

C

```
void InitGraph();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

This function initializes the display controller, sets the line type to SOLID_LINE (█), sets the screen to all BLACK (█), sets the current drawing color to WHITE (█), sets the graphic cursor position to upper left corner of the screen, sets active and visual pages to page #0, clears the active page and disables clipping. This function should be called before using the Graphics Primitive Layer.

Syntax

```
void InitGraph(void)
```

6.3.11 GFX_RESOURCE Enumeration

File

Primitive.h

C

```
typedef enum {
    FLASH = 0x0000,
    EXTERNAL = 0x0001,
    FLASH_JPEG = 0x0002,
    EXTERNAL_JPEG = 0x0003,
    RAM = 0x0004,
    EDS_EPMP = 0x0005,
    IMAGE_MBITMAP = 0x0000,
    IMAGE_JPEG = 0x0100,
    COMP_NONE = 0x0000,
    COMP_RLE = 0x1000,
    COMP_IPU = 0x2000
} GFX_RESOURCE;
```

Members

Members	Description
FLASH = 0x0000	internal flash
EXTERNAL = 0x0001	external memory
FLASH_JPEG = 0x0002	internal flash
EXTERNAL_JPEG = 0x0003	external memory
RAM = 0x0004	RAM
EDS_EPMP = 0x0005	memory in EPMP, base addresses are set in the hardware profile

IMAGE_MBITMAP = 0x0000	data resource is type Microchip bitmap
IMAGE_JPEG = 0x0100	data resource is type JPEG
COMP_NONE = 0x0000	no compression
COMP_RLE = 0x1000	compressed with RLE
COMP_IPU = 0x2000	compressed with DEFLATE (for IPU)

Overview

Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.

6.3.12 GFX_IMAGE_HEADER Structure

File

Primitive.h

C

```
typedef struct {
    GFX_RESOURCE type;
    WORD ID;
    union {
        DWORD extAddress;
        FLASH_BYTE * progByteAddress;
        FLASH_WORD * progWordAddress;
        const char * constAddress;
        char * ramAddress;
        __eds__ char * edsAddress;
    } LOCATION;
    WORD width;
    WORD height;
    DWORD param1;
    DWORD param2;
    WORD colorDepth;
} GFX_IMAGE_HEADER;
```

Members

Members	Description
GFX_RESOURCE type;	Graphics resource type, determines the type and location of data
WORD ID;	memory ID, user defined value to differentiate between graphics resources of the same type When using EDS_EPMP the following ID values are reserved and used by the Microchip display driver 0 - reserved (do not use) 1 - reserved for base address of EPMP CS1 2 - reserved for base address of EPMP CS2
DWORD extAddress;	generic address
FLASH_BYTE * progByteAddress;	for addresses in program section
FLASH_WORD * progWordAddress;	for addresses in program section
const char * constAddress;	for addresses in FLASH
char * ramAddress;	for addresses in RAM
__eds__ char * edsAddress;	for addresses in EDS
WORD width;	width of the image
WORD height;	height of the image
DWORD param1;	Parameters used for the GFX_RESOURCE (). Depending on the GFX_RESOURCE () type definition of param1 can change. For IPU and RLE compressed images, param1 indicates the compressed size of the image.
DWORD param2;	Parameters used for the GFX_RESOURCE (). Depending on the GFX_RESOURCE () type definition of param2 can change. For IPU and RLE compressed images, param2 indicates the uncompressed size of the image.

WORD colorDepth;	color depth of the image
------------------	--------------------------

Overview

Structure for images stored in various system memory (Flash, External Memory (SPI, Parallel Flash, or memory in EPMP)).

6.3.13 IMAGE_FLASH Structure

File

Primitive.h

C

```
typedef struct {
    GFX_RESOURCE type;
    FLASH_BYTE * address;
} IMAGE_FLASH;
```

Members

Members	Description
GFX_RESOURCE type;	must be FLASH
FLASH_BYTE * address;	image address in FLASH

Overview

Structure for images stored in FLASH memory.

6.3.14 IMAGE_RAM Structure

File

Primitive.h

C

```
typedef struct {
    GFX_RESOURCE type;
    DWORD * address;
} IMAGE_RAM;
```

Members

Members	Description
GFX_RESOURCE type;	must be RAM
DWORD * address;	image address in RAM

Overview

Structure for images stored in RAM memory.

6.3.15 GFX_EXTDATA Structure

File

Primitive.h

C

```
typedef struct {
    GFX_RESOURCE type;
    WORD ID;
    DWORD address;
} GFX_EXTDATA;
```

Members

Members	Description
GFX_RESOURCE type;	Resource type. Valid types are: <ul style="list-style-type: none"> • EXTERNAL • EDS_EPMP
WORD ID;	Memory ID, user defined value to differentiate between graphics resources of the same type. When using EDS_EPMP the following ID values are reserved and used by the Microchip display driver <ul style="list-style-type: none"> • 0 - reserved (do not use) • 1 - reserved for base address of EPMP CS1 • 2 - reserved for base address of EPMP CS2
DWORD address;	Data image address (user data, bitmap or font)

Overview

This structure is used to describe external memory.

6.4 Display Device Driver Layer API

Description of Display Device Driver Layer API.

Modules

Name	Description
Advanced Display Driver Features (🔗)	This section lists advanced Display Device Driver Features implemented in select Display Device Driver.

6.4.1 Display Device Driver Level Primitives

This lists the Device Level Primitive rendering functions and macros.

Functions

	Name	Description
💡	GetPixel (🔗)	Returns pixel color at the given x,y coordinate position.
💡	PutPixel (🔗)	Puts pixel with the given x,y coordinate position.
💡	SetClip (🔗)	Enables/disables clipping.
💡	SetClipRgn (🔗)	Sets clipping region.
💡	TransparentColorEnable (🔗)	Sets current transparent color. PutImage (🔗)() will not render pixels that matches the set transparent color. To enable Transparent Color feature, define the macro USE_TRANSPARENT_COLOR (🔗) in the GraphicsConfig.h file.
💡	DisplayBrightness (🔗)	Sets the brightness of the display.

	CopyBlock (🔗)	Copies a block of data from source specified by srcAddr and srcOffset to the destination specified by dstAddr and dstOffset. This is similar to the CopyWindow (🔗 ()) and but instead of using left, top position of the source and destination, it uses offsets instead. This is a blocking function.
	CopyPageWindow (🔗)	This is a blocking call. A windows is a rectangular area with the given width and height of a page. The source and destination window can be located in different pages and each page is assumed to have the same dimensions (equal width and height).
	CopyWindow (🔗)	A windows is a rectangular area with the given width and height located in the given base source address. The source and destination window can be located in the same base address. If this is the case, then srcAddr = dstAddr. The operation is similar to CopyPageWindow (🔗 ()) but instead of using page numbers, addresses are used for versatility. This is a blocking function.
	SetActivePage (🔗)	Sets active graphic page.
	SetVisualPage (🔗)	Sets graphic page to display.

Macros

Name	Description
GetColor (🔗)	Returns current drawing color.
SetColor (🔗)	Sets current drawing color.
GetMaxX (🔗)	Returns maximum horizontal coordinate.
GetMaxY (🔗)	Returns maximum vertical coordinate.
GetClipBottom (🔗)	Returns bottom clipping border.
GetClipLeft (🔗)	Returns left clipping border.
GetClipRight (🔗)	Returns right clipping border.
GetClipTop (🔗)	Returns top clipping border.
CLIP_DISABLE (🔗)	Disables clipping.
CLIP_ENABLE (🔗)	Enables clipping.
TransparentColorDisable (🔗)	Disables the transparent color function.
GetTransparentColorStatus (🔗)	Returns the current transparent color function enable status.
GetTransparentColor (🔗)	Returns the current transparent color value.
TRANSPARENT_COLOR_DISABLE (🔗)	Check of transparent color is not performed
TRANSPARENT_COLOR_ENABLE (🔗)	Check pixel if color is equal to transparent color, if equal do not render pixel
GetPageAddress (🔗)	Returns the address of the given page.

6.4.1.1 GetPixel Function

File

DisplayDriver.h

C

```
GFX_COLOR GetPixel(
    SHORT x,
    SHORT y
);
```

Input Parameters

Input Parameters	Description
SHORT x	x position of the pixel.
SHORT y	y position of the pixel.

Side Effects

none

Returns

pixel color

Preconditions

none

Overview

Returns pixel color at the given x,y coordinate position.

Syntax

```
GFX_COLOR GetPixel(SHORT x, SHORT y)
```

6.4.1.2 PutPixel Function

File

DisplayDriver.h

C

```
void PutPixel(  
    SHORT x,  
    SHORT y  
) ;
```

Input Parameters

Input Parameters	Description
SHORT x	x position of the pixel.
SHORT y	y position of the pixel.

Side Effects

none

Returns

none

Preconditions

none

Overview

Puts pixel with the given x,y coordinate position.

Syntax

```
void PutPixel(SHORT x, SHORT y)
```

6.4.1.3 GetColor Macro

File

DisplayDriver.h

C

```
#define GetColor _color
```

Side Effects

none

Returns

Color where coding is based on GFX_COLOR definition. GFX_COLOR definition is based on the color depth (COLOR_DEPTH (█)) used.

Preconditions

none

Overview

Returns current drawing color.

Syntax

GetColor()

6.4.1.4 SetColor Macro

File

DisplayDriver.h

C

```
#define SetColor(color) _color = (color)
```

Input Parameters

Input Parameters	Description
color	Color coding is based on GFX_COLOR definition. GFX_COLOR definition is based on the color depth (COLOR_DEPTH (█)) used.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets current drawing color.

Syntax

SetColor(color)

6.4.1.5 GetMaxX Macro

File

DisplayDriver.h

C

```
#define GetMaxX (DISP_HOR_RESOLUTION - 1)
```

Side Effects

none

Returns

Maximum horizontal coordinate.

Preconditions

none

Example

```
// Create a window that will occupy the whole screen.
WndCreate(0xFF, // ID
          0,0,
          GetMaxX(),GetMaxY(), // dimension
          WND_DRAW, // will be displayed after creation
          (void*)&mchpIcon, // use icon used
          pText, // set to text pointed to by pText
          NULL); // use default scheme
```

Overview

Returns maximum horizontal coordinate.

Syntax

GetMaxX()

6.4.1.6 GetMaxY Macro

File

DisplayDriver.h

C

```
#define GetMaxY (DISP_VER_RESOLUTION - 1)
```

Side Effects

none

Returns

Maximum vertical coordinate.

Preconditions

none

Example

(see GetMaxX (图)() example.

Overview

Returns maximum vertical coordinate.

Syntax

GetMaxY()

6.4.1.7 SetClip Function

File

DisplayDriver.h

C

```
void SetClip(
    BYTE control
);
```

Input Parameters

Input Parameters	Description
BYTE control	Enables or disables the clipping. <ul style="list-style-type: none"> • CLIP_DISABLE (■): Disable clipping • CLIP_ENABLE (■): Enable clipping

Side Effects

none

Returns

none

Preconditions

none

Overview

Enables/disables clipping.

Syntax

SetClip(control)

6.4.1.8 SetClipRgn Function

File

DisplayDriver.h

C

```
void SetClipRgn(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom
);
```

Input Parameters

Input Parameters	Description
SHORT left	Defines the left clipping region border.
SHORT top	Defines the top clipping region border.
SHORT right	Defines the right clipping region border.
SHORT bottom	Defines the bottom clipping region border.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets clipping region.

Syntax

SetClipRgn(left, top, right, bottom)

6.4.1.9 GetClipBottom Macro

File

DisplayDriver.h

C

```
#define GetClipBottom _clipBottom
```

Side Effects

none

Returns

Bottom clipping border.

Preconditions

none

Overview

Returns bottom clipping border.

Syntax

```
GetClipBottom()
```

6.4.1.10 GetClipLeft Macro

File

DisplayDriver.h

C

```
#define GetClipLeft _clipLeft
```

Side Effects

none

Returns

Left clipping border.

Preconditions

none

Overview

Returns left clipping border.

Syntax

```
GetClipLeft()
```

6.4.1.11 GetClipRight Macro

File

DisplayDriver.h

C

```
#define GetClipRight _clipRight
```

Side Effects

none

Returns

Right clipping border.

Preconditions

none

Overview

Returns right clipping border.

Syntax

```
GetClipRight()
```

6.4.1.12 GetClipTop Macro

File

DisplayDriver.h

C

```
#define GetClipTop _clipTop
```

Side Effects

none

Returns

Top clipping border.

Preconditions

none

Overview

Returns top clipping border.

Syntax

```
GetClipTop()
```

6.4.1.13 CLIP_DISABLE Macro

File

DisplayDriver.h

C

```
#define CLIP_DISABLE 0 // Disables clipping.
```

Description

Disables clipping.

6.4.1.14 CLIP_ENABLE Macro

File

DisplayDriver.h

C

```
#define CLIP_ENABLE 1 // Enables clipping.
```

Description

Enables clipping.

6.4.1.15 TransparentColorEnable Function

File

DisplayDriver.h

C

```
void TransparentColorEnable(
    GFX_COLOR color
);
```

Input Parameters

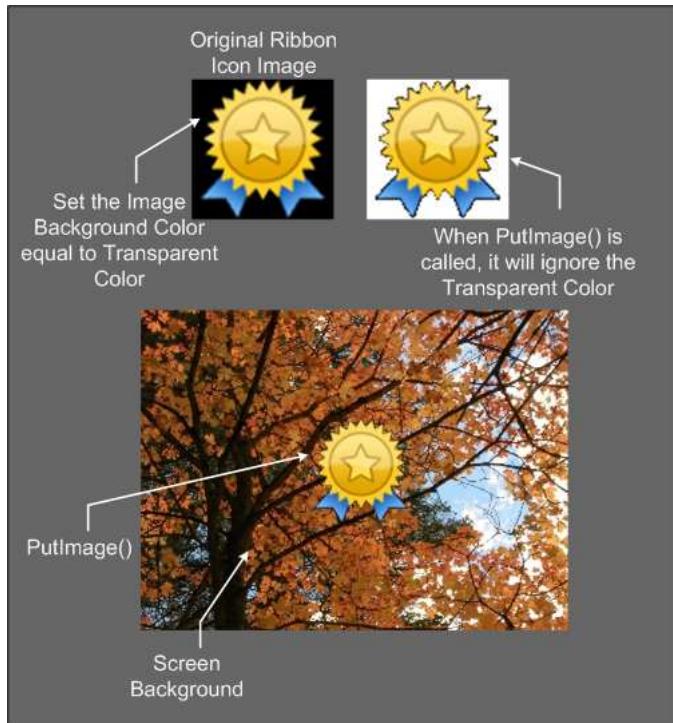
Input Parameters	Description
GFX_COLOR color	Chosen transparent color.

Side Effects

None

Returns

none

Description**Preconditions**

none

Example

```
TransparentColorEnable(BLACK);
PutImage(0,0, (void*)&ScreenBackground);
```

```
PutImage(0,0, (void*)&RibbonIcon);
```

Overview

Sets current transparent color. PutImage () will not render pixels that matches the set transparent color. To enable Transparent Color feature, define the macro USE_TRANSPARENT_COLOR () in the GraphicsConfig.h file.

Syntax

```
void TransparentColorEnable(GFX_COLOR color)
```

6.4.1.16 TransparentColorDisable Macro

File

DisplayDriver.h

C

```
#define TransparentColorDisable _colorTransparentEnable = TRANSPARENT_COLOR_DISABLE;
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Disables the transparent color function.

Syntax

```
TransparentColorDisable()
```

6.4.1.17 GetTransparentColorStatus Macro

File

DisplayDriver.h

C

```
#define GetTransparentColorStatus _colorTransparentEnable
```

Side Effects

None

Returns

Returns the current transparent color function enable status

0 - Transparent color function is disabled.
1 - Transparent color function is enabled.

Preconditions

none

Overview

Returns the current transparent color function enable status.

Syntax

```
GetTransparentColorStatus()
```

6.4.1.18 GetTransparentColor Macro

File

DisplayDriver.h

C

```
#define GetTransparentColor _colorTransparent
```

Side Effects

none

Returns

Returns the current transparent color used.

Preconditions

none

Overview

Returns the current transparent color value.

Syntax

```
GetTransparentColor()
```

6.4.1.19 TRANSPARENT_COLOR_DISABLE Macro

File

DisplayDriver.h

C

```
#define TRANSPARENT_COLOR_DISABLE 0 // Check of transparent color is not performed
```

Description

Check of transparent color is not performed

6.4.1.20 TRANSPARENT_COLOR_ENABLE Macro

File

DisplayDriver.h

C

```
#define TRANSPARENT_COLOR_ENABLE 1 // Check pixel if color is equal to transparent color,  
if equal do not render pixel
```

Description

Check pixel if color is equal to transparent color, if equal do not render pixel

6.4.1.21 DisplayBrightness Function

File

SSD1926.h

C

```
void DisplayBrightness(
    WORD level
);
```

Input Parameters

Input Parameters	Description
WORD level	Brightness level. Valid values are 0 to 100. <ul style="list-style-type: none"> • 0: brightness level is zero or display is turned off • 1: brightness level is maximum

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

Sets the brightness of the display.

Syntax

```
void DisplayBrightness(WORD level)
```

6.4.1.22 GetPageAddress Macro

File

DisplayDriver.h

C

```
#define GetPageAddress(page) (_PageTable[page])
```

Side Effects

none

Returns

Address in memory of the given page number. The number of pages is dictated by GFX_DRV_PAGE_COUNT value defined in the hardware profile. GFX_DRV_PAGE_COUNT is not mandatory. Drivers that do not have enough memory for paging may not define this constant. If GFX_DRV_PAGE_COUNT is not defined, all API's related to paging operation is will not be available.

Preconditions

none

Overview

Returns the address of the given page.

Syntax

```
GetPageAddress(page)
```

6.4.1.23 CopyBlock Function

File

DisplayDriver.h

C

```
WORD CopyBlock(
    DWORD srcAddr,
    DWORD dstAddr,
    DWORD srcOffset,
    DWORD dstOffset,
    WORD width,
    WORD height
);
```

Input Parameters

Input Parameters	Description
DWORD srcAddr	the base address of the data to be moved
DWORD dstAddr	the base address of the new location of the moved data
DWORD srcOffset	offset of the data to be moved with respect to the source base address.
DWORD dstOffset	offset of the new location of the moved data respect to the source base address.
WORD width	width of the block of data to be moved
WORD height	height of the block of data to be moved

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

Copies a block of data from source specified by srcAddr and srcOffset to the destination specified by dstAddr and dstOffset. This is similar to the CopyWindow (█)() and but instead of using left, top position of the source and destination, it uses offsets instead. This is a blocking function.

Syntax

```
WORD CopyBlock(DWORD srcAddr, DWORD dstAddr, DWORD srcOffset, DWORD dstOffset, WORD width, WORD height)
```

6.4.1.24 CopyPageWindow Function

File

DisplayDriver.h

C

```
void CopyPageWindow(
    BYTE srcPage,
    BYTE dstPage,
    \ WORD srcX,
    WORD srcY,
    \ WORD dstX,
```

```
WORD dstY,  
\\ WORD width,  
WORD height  
) ;
```

Input Parameters

Input Parameters	Description
BYTE srcPage	page number of the source window,
BYTE dstPage	page number of the destination window,
\ WORD srcX	x location of the left top corner of the source window respect to the srcPage.
WORD srcY	y location of the left top corner of the source window respect to the srcPage.
\ WORD dstX	x location of the left top corner of the destination window respect to the dstPage.
WORD dstY	y location of the left top corner of the destination window respect to the dstPage.
\ WORD width	the width in pixels of the window to copy
WORD height	the height in pixels of the window to copy

Side Effects

none

Returns

None

Preconditions

none

Overview

This is a blocking call. A windows is a rectangular area with the given width and height of a page. The source and destination window can be located in different pages and each page is assumed to have the same dimensions (equal width and height).

Syntax

```
CopyPageWindow(srcPage, dstPage, srcX, srcY, dstX, dstY, width, height)
```

6.4.1.25 CopyWindow Function

File

DisplayDriver.h

C

```
WORD CopyWindow(  
    DWORD srcAddr,  
    DWORD dstAddr,  
    \\ WORD srcX,  
    WORD srcY,  
    \\ WORD dstX,  
    WORD dstY,  
    \\ WORD width,  
    WORD height  
) ;
```

Input Parameters

Input Parameters	Description
DWORD srcAddr	Base Address of the source window,
\ WORD srcX	x location of the left top corner of the source window respect to the srcPage.
WORD srcY	y location of the left top corner of the source window respect to the srcPage.
\ WORD dstX	x location of the left top corner of the destination window respect to the dstPage.
WORD dstY	y location of the left top corner of the destination window respect to the dstPage.

\ WORD width	the width in pixels of the window to copy
WORD height	the height in pixels of the window to copy
dstPage	Base Address of the destination window,

Side Effects

none

Returns

None

Preconditions

none

Overview

A windows is a rectangular area with the given width and height located in the given base source address. The source and destination window can be located in the same base address. If this is the case, then srcAddr = dstAddr. The operation is similar to CopyPageWindow (图)() but instead of using page numbers, addresses are used for versatility. This is a blocking function.

Syntax

```
WORD CopyWindow( DWORD srcAddr, DWORD dstAddr, WORD srcX, WORD srcY, WORD dstX, WORD dstY, WORD width, WORD height)
```

6.4.1.26 SetActivePage Function

File

DisplayDriver.h

C

```
void SetActivePage(
    WORD page
);
```

Input Parameters

Input Parameters	Description
WORD page	Graphic page number.

Side Effects

none

Returns

none

Description

Functions: SetActivePage(page)

Preconditions

none

Overview

Sets active graphic page.

6.4.1.27 SetVisualPage Function

File

DisplayDriver.h

C

```
void SetVisualPage(
    WORD page
);
```

Input Parameters

Input Parameters	Description
WORD page	Graphic page number

Side Effects

none

Returns

none

Description

Functions: SetVisualPage(page)

Preconditions

none

Overview

Sets graphic page to display.

6.4.1.28 Color Definition

The device driver must also implement the definition of standard color set based on the color format used.

Macros

Name	Description
BLACK (█)	not USE_PALETTE (█)
BLUE (█)	This is macro BLUE.
BRIGHTBLUE (█)	This is macro BRIGHTBLUE.
BRIGHTCYAN (█)	This is macro BRIGHTCYAN.
BRIGHTGREEN (█)	This is macro BRIGHTGREEN.
BRIGHTMAGENTA (█)	This is macro BRIGHTMAGENTA.
BRIGHTRED (█)	This is macro BRIGHTRED.
BRIGHTYELLOW (█)	This is macro BRIGHTYELLOW.
BROWN (█)	This is macro BROWN.
CYAN (█)	This is macro CYAN.
DARKGRAY (█)	This is macro DARKGRAY.
GRAY0 (█)	This is macro GRAY0.
GRAY1 (█)	This is macro GRAY1.
GRAY2 (█)	This is macro GRAY2.
GRAY3 (█)	This is macro GRAY3.
GRAY4 (█)	This is macro GRAY4.
GRAY5 (█)	This is macro GRAY5.

GRAY6 (█)	This is macro GRAY6.
GREEN (█)	This is macro GREEN.
LIGHTBLUE (█)	This is macro LIGHTBLUE.
LIGHTCYAN (█)	This is macro LIGHTCYAN.
LIGHTGRAY (█)	This is macro LIGHTGRAY.
LIGHTGREEN (█)	This is macro LIGHTGREEN.
LIGHTMAGENTA (█)	This is macro LIGHTMAGENTA.
LIGHTRED (█)	This is macro LIGHTRED.
MAGENTA (█)	This is macro MAGENTA.
RED (█)	This is macro RED.
WHITE (█)	This is macro WHITE.
YELLOW (█)	This is macro YELLOW.

Description

The following lists sample color definitions for a 16-bpp color encoding.

6.4.1.28.1 BLACK Macro

File

gfxcolors.h

C

```
#define BLACK RGBConvert(0, 0, 0)
```

Description

not USE_PALETTE (█)

6.4.1.28.2 BLUE Macro

File

gfxcolors.h

C

```
#define BLUE RGBConvert(0, 0, 128)
```

Description

This is macro BLUE.

6.4.1.28.3 BRIGHTBLUE Macro

File

gfxcolors.h

C

```
#define BRIGHTBLUE RGBConvert(0, 0, 255)
```

Description

This is macro BRIGHTBLUE.

6.4.1.28.4 BRIGHTCYAN Macro

File

gfxcolors.h

C

```
#define BRIGHTCYAN RGBConvert(0, 255, 255)
```

Description

This is macro BRIGHTCYAN.

6.4.1.28.5 BRIGHTGREEN Macro

File

```
gfxcolors.h
```

C

```
#define BRIGHTGREEN RGBConvert(0, 255, 0)
```

Description

This is macro BRIGHTGREEN.

6.4.1.28.6 BRIGHTMAGENTA Macro

File

```
gfxcolors.h
```

C

```
#define BRIGHTMAGENTA RGBConvert(255, 0, 255)
```

Description

This is macro BRIGHTMAGENTA.

6.4.1.28.7 BRIGHTRED Macro

File

```
gfxcolors.h
```

C

```
#define BRIGHTRED RGBConvert(255, 0, 0)
```

Description

This is macro BRIGHTRED.

6.4.1.28.8 BRIGHTYELLOW Macro

File

```
gfxcolors.h
```

C

```
#define BRIGHTYELLOW RGBConvert(255, 255, 0)
```

Description

This is macro BRIGHTYELLOW.

6.4.1.28.9 BROWN Macro

File

```
gfxcolors.h
```

C

```
#define BROWN RGBConvert(255, 128, 0)
```

Description

This is macro BROWN.

6.4.1.28.10 CYAN Macro

File

gfxcolors.h

C

```
#define CYAN RGBConvert(0, 128, 128)
```

Description

This is macro CYAN.

6.4.1.28.11 DARKGRAY Macro

File

gfxcolors.h

C

```
#define DARKGRAY RGBConvert(64, 64, 64)
```

Description

This is macro DARKGRAY.

6.4.1.28.12 GRAY0 Macro

File

gfxcolors.h

C

```
#define GRAY0 RGBConvert(224, 224, 224)
```

Description

This is macro GRAY0.

6.4.1.28.13 GRAY1 Macro

File

gfxcolors.h

C

```
#define GRAY1 RGBConvert(192, 192, 192)
```

Description

This is macro GRAY1.

6.4.1.28.14 GRAY2 Macro

File

gfxcolors.h

C

```
#define GRAY2 RGBConvert(160, 160, 160)
```

Description

This is macro GRAY2.

6.4.1.28.15 GRAY3 Macro

File

```
gfxcolors.h
```

C

```
#define GRAY3 RGBConvert(128, 128, 128)
```

Description

This is macro GRAY3.

6.4.1.28.16 GRAY4 Macro

File

```
gfxcolors.h
```

C

```
#define GRAY4 RGBConvert(96, 96, 96)
```

Description

This is macro GRAY4.

6.4.1.28.17 GRAY5 Macro

File

```
gfxcolors.h
```

C

```
#define GRAY5 RGBConvert(64, 64, 64)
```

Description

This is macro GRAY5.

