made with doc-o-matic

Microchip Graphics Library

Manual for Microchip Graphics Library

Copyright 2011 Microchip Technology, Inc.; All rights reserved

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

Table of Contents

Introduction 1

Microchip Software Bundle License.txt 2

Release Notes 5

Getting Started 21

Demo Projects 28

Demo Summary 28

Microchip Application Library Abbreviations 28

Demo Compatibility Matrix 29

Graphics Library Configuration 30

Graphics Object Layer Configuration 30

Input Device Selection 30

USE_KEYBOARD Macro 30

USE_TOUCHSCREEN Macro 30

Focus Support Selection 31

USE_FOCUS Macro 31

Graphics Object Selection 31

USE_BUTTON Macro 32

USE_BUTTON_MULTI_LINE Macro 32

USE_CHECKBOX Macro 32

USE_DIGITALMETER Macro 33

USE_EDITBOX Macro 33

USE_GROUPBOX Macro 33

USE_LISTBOX Macro 33

USE_METER Macro 33

USE_PICTURE Macro 34

USE_PROGRESSBAR Macro 34

USE_RADIOBUTTON Macro 34

USE_ROUNDDIAL Macro 34

USE_SLIDER Macro 35

USE_STATICTEXT Macro 35

USE_WINDOW Macro 35

USE_CUSTOM Macro 35
USE_GOL Macro 35
USE_ANALOGCLOCK Macro 36
USE_TEXTENTRY Macro 36

Graphics Primitive Layer Configuration 36
Font Type Selection 36
USE_MULTIBYTECHAR Macro 37

Display Device Driver Layer Configuration 37
GFX_LCD_TYPE Macro 38
GFX_LCD_CSTN Macro 39
GFX_LCD_MSTN Macro 39
GFX_LCD_OFF Macro 39
GFX_LCD_TFT Macro 39
STN_DISPLAY_WIDTH Macro 40
STN_DISPLAY_WIDTH_16 Macro 40
STN_DISPLAY_WIDTH_4 Macro 40
STN_DISPLAY_WIDTH_8 Macro 41
COLOR_DEPTH Macro 41
Graphics Hardware Platform 41

Application Configuration 41
Configuration Setting 41
USE_NONBLOCKING_CONFIG Macro 42
Font Source Selection 42
USE_FONT_FLASH Macro 42
Image Source Selection 42
USE_BITMAP_FLASH Macro 43
Miscellaneous 43
GFX_free Macro 43
GFX_malloc Macro 43
GraphicsConfig.h Example 44

Hardware Profile 44
Device Driver Options 44
HardwareProfile.h Example 45

Library Structure 47

Graphics Object Layer 48

GOL Objects 48
GOL_OBJ_TYPE Enumeration 51
OBJ_HEADER Structure 52

DRAW_FUNC Type 52
FREE_FUNC Type 53
MSG_DEFAULT_FUNC Type 53
MSG_FUNC Type 53
Analog Clock 53
Analog Clock States 54
AC_DRAW Macro 55
AC_DISABLED Macro 55
AC_HIDE Macro 55
AC_PRESSED Macro 55
AC_TICK Macro 56
UPDATE_HOUR Macro 56
UPDATE_MINUTE Macro 56
UPDATE_SECOND Macro 56
AcCreate Function 56
AcDraw Function 58
AcHandsDraw Function 59
AcSetHour Function 60
AcSetMinute Function 60
AcSetSecond Function 61
AcMsgDefault Function 61
AcTranslateMsg Function 62
ANALOGCLOCK Structure 64
Button 65
Button States 66
BTN_DISABLED Macro 66
BTN_DRAW Macro 67
BTN_DRAW_FOCUS Macro 67
BTN_FOCUSED Macro 67
BTN_HIDE Macro 67
BTN_PRESSED Macro 67
BTN_TEXTBOTTOM Macro 68
BTN_TEXTLEFT Macro 68
BTN_TEXTRIGHT Macro 68
BTN_TEXTTOP Macro 68
BTN_TOGGLE Macro 68
BTN_TWOTONE Macro 69
BTN_NOPANEL Macro 69
BtnCreate Function 69
BtnDraw Function 70
BtnGetText Macro 72
BtnSetText Function 72