6.4.1.28.18 GRAY6 Macro

File

```
gfxcolors.h
```

C

```
#define GRAY6 RGBConvert(32, 32, 32)
```

Description

This is macro GRAY6.

6.4.1.28.19 GREEN Macro

File

```
gfxcolors.h
```

C

```
#define GREEN RGBConvert(0, 128, 0)
```

Description

This is macro GREEN.

6.4.1.28.20 LIGHTBLUE Macro

File

```
gfxcolors.h
```

C

```
#define LIGHTBLUE RGBConvert(128, 128, 255)
```

Description

This is macro LIGHTBLUE.

6.4.1.28.21 LIGHTCYAN Macro

File

```
gfxcolors.h
```

C

```
#define LIGHTCYAN RGBConvert(128, 255, 255)
```

Description

This is macro LIGHTCYAN.

6.4.1.28.22 LIGHTGRAY Macro

File

```
gfxcolors.h
```

C

```
#define LIGHTGRAY RGBConvert(128, 128, 128)
```

Description

This is macro LIGHTGRAY.

6.4.1.28.23 LIGHTGREEN Macro

File

```
gfxcolors.h
```

C

```
#define LIGHTGREEN RGBConvert(128, 255, 128)
```

Description

This is macro LIGHTGREEN.

6.4.1.28.24 LIGHTMAGENTA Macro

File

```
gfxcolors.h
```

C

```
#define LIGHTMAGENTA RGBConvert(255, 128, 255)
```

Description

This is macro LIGHTMAGENTA.

6.4.1.28.25 LIGHTRED Macro

File

```
gfxcolors.h
```

C

```
#define LIGHTRED RGBConvert(255, 128, 128)
```

Description

This is macro LIGHTRED.

6.4.1.28.26 MAGENTA Macro

File

```
gfxcolors.h
```

C

```
#define MAGENTA RGBConvert(128, 0, 128)
```

Description

This is macro MAGENTA.

6.4.1.28.27 RED Macro

File

```
gfxcolors.h
```

C

```
#define RED RGBConvert(128, 0, 0)
```

Description

This is macro RED.

6.4.1.28.28 WHITE Macro

File

```
gfxcolors.h
```

C

```
#define WHITE RGBConvert(255, 255, 255)
```

Description

This is macro WHITE.

6.4.1.28.29 YELLOW Macro

File

```
gfxcolors.h
```

C

```
#define YELLOW RGBConvert(255, 255, 128)
```

Description

This is macro YELLOW.

6.4.2 Display Device Driver Control

This lists the device control functions and macros.

Functions

	Name	Description
≡	IsDeviceBusy ()	Returns non-zero if LCD controller is busy (previous drawing operation is not completed).
≡	ResetDevice ()	Initializes LCD module.

6.4.2.1 IsDeviceBusy Function

File

DisplayDriver.h

C

```
WORD IsDeviceBusy();
```

Side Effects

none

Returns

Busy status.

Preconditions

none

Overview

Returns non-zero if LCD controller is busy (previous drawing operation is not completed).

Syntax

```
IsDeviceBusy()
```

6.4.2.2 ResetDevice Function

File

DisplayDriver.h

C

```
void ResetDevice();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Initializes LCD module.

Syntax

```
void ResetDevice()
```

6.4.3 Advanced Display Driver Features

This section lists advanced Display Device Driver Features implemented in select Display Device Driver.

6.4.3.1 Alpha Blending

The following APIs are used to implement alpha blending features in the Epson S1D13517 Controller.

Functions

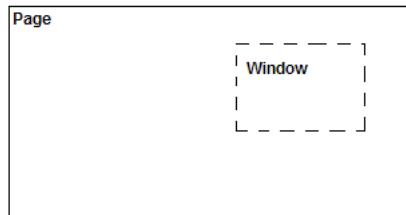
	Name	Description
✳	GFXGetPageOriginAddress (🔗)	This function calculates the address of a certain 0,0 location of a page in memory
✳	GFXGetPageXYAddress (🔗)	This function calculates the address of a certain x,y location in memory

Module

Advanced Display Driver Features (🔗)

Description

Alpha Blending in Epson Controller uses blocks of pixels in the memory called Windows. A window can be any rectangular area in a display buffer. The display buffer is also considered as a page. For a QVGA display buffer a page would be 320x240 pixels. A window is a certain width and height contained inside a page.

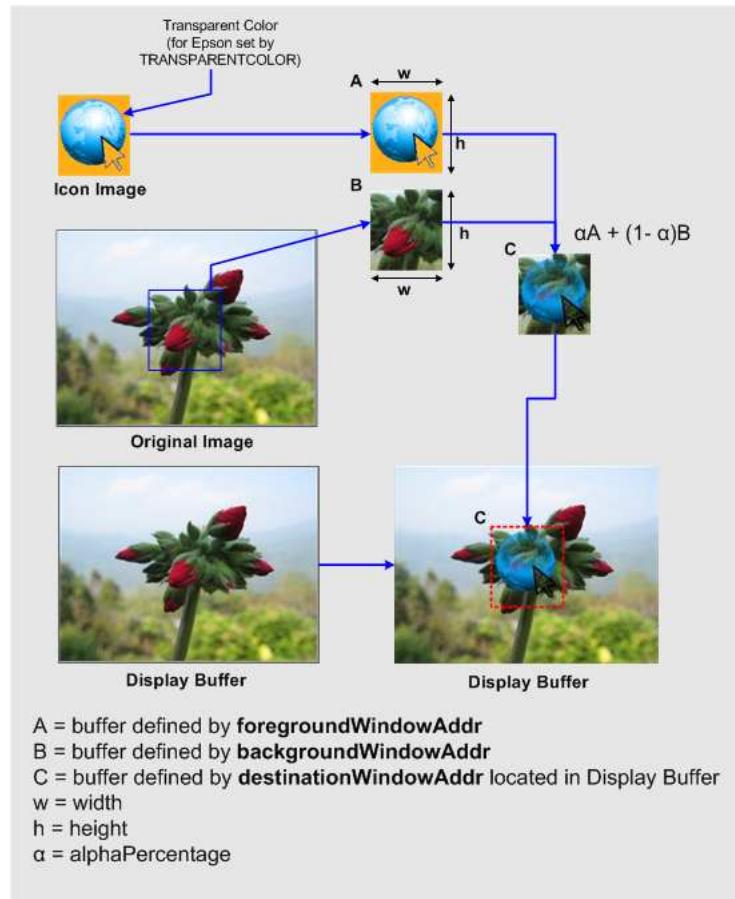


Alpha blending equation:

$$\text{OutPut Window} (\square) = \text{Foreground Window} (\square) * (\text{Alpha}) + \text{Background Window} (\square) * (1-\text{Alpha})$$

when done in software requires a lot of CPU bandwidth. Epson Controller implements alpha blending in hardware which offloads the CPU.

The Display Driver Layer of the Graphics Library utilizes the Epson Controller alpha blending through the AlphaBlendWindow (🔗) function. Three windows are specified with equal widths and heights. An alpha parameter is passed to define the level of alpha blending. The logical flow of the operation is shown below:



An icon image with a given width (w) and height (h) is needed to be alpha blended into the display buffer.

- Buffer A is allocated in memory and its location set by `foregroundWindowAddr`
- Buffer B is allocated in memory and its location set by `backgroundWindowAddr` and
- Buffer C is allocated in memory and its location set by `destinationWindowAddr`. Note that the destination window can be located within the display buffer. Doing this removes an intermediate step after `AlphaBlendWindow` (■) call to put the result of alpha blend in the display buffer.

The final location of the icon on the Display Buffer is used to locate the affected pixels in the Display Buffer. These affected pixels are copied into Buffer B while Buffer A is filled up with the Icon Image. Once Buffers A and B are ready they are alpha blended to Buffer C. Buffer C is then copied to the Display Buffer.

Note that the Icon Image has a background color of ORANGE. To have the effect of the final output Display Buffer, set the Epson Controllers transparent color to ORANGE. This is set in the `TRANSPARENTCOLOR` macro defined in the Epson S1D13517 Driver. This `TRANSPARENTCOLOR` macro is not to be confused with the `TransparentColorEnable` (■) function of the Display Device Driver Level Primitives. `TRANSPARENTCOLOR` is set at build time. In the future release of the Epson driver, this will be converted to use the `TransparentColorEnable` (■) function.

6.4.3.1.1 `GFXGetPageOriginAddress` Function

File

Primitive.h

C

```
DWORD GFXGetPageOriginAddress (
    SHORT pageNumber
);
```

Input Parameters

Input Parameters	Description
SHORT pageNumber	the page number

Side Effects

none

Returns

The address of the start of a certain page in memory

Notes

none

Preconditions

none

Overview

This function calculates the address of a certain 0,0 location of a page in memory

Syntax

```
DWORD GFXGetPageOriginAddress(SHORT pageNumber)
```

6.4.3.1.2 GFXGetPageXYAddress Function

File

Primitive.h

C

```
DWORD GFXGetPageXYAddress(
    SHORT pageNumber,
    WORD x,
    WORD y
);
```

Input Parameters

Input Parameters	Description
SHORT pageNumber	the page number
WORD x	the x (horizontal) offset from 0,0 of the pagenumber
WORD y	the y (vertical) offset from the 0,0 of the pagenumber

Side Effects

none

Returns

The address of an XY position of a certain page in memory

Notes

none

Preconditions

none

Overview

This function calculates the address of a certain x,y location in memory

Syntax

```
DWORD GFXGetPageXYAddress(SHORT pageNumber, WORD x, WORD y)
```

6.4.3.2 Transitions

This section describes screen transition effect when changing screens.

Enumerations

Name	Description
GFX_TRANSITION_DIRECTION (🔗)	Direction enumeration to determine the direction of the selected GFX_TRANSITION_TYPE (🔗).
GFX_TRANSITION_TYPE (🔗)	Transition type enumeration to determine the type of the transition. Each type will require specific parameters when setting up the transition operation (GFXSetupTransition (🔗)) or GFXTransition (🔗)).

Functions

	Name	Description
💡	GFXTransition (🔗)	This immediately executes the transition effect using the GFX_TRANSITION_TYPE (🔗) and the given parameters.
💡	GFXSetupTransition (🔗)	This sets up the transition effect using the GFX_TRANSITION_TYPE (🔗) and the given parameters. The actual transition execution will occur when GFXExecutePendingTransition (🔗)() is called. When DOUBLE_BUFFERING is enabled, GFXExecutePendingTransition (🔗)() is executed after the current screen is fully rendered.
💡	GFXExecutePendingTransition (🔗)	This function executes the transition that was set up by GFXSetupTransition (🔗)(). Status of the transition is returned to indicate if the transition was executed or not. This function is used by the double buffering feature (USE_DOUBLE_BUFFERING (🔗)) to perform transition operation after the current screen is fully rendered. This function assumes that the source page and destination page are already set up.
💡	GFXIsTransitionPending (🔗)	This function returns the status of a pending transition, set up by GFXSetupTransition (🔗)(), waiting to be executed.

Module

Advanced Display Driver Features ([🔗](#))

Description

Applications often require some transition effects while changing screens in order to enhance the look and feel which can be achieved by using Transition APIs provided with the Microchip Graphics Library. Refer to Appendix C of the application note [AN1368: Developing Embedded Graphics Applications using PIC® Microcontrollers with Integrated Graphics Controller](#) for a graphical illustration of how transitions can be achieved.

In order to translate a new screen, the new screen has to be developed in a separate buffer and has to be copied to the frame buffer (visible display buffer) part by part. The way the new screen data is copied into the frame buffer determines the transition effect. A number of transition effects can be achieved by using the API: GFXTransition ([🔗](#))(left, top, right, bottom, type, srcpageaddr, destpageaddr, delay_ms, param1, param2). This API copies the rectangular area defined by left, top, right and bottom from address defined by srcpageaddr to the destpageaddr as per the transition defined by type, param1 and param2. The srcpageaddr should contain the new screen data and the destpageaddr must be the frame buffer address. The speed of the transition can be configured by the parameter delay_ms which determines the delay between each copy operations. Note that this API must be called only at the end of GOLDrawCallback ([🔗](#))() function in order to ensure that the new screen is fully created in the RAM.

If double buffering is enabled, then using transitions is made easier by another API: GFXSetupTransition ([🔗](#))(left, top, right, bottom, type, delay_ms, param1, param2). This API can be called immediately after creating the new screen and the graphics library will store the request and initiate the transition after the new screen is fully created in the RAM. Note that this API doesn't have address parameters as the address of draw-buffer is already known to the double buffering subsystem of

the graphics engine.

6.4.3.2.1 GFXTransition Function

File

Transitions.h

C

```
BYTE GFXTransition(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    GFX_TRANSITION_TYPE type,
    DWORD srcpageaddr,
    DWORD destpageaddr,
    WORD delay_ms,
    WORD param1,
    WORD param2
);
```

Input Parameters

Input Parameters	Description
SHORT left	left x coordinate
SHORT top	top y coordinate
SHORT right	right x coordinate
SHORT bottom	bottom y coordinate
GFX_TRANSITION_TYPE type	Transition type
DWORD srcpageaddr	Source page address for the transition
DWORD destpageaddr	Destination page address for the transition
WORD delay_ms	Delay in milli seconds between redraws in the screen while executing the transition
WORD param1	Transition-type specific parameter
WORD param2	Transition-type specific parameter

Returns

Returns status of transition

- 0 : Transition executed successfully
- -1 : Transition not executed

Overview

This immediately executes the transition effect using the GFX_TRANSITION_TYPE (■) and the given parameters.

Syntax

```
BYTE GFXTransition(SHORT left, SHORT top, SHORT right, SHORT bottom, GFX_TRANSITION_TYPE (■) type, DWORD
srcpageaddr, DWORD destpageaddr, WORD delay_ms, WORD param1, WORD param2)
```

6.4.3.2.2 GFXSetupTransition Function

File

Transitions.h

C

```
BYTE GFXSetupTransition(
    SHORT left,
    SHORT top,
    SHORT right,
```

```

    SHORT bottom,
    GFX_TRANSITION_TYPE type,
    WORD delay_ms,
    WORD param1,
    WORD param2
);

```

Input Parameters

Input Parameters	Description
SHORT left	left x coordinate
SHORT top	top y coordinate
SHORT right	right x coordinate
SHORT bottom	bottom y coordinate
GFX_TRANSITION_TYPE type	Transition type
WORD delay_ms	Delay in milli seconds between redraws in the screen while executing the transition
WORD param1	Transition-type specific parameter
WORD param2	Transition-type specific parameter

Returns

Returns success of the setup

- 0 : Parameters successfully saved for the new transition
- -1 : Parameters not saved, there is a pending transition

Overview

This sets up the transition effect using the GFX_TRANSITION_TYPE (§) and the given parameters. The actual transition execution will occur when GFXExecutePendingTransition (§)() is called. When DOUBLE_BUFFERING is enabled, GFXExecutePendingTransition (§)() is executed after the current screen is fully rendered.

Syntax

```
BYTE GFXSetupTransition(SHORT left, SHORT top, SHORT right, SHORT bottom, GFX_TRANSITION_TYPE (§) type,
WORD delay_ms, WORD param1, WORD param2)
```

6.4.3.2.3 GFXExecutePendingTransition Function**File**

Transitions.h

C

```

BYTE GFXExecutePendingTransition(
    DWORD srcpageaddr,
    DWORD destpageaddr
);

```

Input Parameters

Input Parameters	Description
DWORD srcpageaddr	Source page address for the transition
DWORD destpageaddr	Destination page address for the transition

Returns

Returns status of transition

- 0 : Transition executed successfully
- -1 : Transition not executed

Overview

This function executes the transition that was set up by GFXSetupTransition (§)(). Status of the transition is returned to

indicate if the transition was executed or not. This function is used by the double buffering feature (USE_DOUBLE_BUFFERING ()) to perform transition operation after the current screen is fully rendered. This function assumes that the source page and destination page are already set up.

Syntax

```
BYTE GFXExecutePendingTransition(DWORD srcpageaddr, DWORD destpageaddr)
```

6.4.3.2.4 GFXIsTransitionPending Function

File

Transitions.h

C

```
BYTE GFXIsTransitionPending();
```

Returns

Returns transition status

- 0 : No pending transition
- 1 : There is a pending transition

Overview

This function returns the status of a pending transition, set up by GFXSetupTransition ()(), waiting to be executed.

Syntax

```
BYTE GFXIsTransitionPending(void)
```

6.4.3.2.5 GFX_TRANSITION_DIRECTION Enumeration

File

Transitions.h

C

```
typedef enum {
    LEFT_TO_RIGHT = 0,
    RIGHT_TO_LEFT,
    TOP_TO_BOTTOM,
    BOTTOM_TO_TOP,
    HORIZONTAL,
    VERTICAL
} GFX_TRANSITION_DIRECTION;
```

Members

Members	Description
LEFT_TO_RIGHT = 0	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE ())
RIGHT_TO_LEFT	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE ())
TOP_TO_BOTTOM	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE ())
BOTTOM_TO_TOP	option used in SLIDE, PUSH transition type (GFX_TRANSITION_TYPE ())
HORIZONTAL	option used in EXPANDING_LINE and CONTRACTING_LINE transition type (GFX_TRANSITION_TYPE ())
VERTICAL	option used in EXPANDING_LINE and CONTRACTING_LINE transition type (GFX_TRANSITION_TYPE ())

Overview

Direction enumeration to determine the direction of the selected GFX_TRANSITION_TYPE ().

6.4.3.2.6 GFX_TRANSITION_TYPE Enumeration

File

Transitions.h

C

```
typedef enum {
    PLAIN = 0,
    SLIDE,
    PUSH,
    EXPANDING_RECTANGLE,
    CONTRACTING_RECTANGLE,
    EXPANDING_LINE,
    CONTRACTING_LINE
} GFX_TRANSITION_TYPE;
```

Members

Members	Description
PLAIN = 0	no transition effect
SLIDE	param1 -> Pixel-block size, param2 -> Sliding direction LEFT_TO_RIGHT/RIGHT_TO_LEFT/TOP_TO_BOTTOM/BOTTOM_TO_TOP
PUSH	param1 -> Pixel-block size, param2 -> Sliding direction LEFT_TO_RIGHT/RIGHT_TO_LEFT/TOP_TO_BOTTOM/BOTTOM_TO_TOP
EXPANDING_RECTANGLE	param1 -> Pixel-block size
CONTRACTING_RECTANGLE	param1 -> Pixel-block size
EXPANDING_LINE	param1 -> Pixel-block size, param2 -> direction HORIZONTAL/VERTICAL
CONTRACTING_LINE	param1 -> Pixel-block size, param2 -> direction HORIZONTAL/VERTICAL

Overview

Transition type enumeration to determine the type of the transition. Each type will require specific parameters when setting up the transition operation (GFXSetupTransition (¶)() or GFXTransition (¶)()).

GFX_TRANSITION_TYPE	param1	param2 (sets the direction of transition)
PLAIN	pixel block size	not used
EXPANDING_RECTANGLE	pixel block size	not used
CONTRACTING_RECTANGLE	pixel block size	not used
SLIDE	pixel block size	LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP
PUSH	pixel block size	LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP
EXPANDING_LINE	pixel block size	HORIZONTAL, VERTICAL
CONTRACTING_LINE	pixel block size	HORIZONTAL, VERTICAL

6.4.3.3 Double Buffering

In the Microchip Graphics Library, if double-buffering is enabled, the frame buffer and draw buffer are exchanged after the changes of all the widgets on a screen are done (i.e., the new screen appears after the whole screen is updated and not after updating an individual widget). This feature is enabled only on the following drivers:

- Microchip Graphics Display Driver - customizable driver for RGB Glass. Currently used in PIC24FJ256DA210 device family.
- Microchip Low-Cost Controllerless (LCC) Graphics Display Driver - customizable driver for RGB Glass. Currently used for

selected PIC32MX device families.

Functions

Name	Description
SwitchOffDoubleBuffering	Switches off the double buffering. All rendering will be performed on the frame buffer. Calls to UpdateDisplayNow () or RequestDisplayUpdate () will have no effect.
SwitchOnDoubleBuffering	Switches on the double buffering. Double buffering utilizes two buffers. The frame buffer and the draw buffer. The frame buffer is the buffer that is being displayed while the draw buffer is used for all rendering. When this function is called, it copies the contents of the frame buffer to the draw buffer once and all succeeding rendering will be performed on the draw buffer. To update the frame buffer with newly drawn items on the draw buffer call UpdateDisplayNow () or RequestDisplayUpdate ().
InvalidateRectangle	Invalidates the specified rectangular area. This increments the number of invalidated areas and if the number of invalidated areas exceed the <code>GFX_MAX_INVALIDATE AREAS</code> , the whole frame buffer is invalidated.
RequestDisplayUpdate	Synchronizes the draw and frame buffers at next VBlank
UpdateDisplayNow	Synchronizes the draw and frame buffers immediately.

Module

[Advanced Display Driver Features](#)

Description

Manipulating pixels on the screen requires direct writes to the frame buffer. While these changes are being executed, the screen is also refreshed. This means that the changes are displayed immediately as the frame buffer is being updated. This is not a suitable scheme when the changes are slow, for example, decoding an image or having a large number of widgets on a screen. The display may appear slow in that case. One solution to this problem is to use a double-buffering scheme supported by the Microchip Graphics Library. In this scheme, the changes are not directly written to the frame buffer, but instead, they are written to a separate buffer, called the 'Draw Buffer'. After all the changes are made, the draw buffer and frame buffer are exchanged. Now the draw buffer becomes the frame buffer, and because of all the changes drawn there, the changes appear spontaneous to the user. Of course, there will be a delay, as all the changes have to be written to the draw buffer before displaying it. This delay is generally more tolerable than displaying the changes slowly. After exchanging of the buffers, the latest buffer (which is now the frame buffer) is copied to the new draw buffer in the background and then the new draw buffer is in sync with what is being displayed. New changes are then written to the draw buffer and the cycle continues. As the double-buffering scheme uses two buffers, the RAM requirement will double.

In the Microchip Graphics Library, if double-buffering is enabled, the frame buffer and draw buffer are exchanged after the changes of all the widgets on a screen are done (i.e., the new screen appears after the whole screen is updated and not after updating an individual widget).

The work flow of double-buffering is graphically explained along with tips on deciding when to use double buffering in the APPENDIX B of the Application note AN1368: Developing Embedded Graphics Applications using PIC® Microcontrollers with Integrated Graphics Controller.

To use double buffering in an application, follow the steps described below:

1. Enable the option `USE_DOUBLE_BUFFERING` (<#>) in `GraphicsConfig.h`
2. Update `GFX_DISPLAY_BUFFER_LENGTH` (<#>) to include both the buffers in `HardwareProfile.h`. Note that when using external memory the `GFX_DISPLAY_BUFFER_LENGTH` (<#>) must fit into the external memory size. For example in PIC24FJ256DA210 external memory can be added using EPMP. External memory mapped to the EPMP must be big enough to accommodate the display buffers required by double buffering. See [Set Up Display Interface](#) (<#>) for more

information on memory requirements.

3. Check the jumper settings to enable the required RAM address space on the development board

If Graphics Objects Layer (GOL) is used in the application, the switching of buffers is handled automatically in order to keep the switching task transparent to the users. If double buffering is enabled in applications using only the Primitive layer, then the switching of buffers has to be handled by the application itself as explained in the following steps.

Steps required for manually handling the switching of buffers:

1. After `InitGraph` () is called, call the APIs `InvalidateAll`() followed by `UpdateDisplayNow` (). The two buffers are properly setup after these calls and from this point onwards, the drawing happens on the draw-buffer.
2. When a shape is drawn on the draw buffer, that rectangular area has to be marked as invalid by using the API `InvalidateRectangle` (left, top, right, bottom). Only the invalidated rectangular areas are copied to the frame buffer in order to reduce the copy operations thereby reducing the overall time and energy required to sync the two buffers. Hence, it is important to invalidate the changed rectangular areas failing which the change doesn't show up on the display.
3. Call either `RequestDisplayUpdate` () or `UpdateDisplayNow` () to sync the two buffers and as a result the changes appear on the display. The former API exchanges the buffers during the next display blanking period on TFT LCDs causing the change to appear smooth and immediate whereas the latter API exchanges the two buffers at the time of the API call probably causing a slight flicker on the display. On displays which doesn't support blanking periods (e.g. CSTN LCDs), the effect of `RequestDisplayUpdate` () is same as that of `UpdateDisplayNow` ().

Even if double buffering is enabled at compile time, it can be switched off and on at run time using APIs `SwitchOffDoubleBuffering` () and `SwitchOnDoubleBuffering` (). Switching double buffering on/off at runtime is useful in applications which need some screens having fast updates like waveform or animation which requires double buffering to be switched off and some other screens where double buffering is beneficial.

Note: In applications using Graphics Objects Layer and double buffering, the double buffering is not immediately enabled after the API `GOLInit` () is called in order to support hassles splash screens but is automatically enabled from the second screen update onwards. If double buffering is needed from the first screen itself, then follow step 1 immediately after calling `GOLInit` ()�.

6.4.3.3.1 `SwitchOffDoubleBuffering` Function

File

`DisplayDriver.h`

C

```
void SwitchOffDoubleBuffering();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Switches off the double buffering. All rendering will be performed on the frame buffer. Calls to UpdateDisplayNow (■) or RequestDisplayUpdate (■) will have no effect.

Syntax

SwitchOffDoubleBuffering()

6.4.3.3.2 SwitchOnDoubleBuffering Function

File

DisplayDriver.h

C

```
void SwitchOnDoubleBuffering( );
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Switches on the double buffering. Double buffering utilizes two buffers. The frame buffer and the draw buffer. The frame buffer is the buffer that is being displayed while the draw buffer is used for all rendering. When this function is called, it copies the contents of the frame buffer to the draw buffer once and all succeeding rendering will be performed on the draw buffer. To update the frame buffer with newly drawn items on the draw buffer call UpdateDisplayNow (■) or RequestDisplayUpdate (■).

Syntax

SwitchOnDoubleBuffering()

6.4.3.3.3 InvalidateRectangle Function

File

DisplayDriver.h

C

```
void InvalidateRectangle(
    WORD left,
    WORD top,
    WORD right,
    WORD bottom
);
```

Input Parameters

Input Parameters	Description
WORD left	left position
WORD top	top position
WORD right	right position
WORD bottom	bottom position

Side Effects

Copies back the invalidated areas only to the draw buffer after the exchange of draw and frame buffers.

Returns

None

Preconditions

None

Overview

Invalidate the specified rectangular area. This increments the number of invalidated areas and if the number of invalidated areas exceed the GFX_MAX_INVALIDATE AREAS, the whole frame buffer is invalidated.

Syntax

```
void InvalidateRectangle(WORD left, WORD top, WORD right, WORD bottom)
```

6.4.3.3.4 RequestDisplayUpdate Function

File

DisplayDriver.h

C

```
void RequestDisplayUpdate();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Synchronizes the draw and frame buffers at next VBlank

Syntax

```
void RequestDisplayUpdate(void)
```

6.4.3.3.5 UpdateDisplayNow Function

File

DisplayDriver.h

C

```
void UpdateDisplayNow();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Synchronizes the draw and frame buffers immediately.

Syntax

```
void UpdateDisplayNow(void)
```

6.4.3.4 Microchip Graphics Controller

This is the generic display driver is intended for the Microchip Graphics Controller Module implemented in PIC24F device family. This driver will drive TFT, CSTN and STN displays.

Module

[Advanced Display Driver Features](#) (§)

Description

The Microchip Graphics Controller has several advanced features that can be enabled by declaring macros in the Graphics Config (GraphicsConfig.h) and Hardware Profile (HardwareProfile.h) of the project. Below is a summary of all macros that pertains to the Microchip Graphics Controller driver.

Macro Name	Description	Location
GFX_DISPLAY_BUFFER_START_ADDRESS (§)	Defines the starting address of the display buffer.	Hardware Profile
GFX_DISPLAY_BUFFER_LENGTH (§)	<p>Defines the size of the display buffer. This macro is used to map memory in RAM when:</p> <ol style="list-style-type: none"> 1. Double buffering is enabled. This is enabled by the macro USE_DOUBLE_BUFFERING (§) 2. IPU module is used. This is enabled by the macro USE_COMP_IPU (§) <p>If the application is designed to fit into the internal memory, and with the If the double buffering and/or IPU module is enabled, the memory required must fit into the internal memory.</p> <p>NOTE: If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then GFX_DISPLAY_BUFFER_LENGTH (§) must be less than or equal to the chip select region size</p> <ul style="list-style-type: none"> • GFX_DISPLAY_BUFFER_LENGTH (§) <= GFX_EPMP_CS1_MEMORY_SIZE (§) if located in chip select 1 (CS1) region • GFX_DISPLAY_BUFFER_LENGTH (§) <= GFX_EPMP_CS2_MEMORY_SIZE (§) if located in chip select 2 (CS2) region 	Hardware Profile
USE_PALETTE (§)	Macro that enables the palette mode in Graphics Library and the Microchip Graphics Controller.	Graphics Config

USE_DOUBLE_BUFFERING ([2])	<p>Optional feature. This enables the double buffering feature. Memory required for display buffer will double in size. There will be two display buffers used and these buffers are swapped automatically by the library draw operations or application can manage the swapping. See Double Buffering ([2]) for details.</p> <p>Buffer Location (Mapping):</p> <pre>Display Buffer 1 = GFX_DISPLAY_BUFFER_START_ADDRESS ([2]) Display Buffer 2 = GFX_DISPLAY_BUFFER_START_ADDRESS ([2]) + GFX_DISPLAY_BUFFER_LENGTH ([2])</pre> <p>NOTE: If the display buffers are located in external memory (i.e in PIC24FJ256DA210) then the total display buffer memory needed must be less than or equal to the chip select region size</p> <ul style="list-style-type: none"> • <math>(\text{GFX_DISPLAY_BUFFER_LENGTH} ([2]) * 2) \leq \text{GFX_EPMP_CS1_MEMORY_SIZE} ([2])</math> if located in chip select 1 (CS1) region • <math>(\text{GFX_DISPLAY_BUFFER_LENGTH} ([2]) * 2) \leq \text{GFX_EPMP_CS2_MEMORY_SIZE} ([2])</math> if located in chip select 2 (CS2) region
--	---

6.4.3.4.1 Rectangle Copy Operations

The following APIs are used move blocks of data from one memory location to another.

Functions

	Name	Description
	ROPBlock ([2])	Performs a Raster Operation (ROP) on source and destination. The type of ROP is decided by the rop and the copyOp parameter.
	Scroll ([2])	Scrolls the rectangular area defined by left, top, right, bottom by delta pixels.

Macros

Name	Description
RCC_SRC_ADDR_CONTINUOUS ([2])	Source (S) and Destination (D) data type. When performing executing commands on the Rectangle ([2]) Copy Processing Unit (RCCGPU). The source and destination data can be treated as a continuous block of data in memory or a discontinuous data in memory. This gives flexibility to the operation where an copy operation can be performed to data already present in the display buffer or anywhere else in data memory. Both source and destination data can be set to continuous or discontinuous data. These macros are only used in RCCGPU operations. <ul style="list-style-type: none"> • RCC_SRC_ADDR_CONTINUOUS - source data is continuous • RCC_SRC_ADDR_DISCONTINUOUS ([2]) - source data is discontinuous... more ([2])

RCC_SRC_ADDR_DISCONTINUOUS (🔗)	<p>Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (🔗) Copy Processing Unit (RCCGPU). The source and destination data can be treated as a continuous block of data in memory or a discontinuous data in memory. This gives flexibility to the operation where an copy operation can be performed to data already present in the display buffer or anywhere else in data memory. Both source and destination data can be set to continuous or discontinuous data. These macros are only used in RCCGPU operations.</p> <ul style="list-style-type: none"> • RCC_SRC_ADDR_CONTINUOUS - source data is continuous • RCC_SRC_ADDR_DISCONTINUOUS (🔗) - source data is discontinuous... more (🔗)
RCC_DEST_ADDR_CONTINUOUS (🔗)	<p>Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (🔗) Copy Processing Unit (RCCGPU). The source and destination data can be treated as a continuous block of data in memory or a discontinuous data in memory. This gives flexibility to the operation where an copy operation can be performed to data already present in the display buffer or anywhere else in data memory. Both source and destination data can be set to continuous or discontinuous data. These macros are only used in RCCGPU operations.</p> <ul style="list-style-type: none"> • RCC_SRC_ADDR_CONTINUOUS - source data is continuous • RCC_SRC_ADDR_DISCONTINUOUS (🔗) - source data is discontinuous... more (🔗)
RCC_DEST_ADDR_DISCONTINUOUS (🔗)	<p>Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (🔗) Copy Processing Unit (RCCGPU). The source and destination data can be treated as a continuous block of data in memory or a discontinuous data in memory. This gives flexibility to the operation where an copy operation can be performed to data already present in the display buffer or anywhere else in data memory. Both source and destination data can be set to continuous or discontinuous data. These macros are only used in RCCGPU operations.</p> <ul style="list-style-type: none"> • RCC_SRC_ADDR_CONTINUOUS - source data is continuous • RCC_SRC_ADDR_DISCONTINUOUS (🔗) - source data is discontinuous... more (🔗)
RCC_ROP_0 (🔗)	<p>Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (🔗) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D).</p> <ul style="list-style-type: none"> • RCC_ROP_0 - 0 (BLACK (🔗)) • RCC_ROP_1 (🔗) - not (S or D) • RCC_ROP_2 (🔗) - (not S) and D • RCC_ROP_3 (🔗) - not (S) • RCC_ROP_4 (🔗) - S and not (D)) • RCC_ROP_5 (🔗) - not (D) • RCC_ROP_6 (🔗) - S xor D • RCC_ROP_7 (🔗) - not (S and D) • RCC_ROP_8 (🔗) - S and D • RCC_ROP_9 (🔗) - not (S xor D) • RCC_ROP_A (🔗) - D • RCC_ROP_B (🔗)... more (🔗)

RCC_ROP_1 (■)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (■) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (■))• RCC_ROP_1 (■) - not (S or D)• RCC_ROP_2 (■) - (not S) and D• RCC_ROP_3 (■) - not (S)• RCC_ROP_4 (■) - S and not (D))• RCC_ROP_5 (■) - not (D)• RCC_ROP_6 (■) - S xor D• RCC_ROP_7 (■) - not (S and D)• RCC_ROP_8 (■) - S and D• RCC_ROP_9 (■) - not (S xor D)• RCC_ROP_A (■) - D• RCC_ROP_B (■)... more (■)
RCC_ROP_2 (■)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (■) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (■))• RCC_ROP_1 (■) - not (S or D)• RCC_ROP_2 (■) - (not S) and D• RCC_ROP_3 (■) - not (S)• RCC_ROP_4 (■) - S and not (D))• RCC_ROP_5 (■) - not (D)• RCC_ROP_6 (■) - S xor D• RCC_ROP_7 (■) - not (S and D)• RCC_ROP_8 (■) - S and D• RCC_ROP_9 (■) - not (S xor D)• RCC_ROP_A (■) - D• RCC_ROP_B (■)... more (■)

RCC_ROP_3 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)
RCC_ROP_4 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)

RCC_ROP_5 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)
RCC_ROP_6 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)

RCC_ROP_7 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data. and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)
RCC_ROP_8 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data. and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)

RCC_ROP_9 (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data. and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)
RCC_ROP_A (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data. and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)

RCC_ROP_B (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)
RCC_ROP_C (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)

RCC_ROP_D (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)
RCC_ROP_E (█)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D). <ul style="list-style-type: none">• RCC_ROP_0 - 0 (BLACK (█))• RCC_ROP_1 (█) - not (S or D)• RCC_ROP_2 (█) - (not S) and D• RCC_ROP_3 (█) - not (S)• RCC_ROP_4 (█) - S and not (D))• RCC_ROP_5 (█) - not (D)• RCC_ROP_6 (█) - S xor D• RCC_ROP_7 (█) - not (S and D)• RCC_ROP_8 (█) - S and D• RCC_ROP_9 (█) - not (S xor D)• RCC_ROP_A (█) - D• RCC_ROP_B (█)... more (█)

RCC_ROP_F (🔗)	<p>Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (🔗) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data, and the result written to the destination (D).</p> <ul style="list-style-type: none"> • RCC_ROP_0 - 0 (BLACK (🔗)) • RCC_ROP_1 (🔗) - not (S or D) • RCC_ROP_2 (🔗) - (not S) and D • RCC_ROP_3 (🔗) - not (S) • RCC_ROP_4 (🔗) - S and not (D)) • RCC_ROP_5 (🔗) - not (D) • RCC_ROP_6 (🔗) - S xor D • RCC_ROP_7 (🔗) - not (S and D) • RCC_ROP_8 (🔗) - S and D • RCC_ROP_9 (🔗) - not (S xor D) • RCC_ROP_A (🔗) - D • RCC_ROP_B (🔗)... more (🔗)
RCC_COPY (🔗)	<p>Type of Rectangle (🔗) Copy Operations. Select one of the following rectangle copy operations and together with the ROP; the source, destination, current color set and transparency are evaluated on each pixel and the result written to the destination.</p> <ul style="list-style-type: none"> • RCC_COPY - Copies the source data to the destination address with the selected ROP. • RCC_SOLID_FILL (🔗) - Fills the specified rectangle with the current color set. • RCC_TRANSPARENT_COPY (🔗) - Operation is the same as the COPY operation except that the source data is compared against the current color set. If the values match, the source data is not written to the destination. The source... more (🔗)
RCC_SOLID_FILL (🔗)	<p>Type of Rectangle (🔗) Copy Operations. Select one of the following rectangle copy operations and together with the ROP; the source, destination, current color set and transparency are evaluated on each pixel and the result written to the destination.</p> <ul style="list-style-type: none"> • RCC_COPY - Copies the source data to the destination address with the selected ROP. • RCC_SOLID_FILL (🔗) - Fills the specified rectangle with the current color set. • RCC_TRANSPARENT_COPY (🔗) - Operation is the same as the COPY operation except that the source data is compared against the current color set. If the values match, the source data is not written to the destination. The source... more (🔗)

RCC_TRANSPARENT_COPY (🔗)	Type of Rectangle (🔗) Copy Operations. Select one of the following rectangle copy operations and together with the ROP; the source, destination, current color set and transparency are evaluated on each pixel and the result written to the destination. <ul style="list-style-type: none"> • RCC_COPY - Copies the source data to the destination address with the selected ROP. • RCC_SOLID_FILL (🔗) - Fills the specified rectangle with the current color set. • RCC_TRANSPARENT_COPY (🔗) - Operation is the same as the COPY operation except that the source data is compared against the current color set. If the values match, the source data is not written to the destination. The source... more (🔗)
--	--

6.4.3.4.1.1 ROPBlock Function

File

mchpGfxDrv.h

C

```
WORD ROPBlock(
    DWORD srcAddr,
    DWORD dstAddr,
    DWORD srcOffset,
    DWORD dstOffset,
    WORD srcType,
    WORD dstType,
    WORD copyOp,
    WORD rop,
    WORD color,
    WORD width,
    WORD height
);
```

Input Parameters

Input Parameters	Description
DWORD srcAddr	the base address of the data to be moved
DWORD dstAddr	the base address of the new location of the moved data
DWORD srcOffset	offset of the data to be moved with respect to the source base address.
DWORD dstOffset	offset of the new location of the moved data respect to the source base address.
WORD srcType	sets the source type (GFX_DATA_CONTINUOUS or GFX_DATA_DISCONTINUOUS)
WORD dstType	sets the destination type (GFX_DATA_CONTINUOUS or GFX_DATA_DISCONTINUOUS)
WORD copyOp	sets the type of copy operation <ul style="list-style-type: none"> • RCC_SOLID_FILL (🔗): Solid fill of the set color • RCC_COPY (🔗): direct copy of source to destination • RCC_TRANSPARENT_COPY (🔗): copy with transparency. Transparency color is set by color

WORD rop	sets the raster operation equation <ul style="list-style-type: none"> • RCC_ROP_0 (█): Solid black color fill • RCC_ROP_1 (█): not (Source or Destination) • RCC_ROP_2 (█): (not Source) and Destination • RCC_ROP_3 (█): not Source • RCC_ROP_4 (█): Source and (not Destination) • RCC_ROP_5 (█): not Destination • RCC_ROP_6 (█): Source xor Destination • RCC_ROP_7 (█): not (Source and Destination) • RCC_ROP_8 (█): Source and Destination • RCC_ROP_9 (█): not (Source xor Destination) • RCC_ROP_A (█): Destination • RCC_ROP_B (█): (not Source) or Destination • RCC_ROP_C (█): Source • RCC_ROP_D (█): Source or (not Destination) • RCC_ROP_E (█): Source or Destination • RCC_ROP_F (█): Solid white color fill
WORD color	color value used when transparency operation is set or using solid color fill
WORD width	width of the block of data to be moved
WORD height	height of the block of data to be moved

Side Effects

none

Returns

For NON-Blocking configuration:

- Returns 0 when device is busy and operation is not completely performed.
- Returns 1 when the operation is completely performed.

For Blocking configuration:

- Always return 1.

Notes

none

Preconditions

none

Overview

Performs a Raster Operation (ROP) on source and destination. The type of ROP is decided by the rop and the copyOp parameter.

Syntax

```
WORD ROPBlock (DWORD srcAddr, DWORD dstAddr, DWORD srcOffset, DWORD dstOffset, WORD srcType, WORD dstType, WORD copyOp, WORD rop, WORD color, WORD width, WORD height)
```

6.4.3.4.1.2 Scroll Function**File**

mchpGfxDrv.h

C

```
WORD Scroll(
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    SHORT delta,
    WORD dir
);
```

Input Parameters

Input Parameters	Description
SHORT left	left position of the scrolled rectangle
SHORT top	top position of the scrolled rectangle
SHORT right	right position of the scrolled rectangle
SHORT bottom	bottom position of the scrolled rectangle
SHORT delta	defines the scroll delta
WORD dir	defines the direction of the scroll. <ul style="list-style-type: none"> • 0 : scroll to the left • 1 : scroll to the right

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

Scrolls the rectangular area defined by left, top, right, bottom by delta pixels.

Syntax

```
WORD Scroll(SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT delta)
```

6.4.3.4.1.3 RCC_SRC_ADDR_CONTINUOUS Macro**File**

mchpGfxDrv.h

C

```
#define RCC_SRC_ADDR_CONTINUOUS 0x00000002ul
#define RCC_SRC_ADDR_DISCONTINUOUS 0x00000000ul
#define RCC_DEST_ADDR_CONTINUOUS 0x00000004ul
#define RCC_DEST_ADDR_DISCONTINUOUS 0x00000000ul
```

Overview

Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (█) Copy Processing Unit (RCCGPU). The source and destination data can be treated as a continuous block of data in memory or a discontinuous data in memory. This gives flexibility to the operation where an copy operation can be performed to data already present in the display buffer or anywhere else in data memory. Both source and destination data can be set to continuous or discontinuous data. These macros are only used in RCCGPU operations.

- RCC_SRC_ADDR_CONTINUOUS - source data is continuous
- RCC_SRC_ADDR_DISCONTINUOUS - source data is discontinuous
- RCC_DEST_ADDR_CONTINUOUS - destination data is continuous
- RCC_DEST_ADDR_DISCONTINUOUS - destination data is discontinuous

6.4.3.4.1.4 RCC_ROP_0 Macro

File

mchpGfxDrv.h

C

```
#define RCC_ROP_0 0x00000000ul          // not (S or D)
#define RCC_ROP_1 0x00000008ul          // (not S) and D
#define RCC_ROP_2 0x00000010ul          // not (S)
#define RCC_ROP_3 0x00000018ul          // S and not (D)
#define RCC_ROP_4 0x00000020ul          // not (D)
#define RCC_ROP_5 0x00000028ul          // S xor D
#define RCC_ROP_6 0x00000030ul          // not (S and D)
#define RCC_ROP_7 0x00000038ul          // S and D
#define RCC_ROP_8 0x00000040ul          // not (S xor D)
#define RCC_ROP_9 0x00000048ul          // D
#define RCC_ROP_A 0x00000050ul          // not (S) or D
#define RCC_ROP_B 0x00000058ul          // S
#define RCC_ROP_C 0x00000060ul          // S or not (D)
#define RCC_ROP_D 0x00000068ul          // S or D
#define RCC_ROP_E 0x00000070ul          // 1 (WHITE)
#define RCC_ROP_F 0x00000078ul          // 1 (WHITE)
```

Overview

Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (█) Copy Processing Unit (RCCGPU) is used. The raster operation is performed on the source (S) and destination (D) data. and the result written to the destination (D).

- RCC_ROP_0 - 0 (BLACK (█))
- RCC_ROP_1 - not (S or D)
- RCC_ROP_2 - (not S) and D
- RCC_ROP_3 - not (S)
- RCC_ROP_4 - S and not (D))
- RCC_ROP_5 - not (D)
- RCC_ROP_6 - S xor D
- RCC_ROP_7 - not (S and D)
- RCC_ROP_8 - S and D
- RCC_ROP_9 - not (S xor D)
- RCC_ROP_A - D
- RCC_ROP_B - not (S) or D
- RCC_ROP_C - S
- RCC_ROP_D - S or not (D)
- RCC_ROP_E - S or D
- RCC_ROP_F - 1 (WHITE (█))

6.4.3.4.1.5 RCC_COPY Macro

File

mchpGfxDrv.h

C

```
#define RCC_COPY 0x00000080ul
#define RCC_SOLID_FILL 0x00000000ul
#define RCC_TRANSPARENT_COPY 0x00000300ul
```

Overview

Type of Rectangle (█) Copy Operations. Select one of the following rectangle copy operations and together with the ROP; the source, destination, current color set and transparency are evaluated on each pixel and the result written to the destination.

- RCC_COPY - Copies the source data to the destination address with the selected ROP.
- RCC_SOLID_FILL - Fills the specified rectangle with the current color set.
- RCC_TRANSPARENT_COPY - Operation is the same as the COPY operation except that the source data is compared against the current color set. If the values match, the source data is not written to the destination. The source image is, therefore, transparent at such a location, allowing

6.4.3.4.2 Decompressing DEFLATEd data

The Microchip Graphics Controller features a decompression module for data compressed using the DEFLATE algorithm. Compressed data are limited to fixed huffman codes. Compressed data with dynamic huffman codes are not supported.

Functions

	Name	Description
✳	Decompress (█)	Decompresses the nbytes number of data at SrcAddress and places starting from DestAddress. (Blocking)

6.4.3.4.2.1 Decompress Function**File**

mchpGfxDrv.h

C

```
BYTE Decompress(
    DWORD SrcAddress,
    DWORD DestAddress,
    DWORD nbytes
);
```

Input Parameters

Input Parameters	Description
DWORD SrcAddress	Source address
DWORD DestAddress	Destination address
DWORD nbytes	Number of bytes to be decompressed

Side Effects

Modifies workarea_1 & workarea_2 registers.

Returns

error flag

Preconditions

SrcAddress must point to the start of a compressed block.

Overview

Decompresses the nbytes number of data at SrcAddress and places starting from DestAddress. (Blocking)

Syntax

```
BYTE Decompress(DWORD SrcAddress, DWORD DestAddress, DWORD nbytes);
```

6.4.3.4.3 Palette Mode

The Microchip Graphics Controller features a palette mode for a smaller frame buffer requirement. This option uses the built-in 256 entry Color Look-up Table (CLUT) to represent pixels from the display buffer in memory. If the CLUT is enabled, each pixel in the display buffer is assumed to contain the color index. This color index is used as the address of the CLUT entry that contains the color value that will be used for the given pixel.

Files

Name	Description
Palette.h (🔗)	This is file Palette.h.

Functions

	Name	Description
💡	ClearPaletteChangeError (🔗)	Clears the Palette change error status
💡	DisablePalette (🔗)	Disables the Palette mode.
💡	EnablePalette (🔗)	Enables the Palette mode.
💡	GetPaletteChangeError (🔗)	Returns the Palette change error status
💡	IsPaletteEnabled (🔗)	Returns if the Palette mode is enabled or not.
💡	Palettelnit (🔗)	Initializes the color look up table (CLUT).
💡	RequestPaletteChange (🔗)	Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.
💡	SetPalette (🔗)	Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.
💡	SetPaletteBpp (🔗)	Sets the color look up table (CLUT) number of valid entries.
💡	SetPaletteFlash (🔗)	Loads the palettes from the flash immediately.

Macros

Name	Description
RequestEntirePaletteChange (🔗)	Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.
SetEntirePalette (🔗)	Programs the whole 256 entry palette with new color values from flash.

Structures

Name	Description
PALETTE_FLASH (🔗)	Structure for the palette stored in FLASH memory.
PALETTE_HEADER (🔗)	Structure for the palette header.

Types

Name	Description
PALETTE_EXTERNAL (🔗)	Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

6.4.3.4.3.1 ClearPaletteChangeError Function

File

Palette.h ([🔗](#))

C

```
void ClearPaletteChangeError();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Clears the Palette change error status

Syntax

```
void ClearPaletteChangeError(void)
```

6.4.3.4.3.2 DisablePalette Function

File

Palette.h ()

C

```
void DisablePalette();
```

Returns

none

Preconditions

none

Overview

Disables the Palette mode.

Syntax

```
void DisablePalette(void)
```

6.4.3.4.3.3 EnablePalette Function

File

Palette.h ()

C

```
void EnablePalette();
```

Returns

none

Preconditions

none

Overview

Enables the Palette mode.

Syntax

```
void EnablePalette(void)
```

6.4.3.4.3.4 GetPaletteChangeError Function

File

Palette.h ([🔗](#))

C

```
BYTE GetPaletteChangeError();
```

Side Effects

none

Returns

Returns the palette change status. 1 - If the palette change error occurred 0 - If no palette change error occurred

Preconditions

none

Overview

Returns the Palette change error status

Syntax

```
BYTE GetPaletteChangeError(void)
```

6.4.3.4.3.5 IsPaletteEnabled Function

File

Palette.h ([🔗](#))

C

```
BYTE IsPaletteEnabled();
```

Returns

Returns the palette mode status. 1 - If the palette mode is enabled 0 - If the palette mode is disabled

Preconditions

none

Overview

Returns if the Palette mode is enabled or not.

Syntax

```
BYTE IsPaletteEnabled(void)
```

6.4.3.4.3.6 Palettelnit Function

File

Palette.h ([🔗](#))

C

```
void Palettelnit();
```

Side Effects

All rendering will use the new palette entries.

Returns

none

Preconditions

none

Overview

Initializes the color look up table (CLUT).

Syntax

```
void Palettelnit(void)
```

6.4.3.4.3.7 RequestPaletteChange Function**File**

Palette.h ([🔗](#))

C

```
void RequestPaletteChange(
    void * pPalette,
    WORD startEntry,
    WORD length
);
```

Input Parameters

Input Parameters	Description
void * pPalette	Pointer to the palette structure
WORD startEntry	Start entry to load (inclusive)
WORD length	Number of entries

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

Returns

none

Preconditions

Palette must be initialized by PaletteInit ([🔗](#)()).

Overview

Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.

Syntax

```
void RequestPaletteChange(void *pPalette, WORD startEntry, WORD length)
```

6.4.3.4.3.8 SetPalette Function**File**

Palette.h ([🔗](#))

C

```
BYTE SetPalette(
    void * pPalette,
    WORD startEntry,
    WORD length
);
```

Input Parameters

Input Parameters	Description
void * pPalette	Pointer to the palette structure

WORD startEntry	Start entry to load (inclusive)
WORD length	Number of entries

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

Returns

Returns the status of the palette set. 0 - Set was successful 1 - Set was not successful

Preconditions

Palette must be initialized by PaletteInit (§)().

Overview

Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.

Syntax

```
BYTE SetPalette(void *pPalette, WORD startEntry, WORD length)
```

6.4.3.4.3.9 SetPaletteBpp Function**File**

Palette.h (§)

C

```
BYTE SetPaletteBpp(
    BYTE bpp
);
```

Input Parameters

Input Parameters	Description
BYTE bpp	Bits per pixel

Side Effects

none

Returns

Returns the status of the change. 0 - Change was successful 1 - Change was not successful

Preconditions

Palette must be initialized by PaletteInit (§)().

Overview

Sets the color look up table (CLUT) number of valid entries.

Syntax

```
BYTE SetPaletteBpp(BYTE bpp)
```

6.4.3.4.3.10 SetPaletteFlash Function**File**

Palette.h (§)

C

```
BYTE SetPaletteFlash(
    PALETTE_ENTRY * pPaletteEntry,
    WORD startEntry,
    WORD length
);
```

Input Parameters

Input Parameters	Description
PALETTE_ENTRY * pPaletteEntry	Pointer to the palette table in ROM
WORD startEntry	Start entry to load (inclusive)
WORD length	Number of entries

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

Returns

Returns the status of the palette set. 0 - Set was successful 1 - Set was not successful

Preconditions

Palette must be initialized by PaletteInit (🔗).

Overview

Loads the palettes from the flash immediately.

Syntax

```
BYTE SetPaletteFlash(PALETTE_ENTRY *pPaletteEntry, WORD startEntry, WORD length)
```

6.4.3.4.3.11 PALETTE_FLASH Structure**File**

Palette.h (🔗)

C

```
typedef struct {
    SHORT type;
    PALETTE_HEADER header;
    PALETTE_ENTRY * pPaletteEntry;
} PALETTE_FLASH;
```

Members

Members	Description
SHORT type;	Type must be FLASH
PALETTE_HEADER header;	Contains information on the palette
PALETTE_ENTRY * pPaletteEntry;	Pointer to the palette. Number of entries is determined by the header.

Overview

Structure for the palette stored in FLASH memory.

6.4.3.4.3.12 PALETTE_HEADER Structure**File**

Palette.h (🔗)

C

```
typedef struct {
    WORD id;
    WORD length;
} PALETTE_HEADER;
```

Members

Members	Description
WORD id;	User defined ID

WORD length;	number of palette entries (number of colors in the palette)
--------------	---

Overview

Structure for the palette header.