BtnGetBitmap Macro 73
BtnSetBitmap Macro 74
BtnMsgDefault Function 74
BtnTranslateMsg Function 75
BUTTON Structure 77
Chart 77
Chart States 80
CH_DISABLED Macro 81
CH_DRAW Macro 81
CH_DRAW_DATA Macro 81
CH_3D_ENABLE Macro 81
CH_BAR Macro 82
CH_BAR_HOR Macro 82
CH_DONUT Macro 82
CH_LEGEND Macro 82
CH_NUMERIC Macro 82
CH_PERCENT Macro 83
CH_PIE Macro 83
CH_VALUE Macro 83
CH_HIDE Macro 83
Data Series Status Settings 84
HIDE_DATA Macro 84
SHOW_DATA Macro 84
Chart Examples 84
ChCreate Function 86
ChDraw Function 87
ChAddDataSeries Function 88
ChRemoveDataSeries Function 89
ChShowSeries Macro 90
ChHideSeries Macro 91
ChGetShowSeriesCount Macro 91
ChGetShowSeriesStatus Macro 92
ChSetValueLabel Macro 92
ChGetValueLabel Macro 93
ChGetValueMax Macro 94
ChGetValueMin Macro 94
ChSetValueRange Function 95
ChGetValueRange Macro 95
ChSetSampleLabel Macro 96
ChGetSampleLabel Macro 97
ChGetSampleStart Macro 97
ChGetSampleEnd Macro 98

ChSetPercentRange Function 98
ChGetPercentRange Macro 99
ChSetSampleRange Function 99
ChGetSampleRange Macro 100
ChGetPercentMax Macro 101
ChGetPercentMin Macro 102
ChSetColorTable Macro 102
ChGetColorTable Macro 103
ChSetTitle Macro 103
ChGetTitle Macro 104
ChSetTitleFont Macro 104
ChGetTitleFont Macro 105
ChGetAxisLabelFont Macro 106
ChSetAxisLabelFont Macro 106
ChGetGridLabelFont Macro 107
ChSetGridLabelFont Macro 107
ChFreeDataSeries Function 108
ChTranslateMsg Function 109
CHART Structure 110
DATASERIES Structure 110
CHARTPARAM Structure 111
Color Table 111
CH_CLR0 Macro 112
CH_CLR1 Macro 112
CH_CLR2 Macro 112
CH_CLR3 Macro 113
CH_CLR4 Macro 113
CH_CLR5 Macro 113
CH_CLR6 Macro 113
CH_CLR7 Macro 113
CH_CLR8 Macro 114
CH_CLR9 Macro 114
CH_CLR10 Macro 114
CH_CLR11 Macro 114
CH_CLR12 Macro 114
CH_CLR13 Macro 115
CH_CLR14 Macro 115
CH_CLR15 Macro 115
Checkbox 115
Check Box States 117
CB_CHECKED Macro 117
CB_DISABLED Macro 118

CB_DRAW Macro 118
CB_DRAW_CHECK Macro 118
CB_DRAW_FOCUS Macro 118
CB_FOCUSED Macro 118
CB_HIDE Macro 119
CbCreate Function 119
CbDraw Function 120
CbGetText Macro 121
CbSetText Function 121
CbMsgDefault Function 122
CbTranslateMsg Function 123
CHECKBOX Structure 124
Dial 124
Dial States 126
RDIA_DISABLED Macro 126
RDIA_DRAW Macro 126
RDIA_HIDE Macro 126
RDIA_ROT_CCW Macro 127
RDIA_ROT_CW Macro 127
RdiaCreate Function 127
RdiaDraw Function 128
RdiaIncVal Macro 129
RdiaDecVal Macro 130
RdiaGetVal Macro 130
RdiaSetVal Macro 131
RdiaMsgDefault Function 132
RdiaTranslateMsg Function 132
ROUNDDIAL Structure 134
Digital Meter 135
Digital Meter States 136
DM_DISABLED Macro 136
DM_DRAW Macro 136
DM_HIDE Macro 137
DM_CENTER_ALIGN Macro 137
DM_RIGHT_ALIGN Macro 137
DM_FRAME Macro 137
DM_UPDATE Macro 137
DmCreate Function 138
DmDraw Function 139
DmGetValue Macro 140
DmSetValue Function 140
DmDecVal Macro 141