6.4.3.4.3.13 PALETTE_EXTERNAL Type**File**

Palette.h ([🔗](#))

C

```
typedef GFX_EXTDATA PALETTE_EXTERNAL;
```

Overview

Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

6.4.3.4.3.14 RequestEntirePaletteChange Macro**File**

Palette.h ([🔗](#))

C

```
#define RequestEntirePaletteChange(pPalette) RequestPaletteChange(pPalette, 0, 256)
```

Input Parameters

Input Parameters	Description
pPalette	Pointer to the palette structure

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

Returns

none

Preconditions

PPalette must be initialized by Palettelnit ([🔗](#)()).

Overview

Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.

Syntax

```
RequestEntirePaletteChange(pPalette)
```

6.4.3.4.3.15 SetEntirePalette Macro**File**

Palette.h ([🔗](#))

C

```
#define SetEntirePalette(pPalette) SetPalette(pPalette, 0, 256)
```

Input Parameters

Input Parameters	Description
pPalette	Pointer to the palette structure

Side Effects

There may be a slight flicker when the Palette entries are getting loaded one by one.

Returns

Returns the status of the palette set. 0 - Set was successful 1 - Set was not successful

Preconditions

Palette must be initialized by Palettelnit (█)().

Overview

Programs the whole 256 entry palette with new color values from flash.

Syntax

SetEntirePalette(pPalette)

6.4.3.4.3.16 Palette.h**Functions**

	Name	Description
≡	ClearPaletteChangeError (█)	Clears the Palette change error status
≡	DisablePalette (█)	Disables the Palette mode.
≡	EnablePalette (█)	Enables the Palette mode.
≡	GetPaletteChangeError (█)	Returns the Palette change error status
≡	IsPaletteEnabled (█)	Returns if the Palette mode is enabled or not.
≡	Palettelnit (█)	Initializes the color look up table (CLUT).
≡	RequestPaletteChange (█)	Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.
≡	SetPalette (█)	Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.
≡	SetPaletteBpp (█)	Sets the color look up table (CLUT) number of valid entries.
≡	SetPaletteFlash (█)	Loads the palettes from the flash immediately.

Macros

Name	Description
RequestEntirePaletteChange (█)	Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.
SetEntirePalette (█)	Programs the whole 256 entry palette with new color values from flash.

Structures

Name	Description
PALETTE_FLASH (█)	Structure for the palette stored in FLASH memory.
PALETTE_HEADER (█)	Structure for the palette header.

Types

Name	Description
PALETTE_EXTERNAL (█)	Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Description

This is file Palette.h.

Body Source

```
*****
* Module for Microchip Graphics Library
* Palette Support
*****
* FileName:          Palette.h
* Dependencies:     Graphics.h
```

```

* Processor:          PIC24, PIC32
* Compiler:           MPLAB C30, MPLAB C32
* Linker:             MPLAB LINK30, MPLAB LINK32
* Company:            Microchip Technology Incorporated
*
* Software License Agreement
*
* Copyright © 2008 Microchip Technology Inc. All rights reserved.
* Microchip licenses to you the right to use, modify, copy and distribute
* Software only when embedded on a Microchip microcontroller or digital
* signal controller, which is integrated into your product or third party
* product (pursuant to the sublicense terms in the accompanying license
* agreement).
*
* You should refer to the license agreement accompanying this Software
* for additional information regarding your rights and obligations.
*
* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY
* OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR
* PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR
* OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION,
* BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT
* DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL,
* INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA,
* COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY
* CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
* OR OTHER SIMILAR COSTS.
*
* Date      Comment
* ~~~~~
* 11/06/09  Initial Release
* 08/20/10  Modified PALETTE_EXTERNAL to be of type GFX_EXTDATA.
* 03/20/12  Modified PALETTE_ENTRY structure to have packed attribute.
***** */

#ifndef _PALETTE_H
#define _PALETTE_H

#include "Graphics/Graphics.h"
#include "Primitive.h"
#include "GenericTypeDefs.h"

#ifdef USE_PALETTE

/*********************************************
* Overview: Structure used for the palette entries.
* - For TFT: color is defined as 5-6-5 RGB format
*   (5-bits for RED, 6-bits for GREEN and 5-bits for BLUE.
* - For Monochrome: 4 bits are used to represent the luma.
*
*********************************************/

typedef union __attribute__ ((packed))
{
    WORD      value;                      // a 16-bit value representing a color or palette
entry
    struct __attribute__ ((packed))
    {
        WORD      r : 5;                  // represents the RED component
        WORD      g : 6;                  // represents the GREEN component
        WORD      b : 5;                  // represents the BLUE component
    } color;
    struct __attribute__ ((packed))
    {
        WORD      reserved : 12;         // reserved, used as a filler
        WORD      luma : 4;              // monochrome LUMA value
    } monochrome;
} PALETTE_ENTRY;

/*********************************************

```

```

* Overview: Structure for the palette header.
*
*****  

typedef struct
{
    WORD           id;          // User defined ID
    WORD           length;      // number of palette entries (number of colors in
the palette)
} PALETTE_HEADER;  

*****  

* Overview: Structure for the palette stored in FLASH memory.
*
*****  

typedef struct
{
    SHORT          type;        // Type must be FLASH
    PALETTE_HEADER header;    // Contains information on the palette
    PALETTE_ENTRY  *pPaletteEntry; // Pointer to the palette. Number of entries is
determined by the header.
} PALETTE_FLASH;  

*****  

* Overview: Structure for palette stored in EXTERNAL memory space.
*           (example: External SPI or parallel Flash, EDS_EPMP)
*
*****  

typedef GFX_EXTDATA PALETTE_EXTERNAL;  

*****  

* Function: void PaletteInit(void)
*
* Overview: Initializes the color look up table (CLUT).
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: All rendering will use the new palette entries.
*
*****  

void     PaletteInit(void);  

*****  

* Function: void EnablePalette(void)
*
* Overview: Enables the Palette mode.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects:
*
*****  

void     EnablePalette(void);  

*****  

* Function: void DisablePalette(void)
*
* Overview: Disables the Palette mode.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*

```

```
* Side Effects:  
*  
*****  
void DisablePalette(void);  
  
*****  
* Function: BYTE IsPaletteEnabled(void)  
*  
* Overview: Returns if the Palette mode is enabled or not.  
*  
* PreCondition: none  
*  
* Input: none  
*  
* Output: Returns the palette mode status.  
*          1 - If the palette mode is enabled  
*          0 - If the palette mode is disabled  
*  
* Side Effects:  
*  
*****  
BYTE IsPaletteEnabled(void);  
  
*****  
* Function: BYTE GetPaletteChangeError(void)  
*  
* Overview: Returns the Palette change error status  
*  
* PreCondition: none  
*  
* Input: none  
*  
* Output: Returns the palette change status.  
*          1 - If the palette change error occurred  
*          0 - If no palette change error occurred  
*  
* Side Effects: none  
*  
*****  
BYTE GetPaletteChangeError(void);  
  
*****  
* Function: void ClearPaletteChangeError(void)  
*  
* Overview: Clears the Palette change error status  
*  
* PreCondition: none  
*  
* Input: none  
*  
* Output: none  
*  
* Side Effects: none  
*  
*****  
void ClearPaletteChangeError(void);  
  
*****  
* Function: BYTE SetPaletteBpp(BYTE bpp)  
*  
* Overview: Sets the color look up table (CLUT) number of valid entries.  
*  
* PreCondition: Palette must be initialized by PaletteInit().  
*  
* Input: bpp - Bits per pixel  
*  
* Output: Returns the status of the change.  
*          0 - Change was successful  
*          1 - Change was not successful  
*  
* Side Effects: none  
*
```

```
*****
BYTE SetPaletteBpp(BYTE bpp);

*****
* Function: void RequestPaletteChange(void *pPalette, WORD startEntry, WORD length)
*
* Overview: Loads the palettes from the flash during vertical blanking period
* if possible, otherwise loads immediately.
*
* PreCondition: Palette must be initialized by PaletteInit().
*
* Input: pPalette - Pointer to the palette structure
*        startEntry - Start entry to load (inclusive)
*        length      - Number of entries
*
* Output: none
*
* Side Effects: There may be a slight flicker when the Palette entries
*                are getting loaded one by one.
*
*****
void RequestPaletteChange(void *pPalette, WORD startEntry, WORD length);

*****
* Macro: RequestEntirePaletteChange(pPalette)
*
* Overview: Loads all the palette entries from the flash during
*           vertical blanking period if possible, otherwise
*           loads immediately.
*
* PreCondition: PPalette must be initialized by PaletteInit().
*
* Input: pPalette - Pointer to the palette structure
*
* Output: none
*
* Side Effects: There may be a slight flicker when the Palette entries
*                are getting loaded one by one.
*
*****
#define RequestEntirePaletteChange(pPalette) RequestPaletteChange(pPalette, 0,
256)

*****
* Function: BYTE SetPalette(void *pPalette, WORD startEntry, WORD length)
*
* Overview: Programs a block of palette entries starting from startEntry and
*           until startEntry + length from the flash immediately.
*
* PreCondition: Palette must be initialized by PaletteInit().
*
* Input: pPalette - Pointer to the palette structure
*        startEntry - Start entry to load (inclusive)
*        length      - Number of entries
*
* Output: Returns the status of the palette set.
*         0 - Set was successful
*         1 - Set was not successful
*
* Side Effects: There may be a slight flicker when the Palette entries
*                are getting loaded one by one.
*
*****
BYTE SetPalette(void *pPalette, WORD startEntry, WORD length);

*****
* Macro: SetEntirePalette(pPalette)
*
* Overview: Programs the whole 256 entry palette with new color values
*           from flash.
*
* PreCondition: Palette must be initialized by PaletteInit().

```

```

*
* Input: pPalette - Pointer to the palette structure
*
* Output: Returns the status of the palette set.
*          0 - Set was successful
*          1 - Set was not successful
*
* Side Effects: There may be a slight flicker when the Palette entries
*                are getting loaded one by one.
*
*****  

#define SetEntirePalette(pPalette)  SetPalette(pPalette, 0, 256)

*****  

* Function: BYTE SetPaletteFlash(PALETTE_ENTRY *pPaletteEntry, WORD startEntry, WORD length)
*
* Overview: Loads the palettes from the flash immediately.
*
* PreCondition: Palette must be initialized by PaletteInit().
*
* Input: pPaletteEntry      - Pointer to the palette table in ROM
*        startEntry        - Start entry to load (inclusive)
*        length            - Number of entries
*
* Output: Returns the status of the palette set.
*         0 - Set was successful
*         1 - Set was not successful
*
* Side Effects: There may be a slight flicker when the Palette entries
*                are getting loaded one by one.
*
*****  

BYTE    SetPaletteFlash(PALETTE_ENTRY *pPaletteEntry, WORD startEntry, WORD length);

#endif //USE_PALETTE
#endif

```

6.4.3.4.4 Set Up Display Interface

Macros

Name	Description
GFX_GCLK_DIVIDER (🔗)	<p>The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)).</p> <ul style="list-style-type: none"> When using internal or external memory GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock. GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location. GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2). When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)

GFX_EPMP_CS1_BASE_ADDRESS (🔗)	<p>The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)).</p> <ul style="list-style-type: none"> • When using internal or external memory • GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock. • GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location. • GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2). • When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)
GFX_EPMP_CS1_MEMORY_SIZE (🔗)	<p>The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)).</p> <ul style="list-style-type: none"> • When using internal or external memory • GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock. • GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location. • GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2). • When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)
GFX_EPMP_CS2_BASE_ADDRESS (🔗)	<p>The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)).</p> <ul style="list-style-type: none"> • When using internal or external memory • GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock. • GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location. • GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2). • When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)

GFX_EPMP_CS2_MEMORY_SIZE (🔗)	The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)). <ul style="list-style-type: none">• When using internal or external memory• GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.• GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location.• GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).• When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)
GFX_DISPLAY_BUFFER_LENGTH (🔗)	The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)). <ul style="list-style-type: none">• When using internal or external memory• GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.• GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location.• GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).• When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)
GFX_DISPLAY_BUFFER_START_ADDRESS (🔗)	The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (🔗)). <ul style="list-style-type: none">• When using internal or external memory• GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.• GFX_DISPLAY_BUFFER_START_ADDRESS (🔗) - Set the Display Buffer location.• GFX_DISPLAY_BUFFER_LENGTH (🔗) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).• When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on... more (🔗)

6.4.3.4.4.1 GFX_GCLK_DIVIDER Macro

File

HardwareProfile.h

C

```
#define GFX_GCLK_DIVIDER 61
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that

comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (█)).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS (█) - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH (█) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS (█) - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS (█) - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE (█) - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE (█) - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE (█)	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█)	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█) + USE_COMP_IPU (█)	EQ3	chip select 1 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█)	EQ5	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█) + USE_COMP_IPU (█)	EQ6	chip select 2 region.

Equation	Description
EQ1	GFX_EPMP_CS1_MEMORY_SIZE (█) >= GFX_DISPLAY_BUFFER_LENGTH (█)
EQ2	GFX_EPMP_CS1_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2)
EQ3	GFX_EPMP_CS1_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE
EQ4	GFX_EPMP_CS2_MEMORY_SIZE (█) >= GFX_DISPLAY_BUFFER_LENGTH (█)
EQ5	GFX_EPMP_CS2_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2)
EQ6	GFX_EPMP_CS2_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE

6.4.3.4.4.2 GFX_EPMP_CS1_BASE_ADDRESS Macro

File

HardwareProfile.h

C

```
#define GFX_EPMP_CS1_BASE_ADDRESS 0x00020000ul           // <COPY GFX_GCLK_DIVIDER>
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (█)).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS (█) - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH (█) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS (█) - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE (█) - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE (█) - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE (█)	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█)	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█) + USE_COMP_IPU (█)	EQ3	chip select 1 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█)	EQ5	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█) + USE_COMP_IPU (█)	EQ6	chip select 2 region.

Equation	Description
EQ1	GFX_EPMP_CS1_MEMORY_SIZE (█) >= GFX_DISPLAY_BUFFER_LENGTH (█)
EQ2	GFX_EPMP_CS1_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2)
EQ3	GFX_EPMP_CS1_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE
EQ4	GFX_EPMP_CS2_MEMORY_SIZE (█) >= GFX_DISPLAY_BUFFER_LENGTH (█)
EQ5	GFX_EPMP_CS2_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2)
EQ6	GFX_EPMP_CS2_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE

6.4.3.4.4.3 GFX_EPMP_CS1_MEMORY_SIZE Macro

File

HardwareProfile.h

C

```
#define GFX_EPMP_CS1_MEMORY_SIZE 0x40000ul // <COPY GFX_GCLK_DIVIDER>
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (图)).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS (图) - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH (图) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS (图) - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS (图) - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE (图) - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE	USE_DOUBLE_BUFFERING (图)	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE	USE_DOUBLE_BUFFERING (图) + USE_COMP_IPU (图)	EQ3	chip select 1 region.
GFX_EPMP_CS2_MEMORY_SIZE (图)	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (图)	USE_DOUBLE_BUFFERING (图)	EQ5	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (图)	USE_DOUBLE_BUFFERING (图) + USE_COMP_IPU (图)	EQ6	chip select 2 region.

Equation	Description
EQ1	$\text{GFX_EPMP_CS1_MEMORY_SIZE} \geq \text{GFX_DISPLAY_BUFFER_LENGTH} (\text{图})$
EQ2	$\text{GFX_EPMP_CS1_MEMORY_SIZE} \geq (\text{GFX_DISPLAY_BUFFER_LENGTH} (\text{图}) * 2)$
EQ3	$\text{GFX_EPMP_CS1_MEMORY_SIZE} \geq (\text{GFX_DISPLAY_BUFFER_LENGTH} (\text{图}) * 2) + \text{GFX_COMPRESSED_BUFFER_SIZE} + \text{GFX_DECOMPRESSED_BUFFER_SIZE}$
EQ4	$\text{GFX_EPMP_CS2_MEMORY_SIZE} (\text{图}) \geq \text{GFX_DISPLAY_BUFFER_LENGTH} (\text{图})$
EQ5	$\text{GFX_EPMP_CS2_MEMORY_SIZE} (\text{图}) \geq (\text{GFX_DISPLAY_BUFFER_LENGTH} (\text{图}) * 2)$

EQ6	$\text{GFX_EPMP_CS2_MEMORY_SIZE} \geq (\text{GFX_DISPLAY_BUFFER_LENGTH} * 2) + \text{GFX_COMPRESSED_BUFFER_SIZE} + \text{GFX_DECOMPRESSED_BUFFER_SIZE}$
-----	--

6.4.3.4.4.4 GFX_EPMP_CS2_BASE_ADDRESS Macro

File

HardwareProfile.h

C

```
#define GFX_EPMP_CS2_BASE_ADDRESS 0x00020000u1 // <COPY GFX_GCLK_DIVIDER>
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE	USE_DOUBLE_BUFFERING	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE	USE_DOUBLE_BUFFERING + USE_COMP_IPU	EQ3	chip select 1 region.
GFX_EPMP_CS2_MEMORY_SIZE	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE	USE_DOUBLE_BUFFERING	EQ5	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE	USE_DOUBLE_BUFFERING + USE_COMP_IPU	EQ6	chip select 2 region.

Equation	Description
EQ1	$\text{GFX_EPMP_CS1_MEMORY_SIZE} \geq \text{GFX_DISPLAY_BUFFER_LENGTH}$
EQ2	$\text{GFX_EPMP_CS1_MEMORY_SIZE} \geq (\text{GFX_DISPLAY_BUFFER_LENGTH} * 2)$

EQ3	GFX_EPMP_CS1_MEMORY_SIZE (图) >= (GFX_DISPLAY_BUFFER_LENGTH (图)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE
EQ4	GFX_EPMP_CS2_MEMORY_SIZE (图) >= GFX_DISPLAY_BUFFER_LENGTH (图)
EQ5	GFX_EPMP_CS2_MEMORY_SIZE (图) >= (GFX_DISPLAY_BUFFER_LENGTH (图)*2)
EQ6	GFX_EPMP_CS2_MEMORY_SIZE (图) >= (GFX_DISPLAY_BUFFER_LENGTH (图)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE

6.4.3.4.4.5 GFX_EPMP_CS2_MEMORY_SIZE Macro

File

HardwareProfile.h

C

```
#define GFX_EPMP_CS2_MEMORY_SIZE 0x40000ul // <COPY GFX_GCLK_DIVIDER>
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (图)).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS (图) - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH (图) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS (图) - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS (图) - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE (图) - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE (图)	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (图)	USE_DOUBLE_BUFFERING (图)	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (图)	USE_DOUBLE_BUFFERING (图) + USE_COMP_IPU (图)	EQ3	chip select 1 region.
GFX_EPMP_CS2_MEMORY_SIZE	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE	USE_DOUBLE_BUFFERING (图)	EQ5	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE	USE_DOUBLE_BUFFERING (图) + USE_COMP_IPU (图)	EQ6	chip select 2 region.

Equation	Description
EQ1	GFX_EPMP_CS1_MEMORY_SIZE (█) >= GFX_DISPLAY_BUFFER_LENGTH (█)
EQ2	GFX_EPMP_CS1_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2)
EQ3	GFX_EPMP_CS1_MEMORY_SIZE (█) >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE
EQ4	GFX_EPMP_CS2_MEMORY_SIZE >= GFX_DISPLAY_BUFFER_LENGTH (█)
EQ5	GFX_EPMP_CS2_MEMORY_SIZE >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2)
EQ6	GFX_EPMP_CS2_MEMORY_SIZE >= (GFX_DISPLAY_BUFFER_LENGTH (█)*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE

6.4.3.4.4.6 GFX_DISPLAY_BUFFER_LENGTH Macro

File

HardwareProfile.h

C

```
#define GFX_DISPLAY_BUFFER_LENGTH 0x00025800ul           // <COPY GFX_GCLK_DIVIDER>
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD (█)).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS (█) - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS (█) - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS (█) - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE (█) - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE (█) - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE (█)	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█)	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█) + USE_COMP_IPU (█)	EQ3	chip select 1 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (█)	USE_DOUBLE_BUFFERING (█)	EQ5	chip select 2 region.

GFX_EPMP_CS2_MEMORY_SIZE (¶)	USE_DOUBLE_BUFFERING (¶) + USE_COMP_IPU (¶)	EQ6	chip select 2 region.
---	--	-----	-----------------------

Equation	Description
EQ1	GFX_EPMP_CS1_MEMORY_SIZE (¶) >= GFX_DISPLAY_BUFFER_LENGTH
EQ2	GFX_EPMP_CS1_MEMORY_SIZE (¶) >= (GFX_DISPLAY_BUFFER_LENGTH*2)
EQ3	GFX_EPMP_CS1_MEMORY_SIZE (¶) >= (GFX_DISPLAY_BUFFER_LENGTH*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE
EQ4	GFX_EPMP_CS2_MEMORY_SIZE (¶) >= GFX_DISPLAY_BUFFER_LENGTH
EQ5	GFX_EPMP_CS2_MEMORY_SIZE (¶) >= (GFX_DISPLAY_BUFFER_LENGTH*2)
EQ6	GFX_EPMP_CS2_MEMORY_SIZE (¶) >= (GFX_DISPLAY_BUFFER_LENGTH*2) + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE

6.4.3.4.4.7 GFX_DISPLAY_BUFFER_START_ADDRESS Macro

File

HardwareProfile.h

C

```
#define GFX_DISPLAY_BUFFER_START_ADDRESS 0x00020000ul // <COPY GFX\_GCLK\_DIVIDER>
```

Overview

The following are additional Hardware Profile macros used when using the driver for the Microchip Graphics Module that comes with the PIC Microcontroller (PIC24FJ256DA210 Device Family) (PIC24FJ256DA210_DEV_BOARD ([¶](#))).

- When using internal or external memory
- GFX_GCLK_DIVIDER - Set the clock divider for the pixel clock.
- GFX_DISPLAY_BUFFER_START_ADDRESS - Set the Display Buffer location.
- GFX_DISPLAY_BUFFER_LENGTH ([¶](#)) - Set the Display Buffer length (size) in bytes. This is calculated by the display's width*height*(color depth/2).
- When using external memory only. External memory may be placed in chip select 1 (CS1) and/or chip select 2 (CS2) regions. Refer to EPMP Family Reference Manual for details on how to use EPMP for graphics applications.
- GFX_EPMP_CS1_BASE_ADDRESS ([¶](#)) - Set the location of the external memory mapped to the EPMP CS1.
- GFX_EPMP_CS2_BASE_ADDRESS ([¶](#)) - Set the location of the external memory mapped to the EPMP CS2.
- GFX_EPMP_CS1_MEMORY_SIZE ([¶](#)) - Set the size of the memory mapped to the EPMP CS1. This value sets how many EPMP address lines will be used.
- GFX_EPMP_CS2_MEMORY_SIZE ([¶](#)) - Set the size of the memory mapped to the EPMP CS2. This value sets how many EPMP address lines will be used.

If the display buffer is located in external memory (i.e in PIC24FJ256DA210) then the memory requirement for the graphics use must fit into the chip select region size. The table below summarizes the requirements.

Buffer Name	Enabled Features	External Memory Required	Display Buffer Location
GFX_EPMP_CS1_MEMORY_SIZE (¶)	none	EQ1	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (¶)	USE_DOUBLE_BUFFERING (¶)	EQ2	chip select 1 region.
GFX_EPMP_CS1_MEMORY_SIZE (¶)	USE_DOUBLE_BUFFERING (¶) + USE_COMP_IPU (¶)	EQ3	chip select 1 region.

GFX_EPMP_CS2_MEMORY_SIZE (#)	none	EQ4	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (#)	USE_DOUBLE_BUFFERING (#)	EQ5	chip select 2 region.
GFX_EPMP_CS2_MEMORY_SIZE (#)	USE_DOUBLE_BUFFERING (#) + USE_COMP_IPU (#)	EQ6	chip select 2 region.

Equation	Description
EQ1	GFX_EPMP_CS1_MEMORY_SIZE (#) \geq GFX_DISPLAY_BUFFER_LENGTH (#)
EQ2	GFX_EPMP_CS1_MEMORY_SIZE (#) \geq (GFX_DISPLAY_BUFFER_LENGTH (#))^2
EQ3	GFX_EPMP_CS1_MEMORY_SIZE (#) \geq (GFX_DISPLAY_BUFFER_LENGTH (#))^2 + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE
EQ4	GFX_EPMP_CS2_MEMORY_SIZE (#) \geq GFX_DISPLAY_BUFFER_LENGTH (#)
EQ5	GFX_EPMP_CS2_MEMORY_SIZE (#) \geq (GFX_DISPLAY_BUFFER_LENGTH (#))^2
EQ6	GFX_EPMP_CS2_MEMORY_SIZE (#) \geq (GFX_DISPLAY_BUFFER_LENGTH (#))^2 + GFX_COMPRESSED_BUFFER_SIZE + GFX_DECOMPRESSED_BUFFER_SIZE

6.4.3.4.5 External or Internal Memory and Palettes

This section shows examples on how to set up applications using external memory, internal memory or use palettes for Microchip Graphics Module.

Description

Applications can run in PIC24FJ256DA210 Device Family using internal memory or external memory with palette enabled or disabled. The table below summarizes the allowed color depth when combining palette with the use of internal or external memory.

Memory Location	Palette Mode	Color Depth (BPP)	Notes
Internal	enabled	1, 2, 4, 8	When using internal memory and running TFT display, palette mode must be enabled.
External	enabled	8, 16	Palette mode can only be used for 8BPP color depth when using external memory.
External	disabled	16	When using external memory and palette is disabled and running TFT display, 16 BPP color depth must be used.

Applications Using Palettes

Settings in Graphics Configuration (*GraphicsConfig.h*)

```
#define USE_PALETTE // to include code for palette
#define USE_PALETTE_EXTERNAL // define this if palette data
// will be in the external memory

// (note: PIC24FJ256DA210 Device Family color palette has a maximum of 256
// entries, therefore the maximum color depth that can be used for applications
// using palette is 8 BPP)
#define COLOR_DEPTH 8 // define the color depth to use
```

Settings in Hardware Profile (*HardwareProfile.h*)

```
///////////
// Microchip Specific Development Tools
/////////
#define PIC24FJ256DA210_DEV_BOARD // to use PIC24FJ256DA210
Development Board
```

```

#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E           // to use the Truly 3.2" display
panel

//////////////////////////////  

// PIC24FJ256DA210 Graphics Module required settings  

/////////////////////////////  

#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210                  // to use the mchpGfxDrv.c and
mchpGfxDrv.h driver

#define GFX_GCLK_DIVIDER 38                                     // set divider to generate required
GCLK pin                                                       // frequency (check display panel

data sheet)                                                 // set the display buffer starting
#define GFX_DISPLAY_BUFFER_START_ADDRESS 0x00004B00ul        // address can be located in
address                                                       // external memory)
bytes (value)                                               // set the display buffer length in
and color depth                                            // is derived from width, height
external memory,                                            // (COLOR_DEPTH)). If using
memory size.                                                // size must fit into the external
                                                               // memory

// If using external memory add macros to enable EPMP CS1 or CS2 here and adjust
// location of GFX_DISPLAY_BUFFER_START_ADDRESS (see Application Using External
Memory)

```

In application code initialize and enable palette before initializing the Graphics Library

```

// set the palette color depth
SetPaletteBpp(8);
// initialize the palette
SetPalette((void*)&MainPalette, 0, 256);

// enable the use of the palette
EnablePalette();

```

Applications Using Internal Memory

Settings in Graphics Configuration (GraphicsConfig.h)

```

// define the color depth to use (1, 2, 4, or 8BPP)
#define COLOR_DEPTH 8

```

Settings in Graphics Configuration (GraphicsConfig.h) Settings in Hardware Profile (HardwareProfile.h)

```

/////////////////////////////  

// Microchip Specific Development Tools  

/////////////////////////////  

#define PIC24FJ256DA210_DEV_BOARD                         // to use PIC24FJ256DA210
Development Board                                         // to use the Truly 3.2" display
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E       // to use the mchpGfxDrv.c and
panel                                                       // set the display buffer starting
                                                               // address can be located in
                                                               // external memory)

/////////////////////////////  

// PIC24FJ256DA210 Graphics Module required settings  

/////////////////////////////  

#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210              // to use the mchpGfxDrv.c and
mchpGfxDrv.h driver

#define GFX_GCLK_DIVIDER 38                                 // set divider to generate required
GCLK pin                                                   // frequency (check display panel

data sheet)                                              // set the display buffer starting
#define GFX_DISPLAY_BUFFER_START_ADDRESS 0x00004B00ul    // address can be located in
address                                                    // external memory)

```

internal memory)

#define GFX_DISPLAY_BUFFER_LENGTH 0x0004B000ul
bytes (value

and color depth

into internal memory

Applications Using External Memory

Settings in Graphics Configuration (GraphicsConfig.h)

```
// define the color depth to use (8 or 16BPP)
#define COLOR_DEPTH 16
```

Settings in Hardware Profile (HardwareProfile.h)

```
///////////////////////////////
// Microchip Specific Development Tools
/////////////////////////////
#define PIC24FJ256DA210_DEV_BOARD
Development Board
#define GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E
panel

/////////////////////////////
// PIC24FJ256DA210 Graphics Module required settings
/////////////////////////////
#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210
mchpGfxDrv.h driver

#define GFX_GCLK_DIVIDER 61
GCLK pin

data sheet)
#define USE_16BIT_PMP
#define GFX_DISPLAY_BUFFER_START_ADDRESS 0x00020000ul
address

external memory)

#define GFX_DISPLAY_BUFFER_LENGTH 0x00025800ul
bytes (value

and color depth

into external

region

#define GFX_EPMP_CS1_BASE_ADDRESS 0x00020000ul
starting address

region
#define GFX_EPMP_CS1_MEMORY_SIZE 0x80000ul
bytes)

Address lines to use

#define GFX_EPMP_CS2_BASE_ADDRESS 0x000A0000ul
starting address

#define GFX_EPMP_CS2_MEMORY_SIZE 0x80000ul
bytes)

Address lines to use
```

// set the display buffer length in

// is derived from width, height

// (COLOR_DEPTH)). Size must fit

// to use PIC24FJ256DA210

// to use the Truly 3.2" display

// to use the mchpGfxDrv.c and

// set divider to generate required

// frequency (check display panel

// enable the use of EPMP

// set the display buffer starting

// (address will be located in

// set the display buffer length in

// is derived from width, height

// (COLOR_DEPTH)). Size must fit

// memory

// if using EPMP CS1 set the

// of EPMP CS1

// set the EPMP CS1 region size (in

// size defines the number of EPMP

// if using EPMP CS2 set the

// of EPMP CS2 region

// set the EPMP CS2 region size (in

// size defines the number of EPMP

7 Image Decoders

Demo Project

Name	Description
Image Decoder Demo (🔗)	This demo demonstrates the decoding of images with JPEG and BMP file formats.

Description

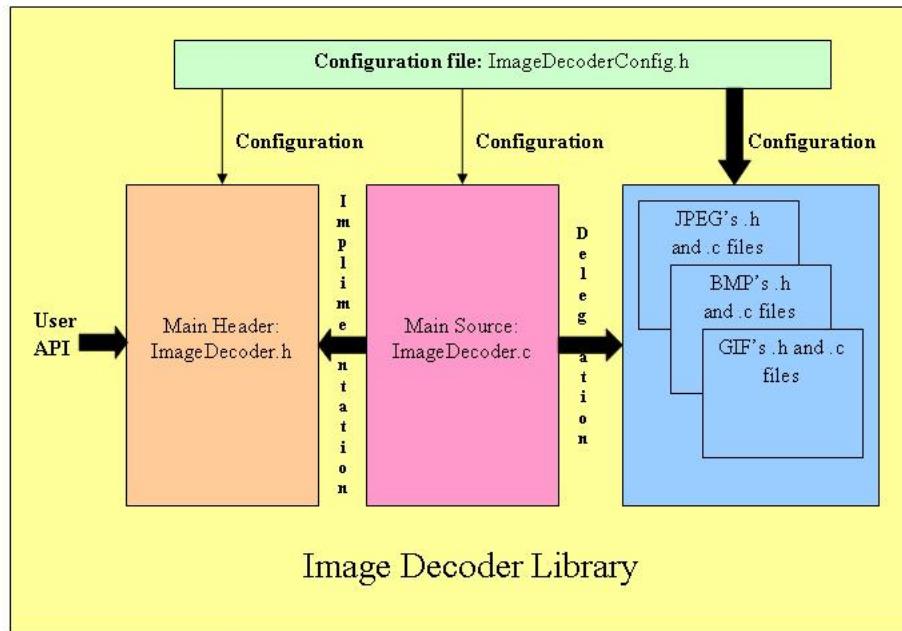
The Image Decoder Library supports the decoding of images in JPEG, BMP, and GIF format in PIC24, dsPIC, and PIC32 devices. This is a supplement to the Graphics Library but could be used without the Graphics Library. It not only supports input data through the Microchip's MDD file system but it can also be configured to support user specific inputs from ROM, external EEPROM, etc. The output can be sent to the Graphics Display through the driver provided with the Graphics Library or to a callback function where user can further render the decoded image (even if Graphics Library is not used). The individual decoders provided uses the stack for the work memory and hence a heap is not required.

Design

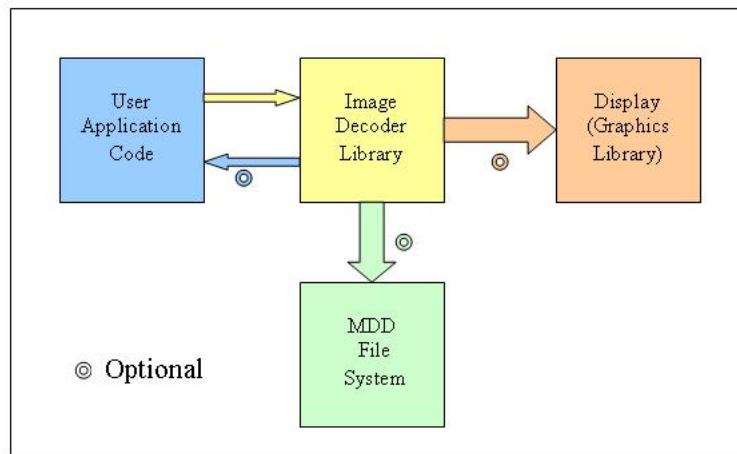
The Image Decoder library includes the following files:

1. Configuration file
 - ImageDecoderConfig.h
2. Header files
 - Main Header
 - ImageDecoder.h
 - Individual decoder headers
 - JpegDecoder.h
 - BmpDecoder.h
 - GifDecoder.h
3. Source files
 - Main Source file
 - ImageDecoder.c
 - Individual Source files
 - JpegDecoder.c
 - BmpDecoder.c
 - GifDecoder.c
4. Support files
 - jidctint.c

This can be diagrammatically explained as below:



The user application interacts with the Image Decoder Library as per the diagram below:



User can use the Graphics Library for output or can provide his/her own output pixel handler callback function. Similarly, user can use MDD file system or provide his/her own input source which implements some specific APIs explained later.

Configuration

The compile time configurations can be done using the ImageDecoderConfig.h file. This file must be copied into the application folder like the other config files. The options provided are as explained below:

1. Image format support

The image formats which are not required can be compiled out by commenting out the respective defines. The below section says that GIF support to be excluded.

```
/* Comment out the image formats which are not required */
#define IMG_SUPPORT_BMP
#define IMG_SUPPORT_JPEG
//#define IMG_SUPPORT_GIF
```

2. Optimize for Graphics Library

If user application requires to output the image directly to the display without any double buffering, then the code can be

optimized by defining as below:

```
/* Comment out if output has to be given through a callback function */
/* If defined, the code is optimized to use only the graphics driver, only
16-bit-565-color-format is supported */
#define IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
```

If either double buffering has to be done or if user specific rendering has to be done, comment out the above mentioned line and provide a callback function to render the pixel values. The callback function will be explained in the API section. If commented out, then the width and height of the display screen has to be provided using the following defines

```
/* If the above define is commented out (Graphics driver is not included by default), then
the below defines has to be set by the user */
#ifndef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
#define DISP_HOR_RESOLUTION      320
#define DISP_VER_RESOLUTION      240
#endif
```

3. Optimize for MDD file system

If user application uses only the MDD file system provided by Microchip Technology Inc., the code can be optimized by defining as below:

```
/* If defined, the code is optimized to use only the MDD file system */
#define IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT
```

If MDD file system is not used or if additional input formats have to be supported, the above define has to be commented out and a structure pointing to the required APIs (IMG_FILE_SYSTEM_API defined in ImageDecoder.h) must be provided by the user.

4. Loop callback support

Since decoder takes up significant amount of time decoding an image and user may want to update some information on the display or to send/receive data through communication channel, etc... it is possible to release processing power in the middle of decoding process by calling a callback function provided by the user. The user can do all the housekeeping activities inside the function. This option can be enabled by defining as below:

```
/* If defined, the a loop callback function is called in every
decoding loop so that application can do maintenance activities such as
getting data, updating display, etc... */
#define IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
```

The JPEG decoder calls the callback function after every 8x8 pixel block decode while the BMP and GIF decoders calls after every row decode.

If this support is not required, then this define can be commented out.

Footprint

The following are typical values for PIC24 with default compiler configuration. Actual values may vary with various factors such as compiler optimization levels and device used (PIC24/PIC32).

	RAM (stack)	ROM (no compiler optimizations)	
Image Decoder Core Code	30 Bytes	1 KBytes	
Image Types			Decode Time (in seconds) Using 16 MIPS & Compiler Optimization Level 3
BMP	1 KBytes	10 KBytes	2 ¹
JPEG	3 KBytes	13 KBytes	2 ¹
GIF	11 KBytes(13 KBytes ²)	6 KBytes	2 ¹

(1) Approximate value for average quality image file with a pixel resolution of 320x240. Additional variance in time can be

observed due to image quality/image size and file access time.

(2) GIF code "Crush" optimization (GIF_CRUSH_PREV_SYMBOL_PTR_TABLE) is turned off. This compile switch can be found in GifDecoder.h file. Enabling this will have an additional effect on the GIF decoding time.

7.1 Demo Project

7.1.1 Image Decoder Demo

This demo demonstrates the decoding of images with JPEG and BMP file formats.

Module

[Image Decoders \(](#)

Description

Please refer to the getting started htm document located at <Install Directory>/Microchip/Graphics/Documents/Getting Started/Getting Started - Running the Image Decoders Demo.htm, where <Install Directory> is the root directory of the Microchip Applications Library.

8 Image Decoders API

Functions

	Name	Description
💡	ImageDecode (🔗)	This function decodes and displays the image on the screen
💡	ImageDecoderInit (🔗)	This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called
💡	ImageLoopCallbackRegister (🔗)	This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintenance activities such as fetching data, updating the display, etc...
💡	ImageDecodeTask (🔗)	This function completes one small part of the image decode function

Macros

Name	Description
ImageFullScreenDecode (🔗)	This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required
ImageAbort (🔗)	This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.

Structures

	Name	Description
◆	_BMPDECODER (🔗)	DATA STRUCTURES
◆	_GIFDECODER (🔗)	DATA STRUCTURES
◆	_JPEGDECODER (🔗)	DATA STRUCTURES

Description

The Image Decoder Library distributed with the Graphics Library supports the decoding of images in JPEG, BMP, and GIF format in PIC24, dsPIC, and PIC32 devices. This is a supplement to the Graphics Library but could be used without the Graphics Library. This section describes the APIs for Image Decoder Library.

8.1 Image Decoder Configuration

The Image Decoder Library can be customized by adding or specifying the compile time options located in the application file named ImageDecoderConfig.h.

Macros

Name	Description
IMG_SUPPORT_BMP (🔗)	Add this macro in ImageDecoderConfig.h to enable support for bitmap image format decoding.
IMG_SUPPORT_GIF (🔗)	Add this macro in ImageDecoderConfig.h to enable support for gif image format decoding.
IMG_SUPPORT_JPEG (🔗)	Add this macro in ImageDecoderConfig.h to enable support for jpeg image format decoding.

IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK (🔗)	The decoder may take up significant amount of time decoding an image and user may want to update some information on the display or to send/receive data through some communication channels. It is possible to release processing power in the middle of decoding process by calling a callback function provided by the user. The user should do all the housekeeping activities inside the function. This option can be enabled by adding this macro in ImageDecoderConfig.h.
IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT (🔗)	Add this macro in ImageDecoderConfig.h to optimize code for graphics driver that supports 16-bit 5-6-5 color format and rendering is done directly to the display buffer. If either double buffering has to be done or if user specific rendering has to be done, comment out the above mentioned line and provide a callback function to render the pixel values. The callback function will be explained in the API section. If commented out, then the width and height of the display screen has to be provided using the following defines
IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT (🔗)	Add this macro in ImageDecoderConfig.h to optimize code to use Memory Disk Drive File System(MDDFS) of Microchip MDD File System Interface Library .

Description

8.1.1 IMG_SUPPORT_BMP Macro

File

ImageDecoderConfig.h

C

#define IMG_SUPPORT_BMP

Overview

Add this macro in ImageDecoderConfig.h to enable support for bitmap image format decoding.

8.1.2 IMG_SUPPORT_GIF Macro

File

ImageDecoderConfig.h

C

#define IMG_SUPPORT_GIF

Overview

Add this macro in ImageDecoderConfig.h to enable support for gif image format decoding.

8.1.3 IMG_SUPPORT_JPEG Macro

File

ImageDecoderConfig.h

C

```
#define IMG_SUPPORT_JPEG
```

Overview

Add this macro in ImageDecoderConfig.h to enable support for jpeg image format decoding.

8.1.4

IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK Macro

File

ImageDecoderConfig.h

C

```
#define IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
```

Overview

The decoder may takes up significant amount of time decoding an image and user may want to update some information on the display or to send/receive data through some communication channels. It is possible to release processing power in the middle of decoding process by calling a callback function provided by the user. The user should do all the housekeeping activities inside the function. This option can be enabled by adding this macro in ImageDecoderConfig.h.

8.1.5

IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT Macro

File

ImageDecoderConfig.h

C

```
#define IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
```

Overview

Add this macro in ImageDecoderConfig.h to optimize code for graphics driver that supports 16-bit 5-6-5 color format and rendering is done directly to the display buffer. If either double buffering has to be done or if user specific rendering has to be done, comment out the above mentioned line and provide a callback function to render the pixel values. The callback function will be explained in the API section. If commented out, then the width and height of the display screen has to be provided using the following defines

8.1.6 **IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT Macro**

File

ImageDecoderConfig.h

C

```
#define IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT
```

Overview

Add this macro in ImageDecoderConfig.h to optimize code to use Memory Disk Drive File System(MDDFS) of Microchip MDD File System Interface Library .

8.2 ImageDecode Function

File

ImageDecoder.h

C

```
BYTE ImageDecode(
    IMG_FILE * pImageFile,
    IMG_FILE_FORMAT eImgFormat,
    WORD wStartx,
    WORD wStarty,
    WORD wWidth,
    WORD wHeight,
    WORD wFlags,
    IMG_FILE_SYSTEM_API * pFileAPIs,
    IMG_PIXEL_OUTPUT pPixelOutput
);
```

Side Effects

None

Returns

Error code -> 0 means no error

Example

```
void main(void)
{
    IMG_FILE plmageFile;
    IMG_vInitialize();
    plmageFile = IMG_FOPEN("Image.jpg", "r");
    if(plmageFile == NULL)
    {
        <- Error handling -
    }
    IMG_bDecode(plmageFile, IMG_JPEG, 0, 0, 320, 240, 0, NULL, NULL);
    IMG_FCLOSE(plmageFile);
    while(1);
}
```

Overview

This function decodes and displays the image on the screen

Syntax

```
BYTE ImageDecode(IMG_FILE *pImageFile, IMG_FILE_FORMAT eImgFormat, WORD wStartx, WORD wStarty, WORD wWidth, WORD wHeight, WORD wFlags, IMG_FILE_SYSTEM_API *pFileAPIs, IMG_PIXEL_OUTPUT pPixelOutput)
```

8.3 ImageDecoderInit Function

File

ImageDecoder.h

C

```
void ImageDecoderInit();
```

Side Effects

The graphics driver will be reset

Returns

None

Example

```
void main(void)
{
    ImageInit();
    ...
}
```

Overview

This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called

Syntax

```
void ImageDecoderInit(void)
```

8.4 ImageLoopCallbackRegister Function

File

ImageDecoder.h

C

```
void ImageLoopCallbackRegister(
    IMG_LOOP_CALLBACK pFn
);
```

Side Effects

The graphics driver will be reset

Returns

None

Example

```
void Mainantance(void)
{
    ...
}

void main(void)
{
```

```

    ImageInit();
    ImageLoopCallbackRegister(Mainantance);
    ...
}

```

Overview

This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintenance activities such as fetching data, updating the display, etc...

Syntax

```
void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pLoopCallbackFn)
```

8.5 ImageDecodeTask Function

File

```
ImageDecoder.h
```

C

```
BYTE ImageDecodeTask();
```

Side Effects

None

Returns

Status code - '1' means decoding is completed

- '0' means decoding is not yet completed, call this function again

Example

```
IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
while(!ImageDecodeTask());
```

Overview

This function completes one small part of the image decode function

Syntax

```
BYTE ImageDecodeTask(void)
```

8.6 ImageFullScreenDecode Macro

File

```
ImageDecoder.h
```

C

```
#define ImageFullScreenDecode(pImageFile, eImgFormat, pFileAPIs, pPixelOutput) \
    ImageDecode(pImageFile, eImgFormat, 0, 0, IMG_SCREEN_WIDTH, IMG_SCREEN_HEIGHT, \
    (IMG_ALIGN_CENTER | IMG_DOWN_SCALE), pFileAPIs, pPixelOutput)
```

Side Effects

None

Returns

Error code -> 0 means no error

Example

```
void main(void)
{
    IMG_FILE plImageFile;
    IMG_vInitialize();
    plImageFile = IMG_FOPEN("Image.jpg", "r");
    if(plImageFile == NULL)
    {
        <- Error handling ->
    }
    IMG_bFullScreenDecode(plImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(plImageFile);
    while(1);
}
```

Overview

This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required

Syntax

```
BYTE ImageFullScreenDecode(IMG_FILE *plImageFile, IMG_FILE_FORMAT eImgFormat, IMG_FILE_SYSTEM_API pFileAPIs, IMG_PIXEL_OUTPUT pPixelOutput)
```

8.7 ImageAbort Macro

File

ImageDecoder.h

C

```
#define ImageAbort IMG_bAbortImageDecoding = 1;
```

Side Effects

None

Returns

None

Example

```
void callback(void);
void main(void)
{
    IMG_FILE plImageFile;
    IMG_vInitialize();
    ImageLoopCallbackRegister(callback);
    plImageFile = IMG_FOPEN("Image.jpg", "r");
    if(plImageFile == NULL)
    {
        <- Error handling ->
    }
    IMG_bFullScreenDecode(plImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(plImageFile);
    while(1);
}

void callback(void)
{
    if(<- check for abort condition ->)
```

```
{
    ImageAbort();
}
}
```

Overview

This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.

Syntax

```
void ImageAbort(void)
```

8.8 _BMPDECODER Structure

File

BmpDecoder.c

C

```
struct _BMPDECODER {
    IMG_FILE * pImageFile;
    LONG lWidth;
    LONG lHeight;
    LONG lImageOffset;
    WORD wPaletteEntries;
    BYTE bBitsPerPixel;
    BYTE bHeaderType;
    BYTE blBmMarkerFlag : 1;
    BYTE blCompressionType : 3;
    BYTE bNumOfPlanes : 3;
    BYTE b16bit565Flag : 1;
    BYTE aPalette[256][3];
};
```

Members

Members	Description
IMG_FILE * pImageFile;	Image file pointer
BYTE aPalette[256][3];	Each palette entry has RGB

Description

DATA STRUCTURES

8.9 _GIFDECODER Structure

File

GifDecoder.c

C

```
struct _GIFDECODER {
    IMG_FILE * pImageFile;
    WORD wImageWidth;
    WORD wImageHeight;
    WORD wImageX;
    WORD wImageY;
    WORD wScreenWidth;
    WORD wScreenHeight;
    WORD wGlobalPaletteEntries;
```

```

WORD wLocalPaletteEntries;
BYTE bBgColorIndex;
BYTE bPixelAspectRatio;
BYTE blGifMarkerFlag : 1;
BYTE blGlobalColorTableFlag : 1;
BYTE blLocalColorTableFlag : 1;
BYTE blInterlacedFlag : 1;
BYTE blFirstcodeFlag : 1;
BYTE bInterlacePass : 3;
BYTE aPalette[256][3];
WORD awPalette[256];
BYTE abSymbol[4096];
WORD awPrevSymbolPtr[(4096 * 3)/4];
WORD wInitialSymbols;
WORD wMaxSymbol;
BYTE bInitialSymbolBits;
BYTE bMaxSymbolBits;
LONG lGlobalColorTablePos;
BYTE bWorkBits;
BYTE bRemainingDataInBlock;
BYTE bRemainingBits;
WORD wCurrentX;
WORD wCurrentY;
};

} ;

```

Members

Members	Description
IMG_FILE * pImageFile;	Image file pointer
BYTE aPalette[256][3];	Each palette entry has RGB
WORD awPalette[256];	Each palette entry has RGB
BYTE abSymbol[4096];	For decoding
BYTE bWorkBits;	Work memory

Description

DATA STRUCTURES

8.10 _JPEGDECODER Structure

File

JpegDecoder.c

C

```

struct _JPEGDECODER {
    IMG_FILE * pImageFile;
    BYTE blFIF;
    BYTE bMajorRev;
    BYTE bMinorRev;
    BYTE bDataBits;
    WORD wWidth;
    WORD wHeight;
    BYTE bChannels;
    BYTE abChannelType[MAX_CHANNELS];
    BYTE abChannelHSampFactor[MAX_CHANNELS];
    BYTE abChannelVSampFactor[MAX_CHANNELS];
    BYTE abChannelQuantTableMap[MAX_CHANNELS];
    BYTE blQuantUses16bits;
    WORD awQuantTable[MAX_CHANNELS][64];
    WORD wRestartInterval;
    BYTE bHuffTables;
    BYTE abHuffAcSymLen[MAX_HUFF_TABLES][16];
    BYTE abHuffAcSymbol[MAX_HUFF_TABLES][256];
};

```

```

BYTE abHuffDcSymLen[MAX_HUFF_TABLES][16];
BYTE abHuffDcSymbol[MAX_HUFF_TABLES][16];
WORD awHuffAcSymStart[MAX_HUFF_TABLES][16];
WORD awHuffDcSymStart[MAX_HUFF_TABLES][16];
BYTE abChannelHuffAcTableMap[MAX_CHANNELS];
BYTE abChannelHuffDcTableMap[MAX_CHANNELS];
BYTE bError;
WORD wWorkBits;
BYTE bBitsAvailable;
BYTE bBlocksInOnePass;
SHORT asOneBlock[MAX_BLOCKS][64];
WORD wBlockNumber;
BYTE abChannelMap[MAX_BLOCKS];
BYTE bSubSampleType;
SHORT asPrevDcValue[MAX_CHANNELS];
BYTE * pbCurrentHuffSymLenTable;
BYTE * pbCurrentHuffSymbolTable;
WORD * pwCurrentHuffSymStartTable;
WORD * pwCurrentQuantTable;
BYTE abDataBuffer[MAX_DATA_BUF_LEN];
WORD wBufferLen;
WORD wBufferIndex;
BYTE bFirstBit;
WORD wPrevX;
WORD wPrevY;
};

}

```

Members

Members	Description
IMG_FILE * plImageFile;	Image file pointer
BYTE blJFIF;	JFIF marker found flag
BYTE bMajorRev;	Should be 1
BYTE bMinorRev;	Should be 0-2 but is not a show stopper ----- The x/y densities and thumbnail data are ignored
BYTE bDataBits;	Data precision, can be 8(, 12 or 16)
WORD wWidth;	Width in pixels
WORD wHeight;	Height in pixels
BYTE bChannels;	Number of channels e.g. YCbCr = 3
BYTE blQuantUses16bits;	If flag is set, it is an error as 16 bit is not supported
WORD awQuantTable[MAX_CHANNELS][64];	Supports only 8 & 16 bit resolutions
WORD wRestartInterval;	The restart interval in blocks
BYTE bHuffTables;	From DHT
BYTE abHuffAcSymLen[MAX_HUFF_TABLES][16];	Supports only 8 bit resolution
BYTE abHuffAcSymbol[MAX_HUFF_TABLES][256];	Maximum possible symbols are 256
BYTE abHuffDcSymLen[MAX_HUFF_TABLES][16];	Supports only 8 bit resolution
BYTE abHuffDcSymbol[MAX_HUFF_TABLES][16];	Maximum possible symbols are 16 for DC :-)
WORD awHuffAcSymStart[MAX_HUFF_TABLES][16];	Starting symbol for each length
WORD awHuffDcSymStart[MAX_HUFF_TABLES][16];	Starting symbol for each length
BYTE abChannelHuffAcTableMap[MAX_CHANNELS];	From SOS
WORD wWorkBits;	Work memory
SHORT asOneBlock[MAX_BLOCKS][64];	Temporary storage for a 8x8 block

Description

DATA STRUCTURES

9 Miscellaneous Topics

This section contains common procedures to start using the library or to modify the library.

9.1 Starting a New Project

This outlines the procedure to create a new project that uses the Microchip Graphics Library from scratch.

Description

The following is a step by step instruction to create a New Project from scratch with the Microchip Graphics Library

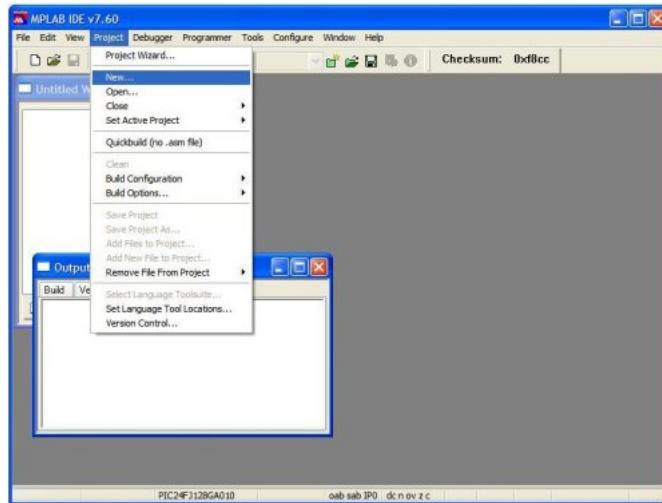
Step 1: Install the Microchip Application Libraries to get the Microchip Graphics Library. The Microchip Application Library installer can be downloaded from www.microchip.com/MLA. Once installed the Microchip Graphics Library files will be located in the directory structure shown below.

```
<MyProjects>
  |
  Board Support Package
  Graphics
  GraphicsProjects1
  Microchip
    |
    Help
    Graphics
      |
      Drivers
      Include
        |
        Graphics
```

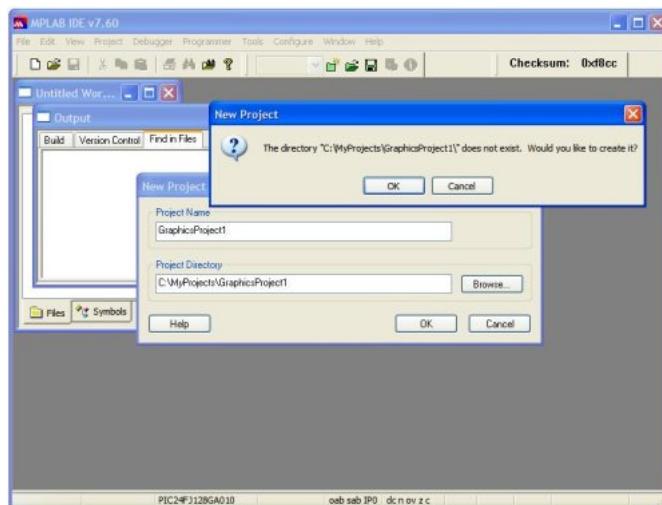
- MyProjects can be any name and is the directory name that you chose when you installed the Microchip Application Libraries.
- Graphics directory will contain demos distributed with the Graphics Library.
- Board Support Package directory will contain drivers for components on Microchip Development Boards that are used by the demos.
- GraphicsProject1 directory can be any directory name that you specify later for your own project.
- Microchip directory will contain all the library files.
- Under Microchip directory, the Help directory will contain the "Graphics.chm" file.
- Under Microchip directory, the Graphics directory under the Microchip directory will contain all the source (C) files for the Graphics Library except the actual display driver files. Under the Drivers directory, all the supported driver files released with the library is located.
- Under Microchip directory, the Include directory will have another Graphics directory that will contain all the header files for the Graphics Library. All other header files will be located under Microchip/Include directory.

Step 2: Creating the MPLAB project

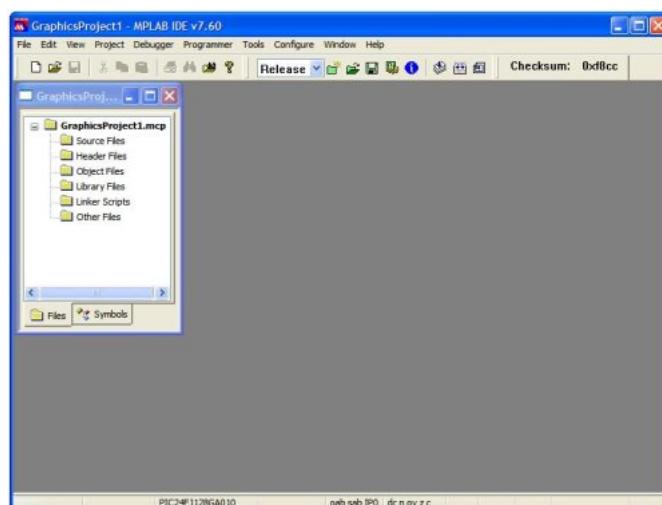
1. Open MPLAB Project.
2. Go to **Project** tab and select **New**



3. **New Project** dialog box will appear. In this dialog box enter your **Project Name** and **Project Directory**. Click **OK**. Project will be created or prompted to create the directory if your project directory does not exist (refer to figure below).



4. The MPLAB **Project** view will appear as soon as the project is created. If it does not appear go to **View** tab and click **Project**.



5. Right click on the **Source Files** and click **Add Files**. Browse to *MyApplication->Microchip->Graphics* directory. Select appropriate c files and click **Open**. The C files that you select will depend on the application that you will be creating. The following tables shows the minimum files required for each layer of the Graphics Library.

Display Driver Layer	
DisplayDriver.h	Header that contains required APIs for display driver to be compatible with Microchip Graphics Library.
gfxepmp.c gfxepmp.h	Required if using display drivers for external display controller that will use the Enhanced Parallel Master Port (EPMP) for display controller communications.
gfpmp.h	Required if using display drivers for external display controller that will use the Parallel Master Port (PMP) for display controller communications.
gftcon.h	Required if using Microchip supplied display drivers that requires timing controller initialization. If creating your own display controller driver, you may integrate the timing controller initialization into your display driver file.
Display Driver c and h files	These are the display driver files needed. These can be your own created display driver or Microchip supplied display drivers distributed with the Graphics Library.

Primitive Layer	
Primitive.c Primitive.h	These two files implements the Primitive Layer of the Microchip Graphics Library.
Transitions.c Transitions.h Transitions_weak.c	Implements the extended primitive functions for screen transitions. The Screen Transition features are only supported by the Epson driver (S1D13517.c and S1D13517.h) and the PIC24FJ256DA210 driver (mchpGfxDrv.c and mchpGfxDrv.h)

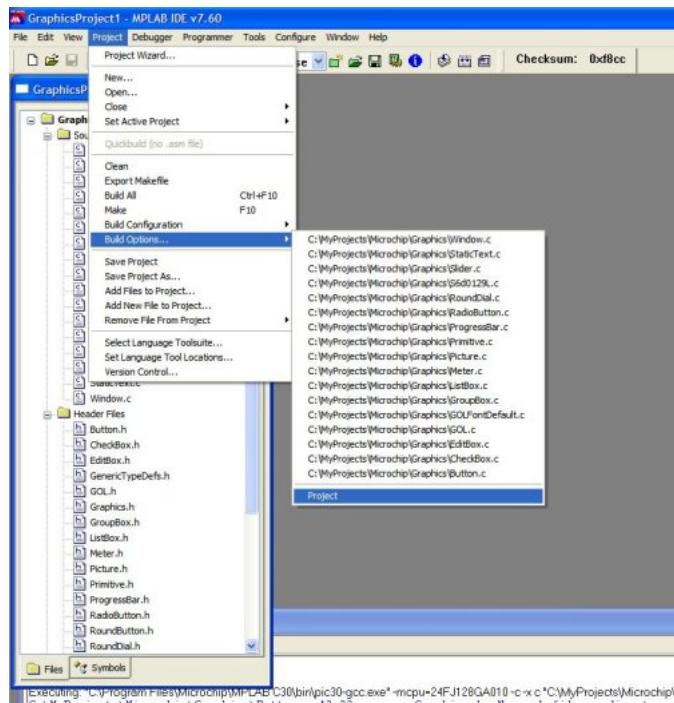
Graphics Object Layer	
GOL.c GOL.h	These two files along with the individual widget files implements the Graphics Object Layer of the Microchip Graphics Library.
GOLFontDefault.c	This is the default font used for the Graphics Object Layer.
GOLSchemeDefault.c	This is the default style scheme used for the Graphics Object Layer.
Widget files (example: Button.c Button.h)	Required files when using the widgets in the Graphics Object Layer.

Configuration	
Graphics.h	Required file that loads the different components of the Graphics Library.
GraphicsConfig.h	Required file to determine the features and modes that are enabled in the Graphics Library.
HardwareProfile.h	Required file when running demos distributed with the Microchip Graphics Library or using drivers distributed in "Board Support Package".

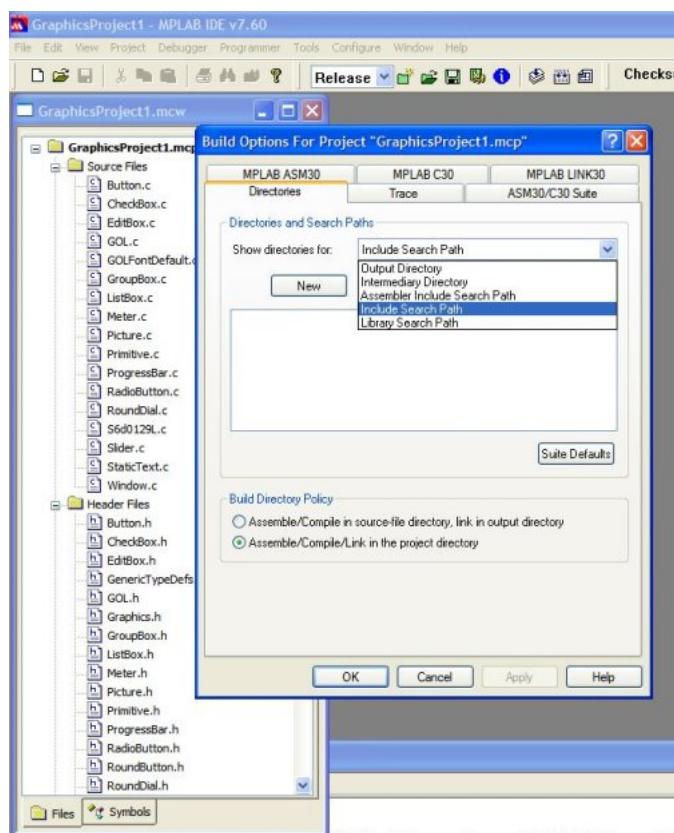
6. Right click on the **Header Files** and click **Add Files**. Browse to *MyApplication->Microchip->Include* directory. Select **GenericTypeDefs.h** file and click **Open**.
7. Right click on the **Header Files** and click **Add Files**. Browse to *MyApplication->Microchip->Include->Graphics* directory. Select all the h files and click **Open**. The header files to include will depend on the application as described in 5.

Step 3: Setting the project directories.

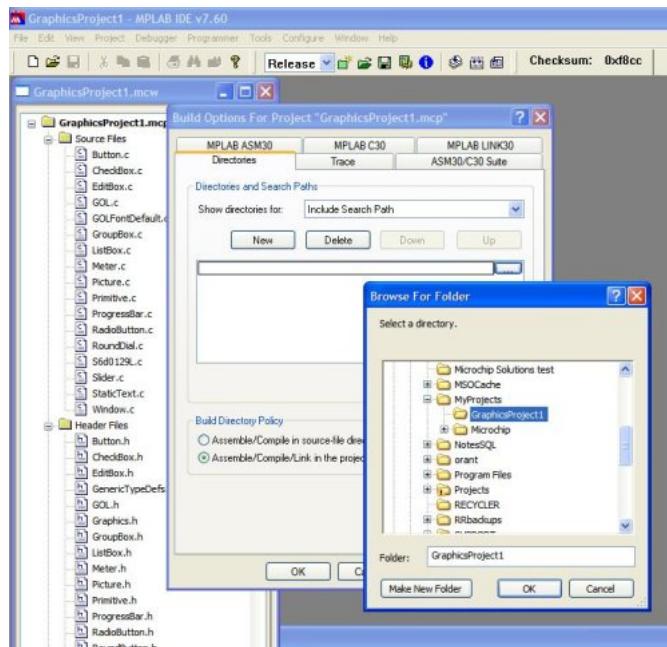
1. In MPLAB Project, go to **Project** tab and point to **Build Options**.



2. A menu will appear showing all your files and the **Project** option. Select the **Project** option. The **Build Options** window will open.
3. Select the **Directories** tab.



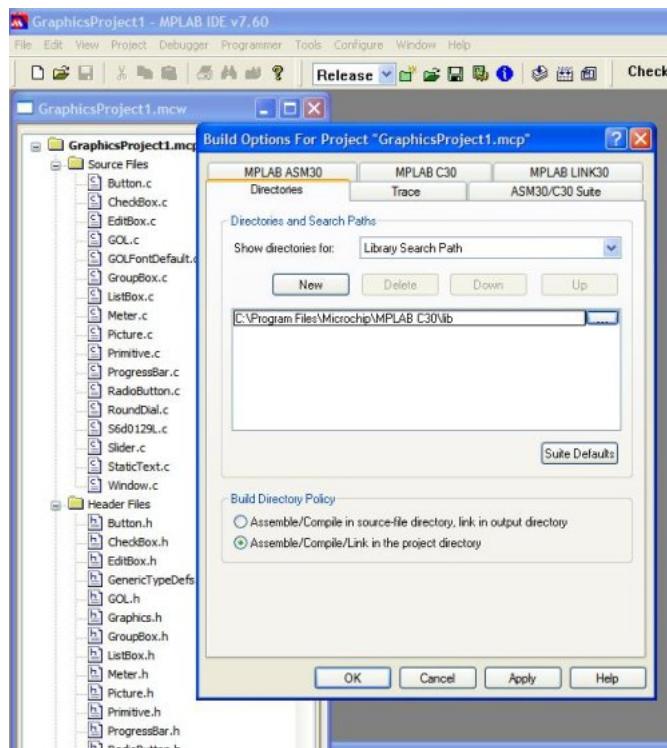
4. In the **Directories** tab, **Directories and Search Paths** group box and **Show directories for:** select **Include Search Path**. Click **New**.
5. Click the button with (...) and browse and select your project directory. Click **OK**. This will add your project directory.



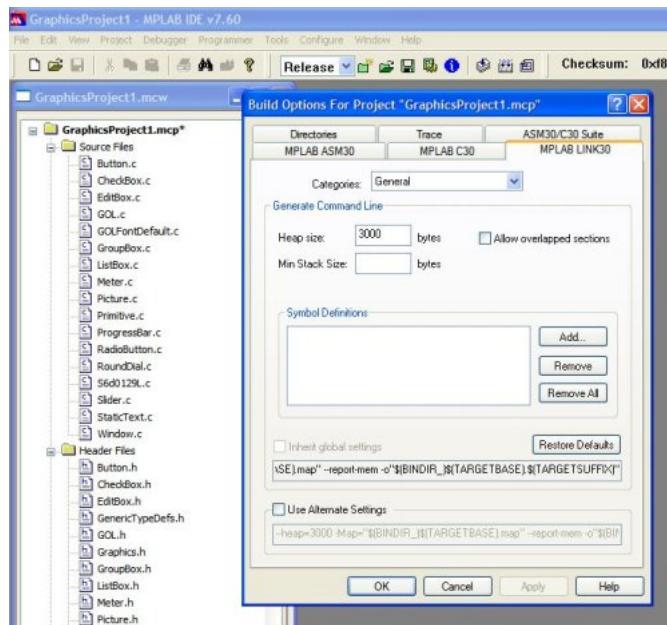
6. Click New again for each of the following paths:

1. .
 2. ..\..\Microchip\Include
 3. ..\..\Board Support Package (do this only if you are using the drivers supplied in the "Board Support Package" director_)
- 7. Directories and Search Paths** group box and **Show directories for:** select **Library Search Path**. Click **New**.

8. Enter the path to the MPLAB C30 lib. For example: C:\Program Files\Microchip\MPLAB C30\lib. Click **OK**.



9. Change from the **Directories** tab to **MPLAB LINK 30** tab. In the **Generate Command Line** group box enter 3000 in the **Heap Size** setting.



10. Click **Apply** and then **OK**.

Step 4: Create your application files.

1. In MPLAB Project, select **File** tab and click **New**. The file editor will open.

2. Create your **GraphicsConfig.h** file. For now add the following lines:

```
#define USE_NONBLOCKING_CONFIG // Comment this line to use blocking configuration
#define USE_BUTTON // Enable Button Object.
#define USE_SLIDER // Enable Slider or Scroll Bar Object.

#define USE_FONT_FLASH // Support for fonts located in internal flash
#define USE_BITMAP_FLASH // Support for bitmaps located in internal flash
```

The file name is important. If the file name is changed, library will not compile properly.

3. Save the file into your **GraphicsProject1** directory.

4. Similarly, create your application header file code by following steps 1 and 2. For simplicity just use this code for now:

```
#define SYSCLK 32000000 // 8MHz x 4PLL Oscillator frequency
// includes
#include <p24Fxxxx.h>
#include "GenericTypeDefs.h"
#include "Graphics.h"
```

5. Save the file in your **GraphicsProject1** directory.

6. After saving the two header files, add these files to your project.

7. Create your application code. For simplicity just use this code for now:

```
#include "GraphicsProject.h" // header file you saved earlier
// Configuration bits
_CONFIG2(FNOSC_PRIPLL & POSCMOD_XT) // Primary XT OSC with PLL
_CONFIG1(JTAGEN_OFF & FWDTEN_OFF) // JTAG off, watchdog timer off

int main(void){

    GOL_MSG msg; // GOL message structure to interact with GOL
    GOLInit(); // initialize graphics library &
    BtnCreate( 1, // object's ID
               20, 160, 150, 210, // object's dimension
               0, // radius of the rounded edge
               BTN_DRAW, // draw the object after creation
               NULL, // no bitmap used
               "LEFT", // use this text
```

```

        NULL);           // use alternative style scheme
BtnCreate( 2,
    170, 160, 300, 210,
    0,
    BTN_DRAW,
    NULL,
    "RIGHT",
    NULL);
SldCreate(3,
    20, 105, 300, 150,
    SLD_DRAW,           // draw the object after creation
    100,               // range
    5,                 // page
    50,                // initial position
    NULL);             // use default style scheme
while(1){
    if (GOLDraw()) {           // Draw GOL object
    }
}
}

WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg){
    return 1;
}

WORD GOLDrawCallback(){
    return 1;
}

```

8. Save the file in your **GraphicsProject1** directory.

9. After saving the application file, add the file to your project.

Step 5 Now build your project. To build go to MPLAB **Project** tab and click **Build All**. Notice there will be some warnings that may come out in the build log. This is due to the fact that we have not used most of the Objects. We only used the Slider (█) and Button (█).

Step 6: Downloading your application - To download your generated code to the Explorer 16 board follow the steps below.

1. In the MPLAB **Programmer** tab click **Select Programmer**.
2. Select **MPLAB ICD2 or REAL ICE** depending on your setup.
3. After connection and testing is done go to **Programmer** tab and click **Program**.

Note that you can also do the steps in the Downloading the Demos (█) sub-section in the Getting Started (█) section to download your application to the Explorer 16 board using your generated hex file.

9.2 Changing the default Font

The library comes with the default font (Gentium 18). This font can be changed in two ways.

Description

This instructions assumes that you have performed the conversion of your raster font or True Type font into C file. For the conversion of your raster font please see the help file of the Graphics Resource Converter utility that comes with the Graphics Library.

There are two ways to replace the default font in the Graphics Library.

Renaming User Font to GOLFonDefault (█):

1. Open the generated font file (generated by the "Graphics Resource Converter" utility) and change the font structure name to **GOLFonDefault** (█) (see figure below).

```

extern const char L1191258975[] __attribute__((aligned(2)));
//FONT STRUCTURE. FONT NAME CAN BE CHANGED HERE.
const struct{short mem; const char* ptr;} GOLFonDefault =
{0,L1191258975};
const char L1191258975[] __attribute__((aligned(2))) = {

```

2. Remove the file **GOLFONDefault.c** in the project and replace it with your own font file that contains the **GOLFonDefault** (█).
3. Build and test your new font.

Replacing the GOLFonDefault (█) with any user defined fonts

1. Open the **GraphicsConfig.h** file and add the following lines:

```
#define FONTDEFAULT yourFontName
```

2. In the project, add the generated font file (generated by the "Graphics Resource Converter" utility).

3. Build and test your new font.

In this method, GOL.h and GOLFonDefault.c files checks if FONTDEFAULT (█) is defined. If it is, it skips the declaration of the GOLFonDefault (█) and uses the user defined font.

9.3 Advanced Font Features

Fonts used in the library can be configured to use anti-aliasing and extended glyph support.

Description

Font Anti-Aliasing

Anti-aliasing is a technique used to make the edges of text appear smooth. This is useful especially with characters like 'A', 'O', etc which has slant or curved lines. Since the pixels of the display are arranged in rectangular fashion, slant edges can't be represented smoothly. To make them appear smooth, a pixel adjacent to the pixels is painted with an average of the foreground and background colors as depicted in Figure 1.

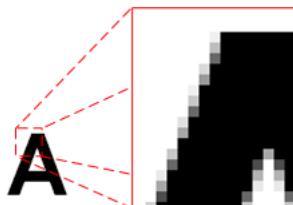


Figure 1: Font with Anti-Aliasing

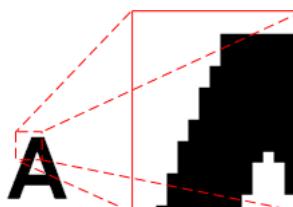


Figure 2: Font with No Anti-Aliasing

When anti-aliasing is turned off, the pixels abruptly changes from background color to foreground color shown in Figure 2. To implement anti-aliasing, adjacent pixels transitions from background to foreground color using 25% or 75% mid-color values from background to foreground colors. This feature in fonts will require roughly twice the size of memory storage

required for font glyphs with no anti-aliasing.

Since the average of foreground and background colors needs to be calculated at runtime, the rendering of anti-aliased fonts take more time than rendering normal fonts. To optimize the rendering speed, a macro named GFX_Font_SetAntiAliasType (🔗) is available where anti-alias type can be set to ANTIALIAS_OPAQUE (🔗) or ANTIALIAS_TRANSLUCENT (🔗).

- ANTIALIAS_OPAQUE (🔗) (default after initialization of graphics) - mid colors are calculated once while rendering each character which is ideal for rendering text over a constant background.
- ANTIALIAS_TRANSLUCENT (🔗) - the mid values are calculated for every necessary pixel and this feature is useful while rendering text over an image or on a non-constant color background.

As a result, rendering anti-aliased text takes longer with ANTIALIAS_TRANSLUCENT (🔗) type than compared to ANTIALIAS_OPAQUE (🔗) type.

To use anti-aliasing, enable the compiler switch #define USE_ANTIALIASED_FONTS (🔗) in the GraphicsConfig.h file and enable the anti-alias checkbox in the Graphics Resource Converter (GRC) tool while selecting the font.

Note: Even when anti-aliasing is enabled, normal fonts can be used without the antialias effect.

Extended Glyphs

Extended glyphs are needed to render characters of certain languages which use more than one byte to represent a single character. For example: Asian languages like Thai, Hindi, etc. In these character set, more than one glyph overlaps each other to form a single character of that language as shown in Figure 3. To use this feature, enable the Extended Glyph checkbox in the Graphics Resource Converter (GRC) tool while selecting the font.

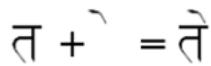


Figure 3: Example of a Character that is Formed by Two Overlapping Glyphs

Note: The fonts used with extended glyphs are normal ANSI fonts and not Unicode fonts.

9.4 Using Primitive Rendering Functions in Blocking and Non-Blocking Modes

Basic rendering functions such as Line(), Rectangle (🔗)(), Circle (🔗)() etc are referred to as functions in the Graphics Primitive Layer. These functions can also be implemented in the device driver layer if the display device supports hardware acceleration of the function. Applications that directly calls these functions can take advantage of the hardware accelerated primitives. How these functions are used will depend on the "Configuration Setting (🔗)".

Description

All primitive rendering functions returns a status.

- 0 – when the primitive was not successfully rendered
- 1 – when the primitive was successfully rendered

When using Graphics Library you can enable the non-blocking mode when calling drawing/rendering functions. This is done by adding this line in your GraphicsConfig.h file:

```
#define USE_NONBLOCKING_CONFIG // Comment this line to use blocking configuration
```

When using a display controller with hardware accelerated primitives (like SSD1926 which is on the Graphics PICtail™ Plus Board Version 3 (AC164127-3) faster primitive rendering on Line(), Rectangle (🔗)() and Bar (🔗)() functions will be

performed. Compiling with the Blocking or Non-Blocking mode set will still use the accelerated primitives but the application code directly calling the primitive functions will have to be coded accordingly.

To explain the two modes when directly calling the primitive functions please take a look at the example below.

Case 1: USE_NONBLOCKING_CONFIG (■) disabled

```
// all primitives are blocking calls
Line(a,b);
Rectangle(c,d,e,f);
Bar(c+2, d+2, e-2, f-2)
```

Case 2: USE_NONBLOCKING_CONFIG (■) enabled

```
// all primitives are non-blocking calls
while(!Line(a,b));
while(!Rectangle(c,d,e,f));
while(!Bar(c+2, d+2, e-2, f-2));
```

If the while check is not in place, it possible that the only primitive that you will see in the screen is the Line().

For case 2, one can also be creative in the application code and implement some form of non-blocking scheme and make use of the time while waiting for the primitives to render.

Another example for case 2:

```
WORD DrawMyFigure(a, b, c, d, e, f)
{
    typedef enum {
        DRAW_LINE,
        DRAW_RECT,
        DRAW_BAR,
    } DRAW_MYFIGURE_STATES;
    static DRAW_MYFIGURE_STATES state = DRAW_LINE;

    if(IsDeviceBusy()) // checks if the hardware is still busy
        return 0;

    switch(state){
        case DRAW_LINE:
            if (!Line(a, b))
                return 0;
            state = DRAW_RECT;
        case DRAW_RECT:
            if (!Rectangle(c,d,e,f))
                return 0;
            state = DRAW_BAR;
        case DRAW_BAR:
            if (!Bar(c+2, d+2, e-2, f-2));
                return 0;
            state = DRAW_LINE;
        return 1;
    }
}
```

This non-blocking code can be used in the application and the application can do other tasks whenever DrawMyFigure() returns 0. Application should call DrawMyFigure() again until it return a 1 signifying that the Line, Rectangle (■) and Bar (■) were drawn successfully.

9.5 Using Microchip Graphics Module Color Look Up Table in Applications

Utilizing the Color Look Up Table (CLUT) of the Microchip Graphics Module saves memory for both storage and display buffer. This short instructional manual outlines the procedure to create source code files to use the CLUT of the Microchip

Graphics Module and enable the Microchip Graphics Library to use the hardware feature.

Description

Apart from regular RGB scheme of representing colors, colors may also be represented using a Color Look Up Table (CLUT) also called Palette table, where there is a table of colors and the color is specified by the index of the table. Depending on the size of the table, the bits used to represent the index will vary. For example 256 entries of RGB (8 bit index), 16 entries of RGB (4 bit index), 4 entries of RGB (2 bit index) and 2 entries of RGB (1 bit index). This scheme is mainly used to save memory. See Figure-1 for an example of 16-entry (4-bit) CLUT where a shade of Green is represented by an index value of 3 consuming 4-bits. To figure the memory requirement for a give screen size at a color depth, bits per pixel, the follow equations is used: total number of pixels x bits per pixel / 8 bits per byte. For example the memory required for a 320x240 with 16 BPP screen; $(320 \times 240) \times (16 / 8) = 153,600$ Bytes. If only 16 different colors are used in the screen, then instead of using raw RGB, a 16-entry (4-bit) CLUT may be used requiring a memory of $(320 \times 240) \times (4/8) = 38,400$ bytes; thereby saving 75% of memory.

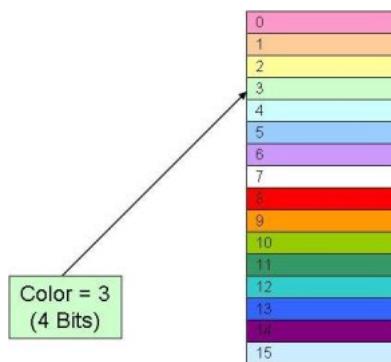


Figure 1. A 16-Entry (4-bit) Color Look Up Table

If the display driver hardware supports CLUT, the index values are translated to the RGB values by the hardware automatically when the signals are sent out to the display.

If the CLUT is enabled, since a CLUT affects the whole screen, all the color patterns like basic shapes and images which are displayed on the screen must use the same CLUT. It means that color defines like RED (█), GREEN (█), etc... must use the index values instead of the absolute RGB values. The bitmap images used must also use the same CLUT in order to appear properly on the screen. Note that the chosen CLUT length (1, 4, 16, or 256) must accommodate all the different colors needed by a screen. The following section explains how to create such a CLUT for a screen.

Creating CLUT

Creating CLUT has 2 steps:

1. Creating a part of CLUT manually.
2. Filling the remaining part with the colors of the images used.
3. Using the Graphics Resource Converter to generate a CLUT based on the images to convert.

Creating a part of CLUT manually - Since users need to select specific colors for the shapes and the widgets, they must enter these specific colors in the CLUT. This can be done by manually creating a new CLUT table using a text editor. Create a text file and save it with a .gpl extension with the form:

```
GIMP Palette
Name: 16_colors
Columns: 4
#
0 0 0 BLACK
0 0 128 BLUE
128 0 0 RED
0 0 0 Unused
0 0 0 Unused
```

The second line specifies the name of the palette which must be unique. The palette table data starts with line 5 which represent index 0 with the RGB values and a caption. The number of such data lines must be equal to the length of the

CLUT. In the example, only 3 colors are used.

Filling the remaining part with the colors of the images used - Suppose 256 entries CLUT is being used, since 3 colors are manually set, only $256 - 3 = 253$ entries are available for the images to be displayed on the same screen. If more than one image is used on the screen, it further implies that

- All images on the screen must use the same CLUT table.
- Different colors used for all the images along with the fixed colors must not exceed the size of the CLUT table being used.

If the source images are of RGB type, they must be converted into CLUT based images. This can be done using free PC tools like GIMP (www.gimp.org).

The following steps shows how to convert the images using GIMP:

Step 1:

Create a new image with sufficient size to paste all the images required on the screen using *File->New* in the GIMP (see Figure-2).

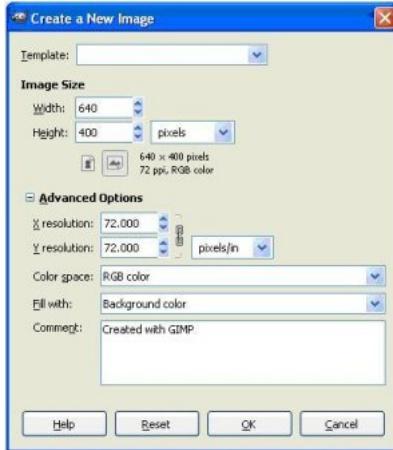


Figure 2. Creating New File in GIMP

Step 2:

Copy and paste all the images to be displayed on the screen into this image like in Figure 3. If 256 colors is enough for the entire application, a single CLUT can be used. If not multiple CLUT can be used. Each CLUT can be configured to use one or more screens. In this case, switching from one screen to another may require re-initializing the hardware CLUT entries before the screen is displayed.



Figure 3. Images Used in One Screen

Step 3:

Set the mode to CLUT by selecting *Image->Mode->Indexed* in the GIMP. Select to generate optimum palette with $256 - 3 = 253$ entries (because we already have 3 fixed entries) as shown in Figure 4 and save it as a BMP (e.g. Collage.bmp) image by selecting *File->SaveAs* menu.

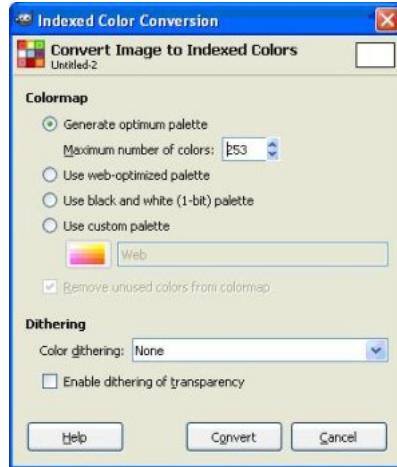


Figure 4. Generate a Palette Table (CLUT)

Step 4:

The next step is to extract the CLUT from the generated image. To do that, go to the palette selection mode by selecting *Image->Mode->RGB* and then again *Image->Mode->Indexed*. Select Use Custom Palette and open the palette selection dialog as shown in Figure 5 and import the previously saved bitmap image (Collage.bmp) as shown in Figure 6 and Figure 7. The palette file will be created in the [Home Folder]\gimp-x.y\palettes folder as a *.gpl file.

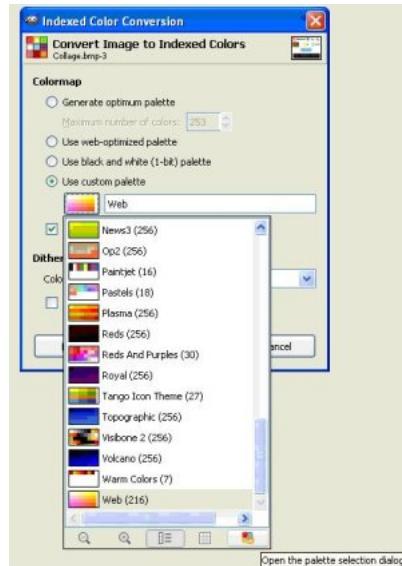


Figure 5. Palette Selection Dialog



Figure 6. Import Palette Dialog

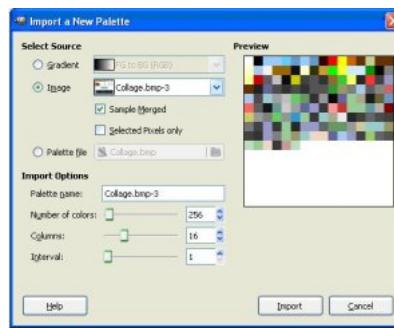


Figure 7. Import Collage Bitmap

Step 5:

Manually edit this and add the fixed color entries from the previously stored fixed palette file. (Manual step). Now a master palette file is created which has to be used in the application.

Step 6:

Open individual images in GIMP and apply this master palette to all of them through *Image->Mode->Indexed* and selecting Use Custom Palette. Select the master palette which was generated and save the images. This will make all the images palette ready.

Step 7:

The next step is to convert this Palette.gpl and the images into the format recognizable by the Microchip Graphics Library. Start the Microchip's Graphics Resource Converter tool and enable the C30 Build (palette support is currently on selected PIC24F devices only) mode as shown in Figure 8. Open the master palette file previously generated by pressing Add Palette button and save it as a .c file (for storing in internal flash) or as a .hex file (for storing in external memory) similar to bitmap or font conversion as shown in Figure 10.

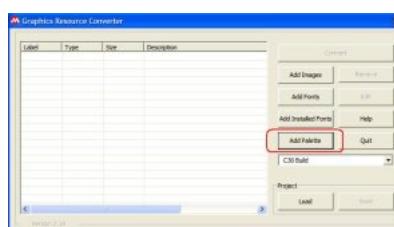


Figure 8. Load Palette

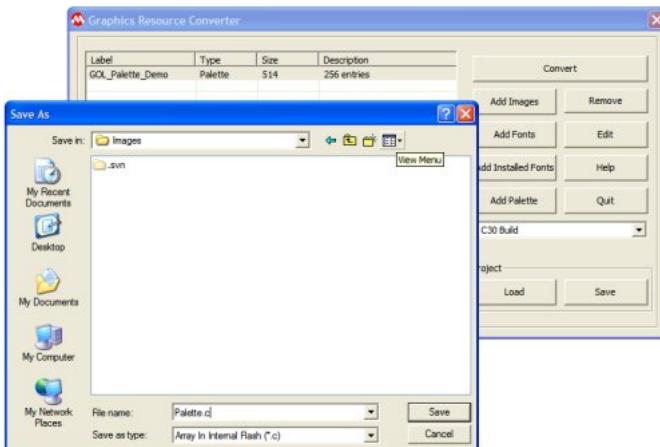


Figure 9. Convert Palette

Step 8:

Convert all the palette ready images to C file (*.c) or Hex file (*.hex) by pressing the Add Images button.

Step 9:

Import the palette with the extern statement like in “extern const PALETTE_FLASH (█) _GOL_Palette_Demo;” in the application. See MainDemo.c in “Graphics Object Layer Palette Demo”. Set the palette using the APIs SetPaletteBpp (█)() and SetPalette (█)() and then enable the palette using the API EnablePalette (█)() as shown in the below code example.

```
GOLInit();
SetPaletteBpp(8);
SetPalette((void*)&_GOL_Palette_Demo, 0, 256);
EnablePalette();
```

Step 10:

Import the images as usual and use them after setting and enabling the palette.

See “Graphics Object Layer Palette Demo” as a practical example.

Using the Graphics Resource Converter to generate a CLUT based on the images to convert - The Graphics Resource Converter will generate a CLUT based on the images to be converted. Please refer to the Graphics Resource Converter help file for more information.

9.6 Converting Images to Use a Common Palette in GIMP

This manual describes how to convert an image or a set of images to use a common palette in GIMP.

Description**INTRODUCTION**

Some controllers have predefined palettes associated with them. The palette's colors can not be altered. When converting images, it is desired to have the images use the controller's predefined color palette. For example, a controller may have a grayscale palette of 16 colors. An image may use a palette of 16 grayscale colors, but the palette may define grayscale colors that are not the same as the controller's palette. By using the GNU Image Manipulation Program, GIMP, images can be converted using a palette matching the grayscale colors of the controller. After converting the image to use this palette, it

can be converted by the Graphics Resource Converter, GRC, to be used by Microchip's Graphics Library.

DOWNLOADS

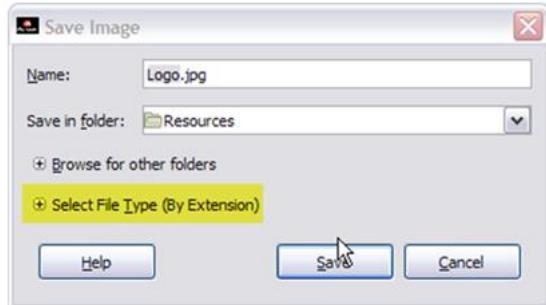
The following are helpful websites for downloaded the tools and firmware needed:

- GIMP 2.6 – GNU Image Manipulation Program (www.gimp.org)
- Graphics Library – This library is part of the Microchip Application Libraries (www.microchip.com/mla)

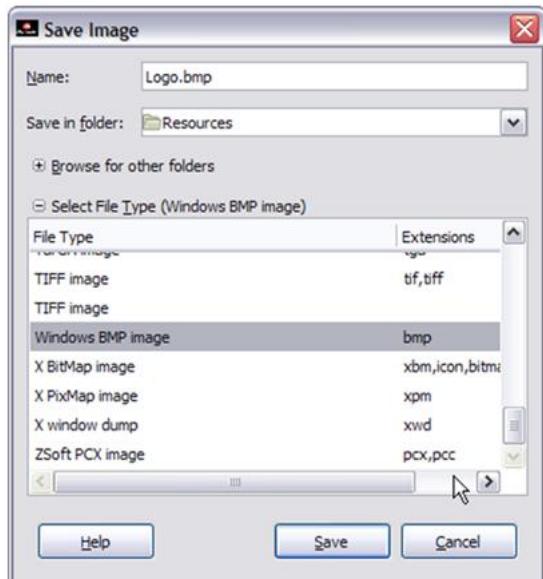
CONVERTING AN IMAGE

Follow these steps to convert an image to use a predefined color palette (for this example, the palette used has 16 grayscale colors):

1. Open the GIMP application.
2. Load the image. FILE->Open
 - If the image is not a Bitmap, you will need to save it as one.
1. FILE-Save as...
2. Choose the Select File Type (By Extension)



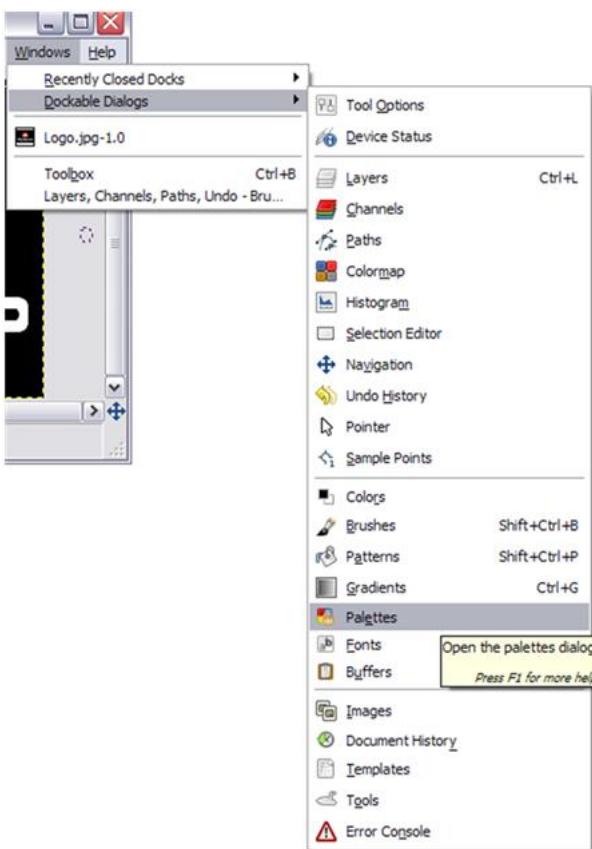
3. Select Windows BMP Image



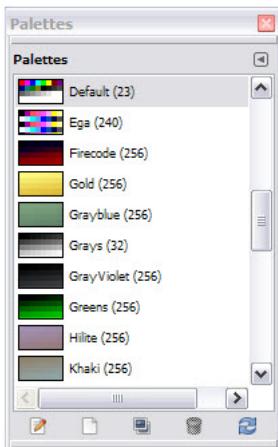
4. Select Save
5. Select Save under the Save as BMP dialog



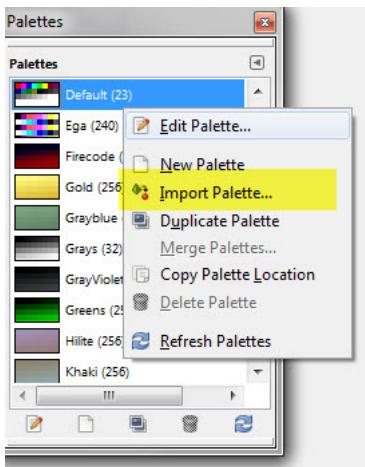
3. Select WINDOWS->Dockable Dialogs->Palettes



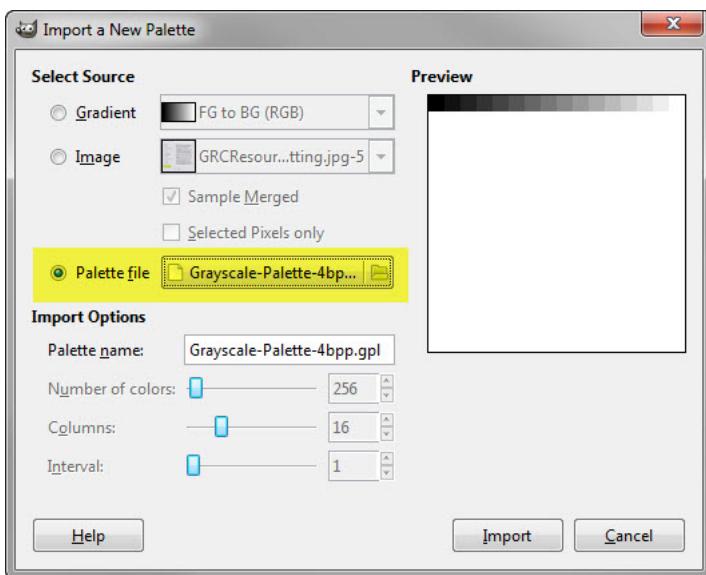
- The Palette Dialog will appear.



4. In the Palette Dialog Select Import Palette. This can be done by a right click on any of the Palettes on the Palette Dialog.



- The Import Palette Dialog will appear.



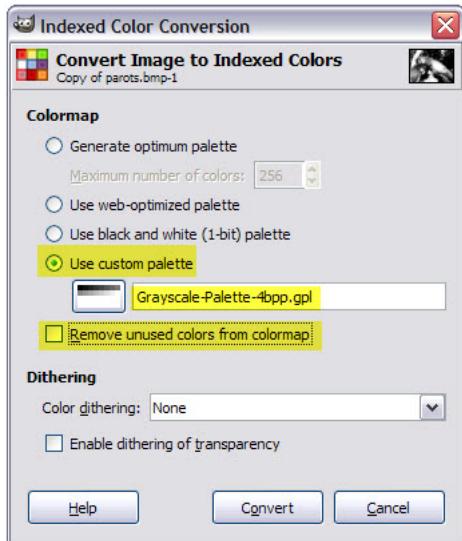
5. Select the Palette file radio button.

- Select the palette file .gpl (For example: Grayscale-Palette-4bpp.gpl) (see image in Step 4).
- Select the Import button.
- The palette, Grayscale-Palette-4bpp, will show up in the Palette Dialog.
- Select IMAGE->Mode->Indexed...



- In the Indexed Color Conversion dialog, select the Use custom palette radio button

- Type the name of the palette.
- **IMPORTANT:** Make sure that the "Remove unused color from colormap" is unchecked.
- Select Convert



9. The image will now be converted using a 16 grayscale color palette.

10. Save the image as a Bitmap.

This image can now be converted by the GRC for use with the Microchip Graphics Library.

Here is an example of a palette file *.gpl (Grayscale-Palette-4bpp.gpl)

```
GIMP Palette
Name: Greyscale Palette 4bpp
Columns: 0
#
0 0 0 BLACK
17 17 17 Untitled
34 34 34 Untitled
51 51 51 Untitled
68 68 68 Untitled
85 85 85 Untitled
102 102 102 Untitled
119 119 119 Untitled
136 136 136 Untitled
153 153 153 Untitled
170 170 170 Untitled
187 187 187 Untitled
204 204 204 Untitled
221 221 221 Untitled
238 238 238 Untitled
255 255 255 WHITE
```

9.7 How to Define Colors in your Applications

In most cases, the application will define its own set of colors and not use the default colors that comes with the Graphics Library. This section shows an example on how to do it.

Description

To override or define a new set of colors follow this steps:

1. Create a new color header file. The color values and data types will depend on the GFX_COLOR type used. This data

type is defined by the COLOR_DEPTH (█) macro. See COLOR_DEPTH (█) for details.

2. In the application code, include the created color header file ahead of the #include "Graphics/Graphics.h". When Graphics.h includes the gfxcolors.h it will ignore color macros that has been defined already in the new color header file.

In the GOL Palette Demo, there is an example on how it is done. In that project there is a file in the application (or project directory) named PaletteColorDefines.h. This file contains all the color definition used in the demo. The main header file of the demo (Main.h) includes the PaletteColorDefines.h file ahead of the Graphics Library header files. Since the PaletteColorDefines.h declared the color values, the macros redefined in gfxcolors.h will be ignored. How is this done? If you look at the gfxcolors.h file you will notice that all colors defined have a check (for example BLACK (█)):

```
#ifndef BLACK (█)
#define BLACK (█) 0
#endif
```

So if BLACK (█) is defined previously, the definition in gfxcolors.h will not take effect.

9.8 Connecting RGB data bus

Display glasses will require 24 bit or 18 bit RGB color data. How do you do the connection when your display controller only puts out 16 bit RGB data bus?

Description

To connect the 16-bit RGB data bus to a 24-bit or 18-bit RGB data bus of a display glass follow the recommended connection to evenly spread the color coverage.

16-bit (5-6-5) RGB Display Controller Data Bus	18-bit (6-6-6) RGB Display Data Bus	24-bit (8-8-8) RGB Display Data Bus
Red[4:0]		
Controller Red[4:0]	Display Red[5:1]	Display Red[7:3]
Controller Red[4]	Display Red[0]	Display Red[2:0]
Green[4:0]		
Controller Green[5:0]	Display Green[5:0]	Display Green[7:2]
Controller Green[5]	Display Green[5]	Display Green[1:0]
Blue[4:0]		
Controller Blue[4:0]	Display Blue[5:1]	Display Blue[7:3]
Controller Blue[4]	Display Blue[0]	Display Blue[2:0]

To illustrate the connection take the Red for example:

To connect the Display Controller Red Data Bus to the Red data bus of a RGB Glass with 18 bit color data bus.

Connect the 5 Red signals from the display controller to the most significant bits of the glass red signals.

Controller Red[4:0] -> Display Red[5:1]

The remaining Display Red[0] signal will be connected to the most significant bit of the display controller red.

Controller Red[4] -> Display Red[0]

To connect the Display Controller Red Data Bus to the Red data bus of a RGB Glass with 24 bit color data bus.

Connect the 5 Red signals from the display controller to the most significant bits of the glass red signals.

Controller Red[4:0] -> Display Red[7:3]

The remaining Display Red[2:0] signals will be connected to the most significant bit of the display controller red.

Controller Red[4] -> Display Red[2]

Controller Red[4] -> Display Red[1]

Controller Red[4] -> Display Red[0]

Doing this spreads out the color coverage while at the same time pure black and pure white colors are achieved.

9.9 Adding New Device Driver

This is a summary of the requirements to add a new device driver.

Description

Adding a new display device driver requires the following functions and macros to be implemented. Please refer to the API section of the Device Driver Layer for details.

Function/Macro	Description
ResetDevice (█)()	Initializes the display device.
GetMaxX (█)()	Returns the maximum x-coordinate for the display.
GetMaxY (█)()	Returns the maximum y-coordinate for the display.
SetColor (█)()	Sets the current drawing color.
GetColor (█)()	Returns the current drawing color.
SetActivePage (█)()	Sets the current active graphic page (optional).
SetVisualPage (█)()	Sets the current visual graphic page (optional).
PutPixel (█)()	Modifies pixel on the screen.
GetPixel (█)()	Returns the pixel color.
PutImage (█)()	Renders an image on the screen. This function is dependent on the color format used.
SetClipRgn (█)()	Set the current clipping region borders.
GetClipLeft (█)(), GetClipTop (█)(), GetClipRight (█)(), GetClipBottom (█)()	Returns the left, top, right and bottom clipping borders.
SetClip (█)()	Enables or disables the clipping region.
IsDeviceBusy (█)()	Checks if the display controller is busy executing the previous rendering operation.
SetPalette (█)()	Sets the palette register of the device.

Adding new Display Device Drivers

The DisplayDriver.h file should be used as a guide to make your new driver compatible with the Microchip Graphics Library. All the API's defined in this header file are required functions to be implemented in the driver. There are portions of that file that states optional functions. These functions are not needed to interface to the Graphics Library. These are only implemented if the display controller used has hardware features that can implement these functions.

The best way to implement this is to try to find the nearest existing driver and modify the C and H files. Most graphics controllers has a lot of control registers to be initialized. Values programmed into the registers depends on the specification of the LCD glass used. If LCD module has a built-in graphics controller, initialization code for the glass can be found in the LCD specifications or get this information from the manufacturer.

10 References

1. "[MPLAB C32 C COMPILER USER'S GUIDE](#)" (DS51686), Microchip Technology Incorporated.
2. "[MPLAB C COMPILER FOR PIC24 MCUs AND dsPIC DSCs USER'S GUIDE](#)" (DS51284), Microchip Technology Incorporated.
3. Microchip Application Note [AN1136](#), "How to Use Widgets in Microchip Graphics Library" (DS01136), Microchip Technology Incorporated.
4. Microchip Application Note [AN1182](#), "Fonts in the Microchip Graphics Library" (DS01182), Microchip Technology Incorporated.
5. Microchip Application Note [AN1227](#), "Using a Keyboard with the Microchip Graphics Library" (DS01227), Microchip Technology Incorporated.
6. Microchip Application Note [AN1246](#), "How to Create Widgets in Microchip Graphics Library" (DS01246), Microchip Technology Incorporated.
7. HIF 2131 – Designing with Microchip Graphics Library, Microchip Regional Training Center web site (www.microchip.com/rtc).

Index

_BMPDECODER structure 479
 _GIFDECODER structure 479
 _JPEGDECODER structure 480
 _pDefaultGolScheme variable 344
 _pGolObjects variable 344
 _pObjectFocused variable 345

A

AC_DISABLED macro 79
 AC_DRAW macro 78
 AC_HIDE macro 79
 AC_PRESSED macro 79
 AC_TICK macro 79
 AcCreate function 80
 AcDraw function 81
 AcSetHour function 83
 AcSetMinute function 83
 AcSetSecond function 84
 Adding New Device Driver 503
 Advanced Display Driver Features 415
 Advanced Font Features 490
 Advanced Font Features Selection 45
 Alpha Blend Option 47
 Alpha Blending 415
 Alpha Blending Functions 377
 AlphaBlendWindow function 379
 Analog Clock 77
 Analog Clock States 78
 ANALOGCLOCK structure 84
 Anti-Alias Type 355
 ANTIALIAS_OPAQUE macro 355
 ANTIALIAS_TRANSLUCENT macro 355
 Application Configuration 51
 Arc function 369

Bar function 365
 BarGradient function 356

Bevel function 371
 BevelGradient function 358
 Bitmap Functions 380
 Bitmap Settings 385
 Bitmap Source 385
 BITMAP_HEADER structure 384
 BLACK macro 408
 BLUE macro 408
 BRIGHTBLUE macro 408
 BRIGHTCYAN macro 408
 BRIGHTGREEN macro 409
 BRIGHTMAGENTA macro 409
 BRIGHTRED macro 409
 BRIGHTYELLOW macro 409
 BROWN macro 409
 BTN_DISABLED macro 88
 BTN_DRAW macro 88
 BTN_DRAW_FOCUS macro 88
 BTN_FOCUSED macro 88
 BTN_HIDE macro 88
 BTN_NOPANEL macro 90
 BTN_PRESSED macro 89
 BTN_TEXTBOTTOM macro 89
 BTN_TEXTLEFT macro 89
 BTN_TEXTRIGHT macro 89
 BTN_TEXTTOP macro 89
 BTN_TOGGLE macro 90
 BTN_TWOTONE macro 90
 BtnCreate function 90
 BtnDraw function 92
 BtnGetBitmap macro 94
 BtnGetText macro 93
 BtnMsgDefault function 95
 BtnSetBitmap macro 95
 BtnSetText function 93
 BtnTranslateMsg function 96
 Button 85
 Button States 87
 BUTTON structure 98

C

CB_CHECKED macro 137

CB_DISABLED macro 138
CB_DRAW macro 138
CB_DRAW_CHECK macro 138
CB_DRAW_FOCUS macro 138
CB_FOCUSED macro 138
CB_HIDE macro 139
CbCreate function 139
CbDraw function 140
CbGetText macro 141
CbMsgDefault function 142
CbSetText function 141
CbTranslateMsg function 143
CH_3D_ENABLE macro 102
CH_BAR macro 103
CH_BAR_HOR macro 103
CH_CLR0 macro 132
CH_CLR1 macro 132
CH_CLR10 macro 134
CH_CLR11 macro 134
CH_CLR12 macro 135
CH_CLR13 macro 135
CH_CLR14 macro 135
CH_CLR15 macro 135
CH_CLR2 macro 133
CH_CLR3 macro 133
CH_CLR4 macro 133
CH_CLR5 macro 133
CH_CLR6 macro 133
CH_CLR7 macro 134
CH_CLR8 macro 134
CH_CLR9 macro 134
CH_DISABLED macro 102
CH_DONUT macro 103
CH_DRAW macro 102
CH_DRAW_DATA macro 102
CH_HIDE macro 104
CH_LEGEND macro 103
CH_NUMERIC macro 104
CH_PERCENT macro 104
CH_PIE macro 104
CH_VALUE macro 104
ChAddDataSeries function 109
Changing the default Font 489
Chart 99
Chart Examples 105
Chart States 101
CHART structure 130
CHARTPARAM structure 131
ChCreate function 106
ChDraw function 108
Check Box States 137
Checkbox 135
CHECKBOX structure 144
ChFreeDataSeries function 128
ChGetAxisLabelFont macro 126
ChGetColorTable macro 123
ChGetGridLabelFont macro 127
ChGetPercentMax macro 121
ChGetPercentMin macro 122
ChGetPercentRange macro 119
ChGetSampleEnd macro 118
ChGetSampleLabel macro 117
ChGetSampleRange macro 121
ChGetSampleStart macro 118
ChGetShowSeriesCount macro 112
ChGetShowSeriesStatus macro 113
ChGetTitle macro 124
ChGetTitleFont macro 125
ChGetValueLabel macro 114
ChGetValueMax macro 114
ChGetValueMin macro 115
ChGetValueRange macro 116
ChHideSeries macro 111
ChRemoveDataSeries function 110
ChSetAxisLabelFont macro 127
ChSetColorTable macro 123
ChSetGridLabelFont macro 128
ChSetPercentRange function 119
ChSetSampleLabel macro 117
ChSetSampleRange function 120
ChSetTitle macro 124
ChSetTitleFont macro 125
ChSetValueLabel macro 113
ChSetValueRange function 115

- ChShowSeries macro 111
ChTranslateMsg function 129
Circle Functions 367
Circle macro 368
ClearDevice function 388
ClearPaletteChangeError function 443
CLIP_DISABLE macro 399
CLIP_ENABLE macro 399
ClrState macro 302
Color Definition 407
Color Table 132
COLOR_DEPTH macro 55
Common Object States 300
Configuration Setting 51
Connecting RGB data bus 502
Converting Images to Use a Common Palette in GIMP 497
CopyBlock function 404
CopyPageWindow function 404
CopyWindow function 405
CYAN macro 410
- D**
- DARKGRAY macro 410
DASHED_LINE macro 363
Data Series Status Settings 105
DATASERIES structure 130
Decompress function 442
Decompressing DEFLATEd data 442
Default Style Scheme Settings 342
Demo Compatibility Matrix 32
Demo Projects 31
Demo Summary 31
Development Platform Used 58
Device Driver Options 66
Dial States 145
Digital Meter 154
Digital Meter States 155
DIGITALMETER structure 162
DISABLED macro 301
DisablePalette function 444
DISP_DATA_WIDTH macro 68
DISP_HOR_BACK_PORCH macro 69
DISP_HOR_FRONT_PORCH macro 69
DISP_HOR_PULSE_WIDTH macro 70
DISP_HOR_RESOLUTION macro 68
DISP_INV_LSHIFT macro 70
DISP_ORIENTATION macro 68
DISP_VER_BACK_PORCH macro 70
DISP_VER_FRONT_PORCH macro 69
DISP_VER_PULSE_WIDTH macro 70
DISP_VER_RESOLUTION macro 69
Display Controller Used 61
Display Device Driver Control 414
Display Device Driver Layer 36
Display Device Driver Layer API 392
Display Device Driver Layer Configuration 47
Display Device Driver Level Primitives 392
Display Panel Used 64
DisplayBrightness function 402
DM_CENTER_ALIGN macro 156
DM_DISABLED macro 155
DM_DRAW macro 155
DM_FRAME macro 156
DM_HIDE macro 156
DM_RIGHT_ALIGN macro 156
DM_UPDATE macro 156
DmCreate function 157
DmDecVal macro 160
DmDraw function 158
DmGetValue macro 159
DmlIncVal macro 160
DmSetValue function 159
DmTranslateMsg function 161
DOTTED_LINE macro 364
Double Buffering 422
DRAW macro 301
DRAW_FOCUS macro 301
DRAW_FUNC type 76
DRAW_UPDATE macro 301
DrawArc function 371
DrawPoly function 366
- E**
- EB_CARET macro 165

EB_CENTER_ALIGN macro 164
EB_DISABLED macro 164
EB_DRAW macro 164
EB_DRAW_CARET macro 165
EB_FOCUSED macro 165
EB_HIDE macro 165
EB_RIGHT_ALIGN macro 165
EbAddChar function 169
EbCreate function 166
EbDeleteChar function 169
EbDraw function 167
EbGetText macro 167
EbMsgDefault function 170
EbSetText function 168
EbTranslateMsg function 171
Edit Box 162
Edit Box States 164
EDITBOX structure 171
EnablePalette function 444
EXPLORER_16 macro 58
External Memory 385
External Memory Buffer 47
External or Internal Memory and Palettes 465
EXTERNAL_FONT_BUFFER_SIZE macro 387
ExternalMemoryCallback function 386

F

FillBevel function 373
FillCircle macro 368
Focus Support Selection 38
FOCUSED macro 300
Font Source Selection 52
Font Type Selection 44
FONT_EXTERNAL type 348
FONT_FLASH structure 347
FONT_HEADER structure 346
FONTDEFAULT variable 342
FREE_FUNC type 77

G

GB_CENTER_ALIGN macro 189
GB_DISABLED macro 190

GB_DRAW macro 190
GB_HIDE macro 190
GB_RIGHT_ALIGN macro 190
GbCreate function 190
GbDraw function 192
GbGetText macro 192
GbSetText function 193
GbTranslateMsg function 194
GetAlpha macro 378
GetClipBottom macro 398
GetClipLeft macro 398
GetClipRight macro 398
GetClipTop macro 399
GetColor macro 394
GetFontOrientation macro 348
GetImageHeight function 383
GetImageWidth function 384
GetMaxX macro 395
GetMaxY macro 396
GetObjID macro 312
GetObjNext macro 313
GetObjType macro 312
GetPageAddress macro 403
GetPaletteChangeError function 445
GetPixel function 393
GetState macro 302
GetTextHeight function 353
GetTextWidth function 354
Getting Started 23
GetTransparentColor macro 402
GetTransparentColorStatus macro 401
GetX macro 375
GetY macro 375
GFX_DISPLAY_BUFFER_LENGTH macro 463
GFX_DISPLAY_BUFFER_START_ADDRESS macro 464
GFX_EPMP_CS1_BASE_ADDRESS macro 458
GFX_EPMP_CS1_MEMORY_SIZE macro 460
GFX_EPMP_CS2_BASE_ADDRESS macro 461
GFX_EPMP_CS2_MEMORY_SIZE macro 462
GFX_EXDATA structure 391
GFX_Font_GetAntiAliasType macro 350
GFX_Font_SetAntiAliasType macro 350

GFX_free macro 55
GFX_GCLK_DIVIDER macro 457
GFX_GRADIENT_STYLE structure 359
GFX_GRADIENT_TYPE enumeration 359
GFX_IMAGE_HEADER structure 390
GFX_LCD_CSTN macro 49
GFX_LCD_MSTN macro 49
GFX_LCD_OFF macro 49
GFX_LCD_TFT macro 50
GFX_LCD_TYPE macro 48
GFX_malloc macro 56
GFX_PICTAIL_LCC macro 60
GFX_PICTAIL_V3 macro 61
GFX_PICTAIL_V3E macro 61
GFX_RESOURCE enumeration 389
GFX_TRANSITION_DIRECTION enumeration 421
GFX_TRANSITION_TYPE enumeration 422
GFX_USE_DISPLAY_CONTROLLER_DMA macro 62
GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210 macro 63
GFX_USE_DISPLAY_CONTROLLER_S1D13517 macro 63
GFX_USE_DISPLAY_CONTROLLER_SSD1926 macro 63
GFX_USE_DISPLAY_PANEL_PH480272T_005_I11Q macro 65
GFX_USE_DISPLAY_PANEL_TFT_640480_8_E macro 65
GFX_USE_DISPLAY_PANEL_TFT_800480_33_E macro 65
GFX_USE_DISPLAY_PANEL_TFT_G240320LTSW_118W_E macro 66
GFXExecutePendingTransition function 420
GFXGetPageOriginAddress function 416
GFXGetPageXYAddress function 417
GFXIsTransitionPending function 421
GFXSetupTransition function 419
GFXTransition function 419
GOL Global Variables 344
GOL Messages 322
GOL Objects 72
GOL_EMBOSS_SIZE macro 342
GOL_MSG structure 327
GOL_OBJ_TYPE enumeration 75
GOL_SCHEME structure 341
GOLAddObject function 305
GOLCanBeFocused function 319
GOLCreateScheme function 339
GOLDeleteObject function 314
GOLDeleteObjectByID function 314
GOLDraw function 308
GOLDrawCallback function 310
GOLDrawComplete macro 309
GOLFindObject function 306
GOLFontDefault variable 342
GOLFree function 311
GOLGetFocus macro 319
GOLGetFocusNext function 320
GOLGetFocusPrev function 320
GOLGetList macro 315
GOLGetScheme macro 340
GOLGetSchemeDefault macro 341
GOLInit function 318
GOLMsg function 325
GOLMsgCallback function 326
GOLNewList macro 315
GOLPanelDraw macro 320
GOLPanelDrawTsk function 321
GOLRedraw macro 307
GOLRedrawRec function 308
GOLSchemeDefault variable 343
GOLSetFocus function 317
GOLSetList macro 316
GOLSetScheme macro 339
GOLTTwoTonePanelDrawTsk function 322
Gradient 355
Gradient Bar Rendering 46
Graphic Cursor 375
Graphics Library Configuration 37
Graphics Object Layer 33
Graphics Object Layer API 72
Graphics Object Layer Configuration 37
Graphics Object Selection 39
Graphics PICtail Used 60
Graphics Primitive Layer 35
Graphics Primitive Layer API 345
Graphics Primitive Layer Configuration 43
GraphicsConfig.h Example 56
GRAY0 macro 410

- GRAY1 macro 410
GRAY2 macro 410
GRAY3 macro 411
GRAY4 macro 411
GRAY5 macro 411
GRAY6 macro 411
GREEN macro 411
Grid 172
Grid Item States 175
Grid States 173
GRID structure 187
GRID_DISABLED macro 174
GRID_DRAW_ALL macro 175
GRID_DRAW_ITEMS macro 175
GRID_FOCUSED macro 174
GRID_HIDE macro 175
GRID_OUT_OF_BOUNDS macro 181
GRID_SHOW_BORDER_ONLY macro 174
GRID_SHOW_FOCUS macro 174
GRID_SHOW_LINES macro 174
GRID_SHOW_SEPARATORS_ONLY macro 175
GRID_SUCCESS macro 181
GridClearCellState function 179
GridCreate function 177
GridDraw function 178
GridFreeItems function 181
GridGetCell function 182
GridGetFocusX macro 180
GridGetFocusY macro 180
GRIDITEM structure 188
GRIDITEM_DRAW macro 177
GRIDITEM_IS_BITMAP macro 176
GRIDITEM_IS_TEXT macro 176
GRIDITEM_SELECTED macro 176
GRIDITEM_TEXTBOTTOM macro 176
GRIDITEM_TEXTLEFT macro 177
GRIDITEM_TEXTRIGHT macro 177
GRIDITEM_TEXTTOP macro 177
GridMsgDefault function 185
GridSetCell function 183
GridSetCellState function 183
GridSetFocus function 184
GridTranslateMsg function 186
Group Box 188
Group Box States 189
GROUPBOX structure 194
- H**
- Hardware Profile 56
HardwareProfile.h Example 71
HIDE macro 301
HIDE_DATA macro 105
How to Define Colors in your Applications 501
- I**
- Image Compression Option 43
Image Decoder Configuration 472
Image Decoder Demo 471
Image Decoders 468
Image Decoders API 472
Image Source Selection 53
IMAGE_FLASH structure 391
IMAGE_NORMAL macro 385
IMAGE_RAM structure 391
IMAGE_X2 macro 385
ImageAbort macro 478
ImageDecode function 475
ImageDecoderInit function 476
ImageDecodeTask function 477
ImageFullScreenDecode macro 477
ImageLoopCallbackRegister function 476
IMG_SUPPORT_BMP macro 473
IMG_SUPPORT_GIF macro 473
IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK macro 474
IMG_SUPPORT_JPEG macro 473
IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT macro 474
IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT macro 474
InitGraph function 389
Input Device Selection 37
INPUT_DEVICE_EVENT enumeration 330
INPUT_DEVICE_TYPE enumeration 330

Introduction 1
InvalidateRectangle function 425
IsDeviceBusy function 414
IsObjUpdated macro 317
IsPaletteEnabled function 445

K

Key Command Types 278
KEYMEMBER structure 291

L

LB_CENTER_ALIGN macro 197
LB_DISABLED macro 198
LB_DRAW macro 198
LB_DRAW_FOCUS macro 198
LB_DRAW_ITEMS macro 198
LB_FOCUSED macro 198
LB_HIDE macro 199
LB_RIGHT_ALIGN macro 197
LB_SINGLE_SEL macro 197
LB_STS_REDRAW macro 199
LB_STS_SELECTED macro 199
LbAddItem function 202
LbChangeSel function 204
LbCreate function 199
LbDeleteItem function 204
LbDeleteItemsList function 210
LbDraw function 201
LbGetBitmap macro 209
LbGetCount macro 207
LbGetFocusedItem function 206
LbGetItemList macro 202
LbGetSel function 205
LbGetVisibleCount macro 208
LbMsgDefault function 210
LbSetBitmap macro 208
LbSetFocusedItem function 207
LbSetSel macro 205
LbTranslateMsg function 211
Library API 37
Library Architecture 33
LIGHTBLUE macro 412

LIGHTCYAN macro 412
LIGHTGRAY macro 412
LIGHTGREEN macro 412
LIGHTMAGENTA macro 412
LIGHTRED macro 413
Line function 360

Line Functions 360
Line Size 364
Line Types 363
LineRel macro 361
LineTo macro 361
List Box 195
List Box States 197
List Item Status 199
LISTBOX structure 212
LISTITEM structure 213

M

MAGENTA macro 413
MEB_BOARD macro 59
Memory Type 388
Meter 213
Meter States 215
METER structure 225
METER_TYPE macro 223
Microchip Application Library Abbreviations 32
Microchip Graphics Controller 427
Miscellaneous 54
Miscellaneous Topics 483
MoveRel macro 376
MoveTo macro 376
MSG_DEFAULT_FUNC type 77
MSG_FUNC type 77
MTR_ACCURACY macro 223
MTR_DISABLED macro 215
MTR_DRAW macro 216
MTR_DRAW_UPDATE macro 216
MTR_HIDE macro 216
MTR_RING macro 216
MtrCreate function 216
MtrDecVal macro 220
MtrDraw function 218

MtrGetVal macro 219
MtrIncVal macro 220
MtrMsgDefault function 224
MtrSetScaleColors macro 221
MtrSetTitleFont macro 222
MtrSetVal function 219
MtrSetValueFont macro 222
MtrTranslateMsg function 224
PICT_DISABLED macro 228
PICT_DRAW macro 228
PICT_FRAME macro 228
PICT_HIDE macro 228
PictCreate function 228
PictDraw function 229
PictGetBitmap macro 230
PictGetScale macro 231
PictSetBitmap macro 230
PictSetScale macro 232
PictTranslateMsg function 232
Picture Control 226
Picture States 227
PICTURE structure 233
PMP Interface 57
Progress Bar 233
Progress Bar States 235
PROGRESSBAR structure 241
PutImage macro 381
PutImagePartial function 381
PutPixel function 394

N

NORMAL_LINE macro 364

O

OBJ_HEADER structure 76
Object Management 304
Object Rendering 34
Object States 299
OutChar function 351
OutText function 351
OutTextXY function 352
PICTURE structure 233
PMP Interface 57
Progress Bar 233
Progress Bar States 235
PROGRESSBAR structure 241
PutImage macro 381
PutImagePartial function 381
PutPixel function 394

P

Palette Mode 443

Palette.h 450

PALETTE_EXTERNAL type 449

PALETTE_FLASH structure 448

PALETTE_HEADER structure 448

PaletteInit function 445

PB_DISABLED macro 235

PB_DRAW macro 235

PB_DRAW_BAR macro 235

PB_HIDE macro 235

PB_VERTICAL macro 236

PbCreate function 236

PbDraw function 237

PbGetPos macro 240

PbGetRange macro 238

PbSetPos function 239

PbSetRange function 238

PbTranslateMsg function 240

PIC_SK macro 59

PIC24FJ256DA210_DEV_BOARD macro 59

R

RadioButton 242

RadioButton States 243

RADIOBUTTON structure 252

RB_CHECKED macro 243

RB_DISABLED macro 243

RB_DRAW macro 244

RB_DRAW_CHECK macro 244

RB_DRAW_FOCUS macro 244

RB_FOCUSED macro 244

RB_GROUP macro 244

RB_HIDE macro 245

RbCreate function 245

RbDraw function 246

RbGetCheck function 247

RbGetText macro 249

RbMsgDefault function 250

RbSetCheck function 248

RbSetText function 249

RbTranslateMsg function 251

RCC_COPY macro 441
RCC_DEST_ADDR_CONTINUOUS macro 440
RCC_DEST_ADDR_DISCONTINUOUS macro 440
RCC_ROP_0 macro 441
RCC_ROP_1 macro 441
RCC_ROP_2 macro 441
RCC_ROP_3 macro 441
RCC_ROP_4 macro 441
RCC_ROP_5 macro 441
RCC_ROP_6 macro 441
RCC_ROP_7 macro 441
RCC_ROP_8 macro 441
RCC_ROP_9 macro 441
RCC_ROP_A macro 441
RCC_ROP_B macro 441
RCC_ROP_C macro 441
RCC_ROP_D macro 441
RCC_ROP_E macro 441
RCC_ROP_F macro 441
RCC_SOLID_FILL macro 441
RCC_SRC_ADDR_CONTINUOUS macro 440
RCC_SRC_ADDR_DISCONTINUOUS macro 440
RCC_TRANSPARENT_COPY macro 441
RDIA_DISABLED macro 146
RDIA_DRAW macro 146
RDIA_HIDE macro 146
RDIA_ROT_CCW macro 146
RDIA_ROT_CW macro 146
RdiaCreate function 147
RdiaDecVal macro 149
RdiaDraw function 148
RdiaGetVal macro 150
RdialncVal macro 148
RdiaMsgDefault function 151
RdiaSetVal macro 150
RdiaTranslateMsg function 152
Rectangle Copy Operations 428
Rectangle Functions 365
Rectangle macro 366
RED macro 413
References 505
Release Notes 2
RequestDisplayUpdate function 426
RequestEntirePaletteChange macro 449
RequestPaletteChange function 446
ResetDevice function 414
RGBConvert macro 343
ROPBlock function 438
Round Dial 144
ROUNDDIAL structure 153

S

Scan Key Codes 331
SCAN_BS_PRESSED macro 332
SCAN_BS_RELEASED macro 332
SCAN_CR_PRESSED macro 332
SCAN_CR_RELEASED macro 332
SCAN_DEL_PRESSED macro 332
SCAN_DEL_RELEASED macro 333
SCAN_DOWN_PRESSED macro 333
SCAN_DOWN_RELEASED macro 333
SCAN_END_PRESSED macro 333
SCAN_END_RELEASED macro 333
SCAN_HOME_PRESSED macro 334
SCAN_HOME_RELEASED macro 334
SCAN_LEFT_PRESSED macro 334
SCAN_LEFT_RELEASED macro 334
SCAN_PGDOWN_PRESSED macro 334
SCAN_PGDOWN_RELEASED macro 335
SCAN_PGUP_PRESSED macro 335
SCAN_PGUP_RELEASED macro 335
SCAN_RIGHT_PRESSED macro 335
SCAN_RIGHT_RELEASED macro 335
SCAN_SPACE_PRESSED macro 336
SCAN_SPACE_RELEASED macro 336
SCAN_TAB_PRESSED macro 336
SCAN_TAB_RELEASED macro 336
SCAN_UP_PRESSED macro 336
SCAN_UP_RELEASED macro 337
Scroll function 439
Set Up Display Interface 455
Set Up Functions 388
SetActivePage function 406
SetAlpha macro 378

SetBevelDrawType macro 374
SetClip function 396
SetClipRgn function 397
SetColor macro 395
SetEntirePalette macro 449
SetFont function 348
SetFontOrientation macro 349
SetLineThickness macro 362
SetLineType macro 362
SetPalette function 446
SetPaletteBpp function 447
SetPaletteFlash function 447
SetState macro 303
SetVisualPage function 407
SHOW_DATA macro 105
SLD_DISABLED macro 254
SLD_DRAW macro 255
SLD_DRAW_FOCUS macro 255
SLD_DRAW_THUMB macro 255
SLD_FOCUSED macro 255
SLD_HIDE macro 255
SLD_SCROLLBAR macro 256
SLD_VERTICAL macro 256
SldCreate function 256
SldDecPos macro 264
SldDraw function 258
SldGetPage macro 259
SldGetPos macro 261
SldGetRange macro 262
SldIncPos macro 263
SldMsgDefault function 265
SldSetPage function 259
SldSetPos function 260
SldSetRange function 262
SldTranslateMsg function 266
Slider States 254
SLIDER structure 267
Slider/Scroll Bar 252
SOLID_LINE macro 363
ST_CENTER_ALIGN macro 269
ST_DISABLED macro 269
ST_DRAW macro 269
ST_FRAME macro 270
ST_HIDE macro 270
ST_RIGHT_ALIGN macro 270
ST_UPDATE macro 270
Starting a New Project 483
Static Text 267
Static Text States 269
STATICTEXT structure 274
StCreate function 270
StDraw function 271
StGetText macro 272
STN_DISPLAY_WIDTH macro 50
STN_DISPLAY_WIDTH_16 macro 50
STN_DISPLAY_WIDTH_4 macro 50
STN_DISPLAY_WIDTH_8 macro 51
StSetText function 273
StTranslateMsg function 273
Style Scheme 337
SwitchOffDoubleBuffering function 424
SwitchOnDoubleBuffering function 425

T

TE_DELETE_COM macro 279
TE_DISABLED macro 277
TE_DRAW macro 278
TE_ECHO_HIDE macro 277
TE_ENTER_COM macro 279
TE_HIDE macro 278
TE_KEY_PRESSED macro 277
TE_SPACE_COM macro 279
TE_UPDATE_KEY macro 278
TE_UPDATE_TEXT macro 278
TeAddChar function 285
TeClearBuffer function 282
TeCreate function 279
TeCreateKeyMembers function 284
TeDelKeyMembers function 286
TeDraw function 280
TeGetBuffer macro 281
TeGetKeyCommand function 283
TeIsKeyPressed function 285
TeMsgDefault function 288

TeSetBuffer function 281
 TeSetKeyCommand function 283
 TeSetKeyText function 287
 TeSpaceChar function 286
 TeTranslateMsg function 289
 Text Entry 275
 Text Functions 345
 TextEntry States 277
 TEXTENTRY structure 290
 THICK_LINE macro 364
 TRANS_MSG enumeration 328
 Transitions 418
 Transparent Color Feature in PutImage() 46
 TRANSPARENT_COLOR_DISABLE macro 402
 TRANSPARENT_COLOR_ENABLE macro 402
 TransparentColorDisable macro 401
 TransparentColorEnable function 400

U

UPDATE_HOUR macro 79
 UPDATE_MINUTE macro 80
 UPDATE_SECOND macro 80
 UpdateDisplayNow function 426
 USE_16BIT_PMP macro 57
 USE_8BIT_PMP macro 57
 USE_ALPHABLEND macro 48
 USE_ALPHABLEND_LITE macro 47
 USE_ANALOGCLOCK macro 39
 USE_ANTIALIASED_FONTS macro 46
 USE_BITMAP_EXTERNAL macro 54
 USE_BITMAP_FLASH macro 53
 USE_BITMAP_NO_PADDING_LINE macro 55
 USE_BUTTON macro 40
 USE_BUTTON_MULTI_LINE macro 40
 USE_CHECKBOX macro 40
 USE_COMP_IPU macro 44
 USE_COMP_RLE macro 44
 USE_CUSTOM macro 43
 USE_DIGITALMETER macro 40
 USE_DOUBLE_BUFFERING macro 48
 USE_EDITBOX macro 40
 USE_FOCUS macro 38
 USE_FONT_EXTERNAL macro 52
 USE_FONT_FLASH macro 52
 USE_GFX_FONT_IN_PROGRAM_SECTION macro 53
 USE_GOL macro 43
 USE_GRADIENT macro 46
 USE_GROUPBOX macro 41
 USE_KEYBOARD macro 38
 USE_LISTBOX macro 41
 USE_METER macro 41
 USE_MULTIBYTECHAR macro 45
 USE_NONBLOCKING_CONFIG macro 51
 USE_PALETTE macro 55
 USE_PALETTE_EXTERNAL macro 55
 USE_PICTURE macro 41
 USE_PROGRESSBAR macro 41
 USE_RADIOBUTTON macro 42
 USE_ROUNDDIAL macro 42
 USE_SLIDER macro 42
 USE_STATICTEXT macro 42
 USE_TEXTENTRY macro 43
 USE_TOUCHSCREEN macro 38
 USE_TRANSPARENT_COLOR macro 46
 USE_UNSIGNED_XCHAR macro 45
 USE_WINDOW macro 42
 Using Microchip Graphics Module Color Look Up Table in Applications 492
 Using Primitive Rendering Functions in Blocking and Non-Blocking Modes 491

W

WHITE macro 413
 Window 292
 Window States 293
 WINDOW structure 299
 WND_DISABLED macro 293
 WND_DRAW macro 294
 WND_DRAW_CLIENT macro 294
 WND_DRAW_TITLE macro 294
 WND_FOCUSED macro 294
 WND_HIDE macro 294
 WND_TITLECENTER macro 295
 WndCreate function 295
 WndDraw function 296

WndGetText macro 297

WndSetText function 297

WndTranslateMsg function 298

X

XCHAR macro 354

Y

YELLOW macro 413