DmIncVal Macro 141
DmTranslateMsg Function 142
DIGITALMETER Structure 143
Edit Box 143
Edit Box States 145
EB_CENTER_ALIGN Macro 145
EB_DISABLED Macro 145
EB_DRAW Macro 145
EB_HIDE Macro 146
EB_FOCUSED Macro 146
EB_RIGHT_ALIGN Macro 146
EB_DRAW_CARET Macro 146
EB_CARET Macro 146
EbCreate Function 147
EbDraw Function 148
EbGetText Macro 149
EbSetText Function 149
EbAddChar Function 150
EbDeleteChar Function 150
EbMsgDefault Function 151
EbTranslateMsg Function 152
EDITBOX Structure 153
Grid 153
Grid States 154
GRID_FOCUSED Macro 155
GRID_DISABLED Macro 155
GRID_SHOW_LINES Macro 155
GRID_SHOW_FOCUS Macro 155
GRID_SHOW_BORDER_ONLY Macro 156
GRID_SHOW_SEPARATORS_ONLY Macro 156
GRID_DRAW_ITEMS Macro 156
GRID_DRAW_ALL Macro 156
GRID_HIDE Macro 156
Grid Item States 157
GRIDITEM_SELECTED Macro 157
GRIDITEM_IS_TEXT Macro 157
GRIDITEM_IS_BITMAP Macro 158
GRIDITEM_TEXTBOTTOM Macro 158
GRIDITEM_TEXTLEFT Macro 158
GRIDITEM_TEXTRIGHT Macro 158
GRIDITEM_TEXTTOP Macro 158
GRIDITEM_DRAW Macro 159

GridCreate Function 159
GridDraw Function 160
GridClearCellState Function 161
GridGetFocusX Macro 161
GridGetFocusY Macro 162
GRID_OUT_OF_BOUNDS Macro 162
GRID_SUCCESS Macro 163
GridFreeItems Function 163
GridGetCell Function 163
GridSetCell Function 164
GridSetCellState Function 165
GridSetFocus Function 166
GridMsgDefault Function 166
GridTranslateMsg Function 167
GRID Structure 169
GRIDITEM Structure 169
Group Box 170
Group Box States 171
GB_CENTER_ALIGN Macro 171
GB_DISABLED Macro 172
GB_DRAW Macro 172
GB_HIDE Macro 172
GB_RIGHT_ALIGN Macro 172
GbCreate Function 172
GbDraw Function 174
GbGetText Macro 174
GbSetText Function 175
GbTranslateMsg Function 176
GROUPBOX Structure 176
List Box 177
List Box States 179
LB_RIGHT_ALIGN Macro 179
LB_SINGLE_SEL Macro 179
LB_CENTER_ALIGN Macro 179
LB_DISABLED Macro 180
LB_DRAW Macro 180
LB_DRAW_FOCUS Macro 180
LB_DRAW_ITEMS Macro 180
LB_FOCUSED Macro 180
LB_HIDE Macro 181
List Item Status 181
LB_STS_SELECTED Macro 181

LB_STS_REDRAW Macro 181
LbCreate Function 181
LbDraw Function 183
LbGetItemList Macro 184
LbAddItem Function 184
LbDelItem Function 186
LbChangeSel Function 186
LbSetSel Macro 187
LbGetSel Function 188
LbGetFocusedItem Function 188
LbSetFocusedItem Function 189
LbGetCount Macro 189
LbGetVisibleCount Macro 190
LbClrtSel Macro 191
LbSetBitmap Macro 191
LbGetBitmap Macro 192
LbDelItemsList Function 193
LbMsgDefault Function 193
LbTranslateMsg Function 194
LISTBOX Structure 195
LISTITEM Structure 196
Meter 196
Meter States 198
MTR_DISABLED Macro 198
MTR_DRAW Macro 199
MTR_HIDE Macro 199
MTR_RING Macro 199
MTR_DRAW_UPDATE Macro 199
MtrCreate Function 200
MtrDraw Function 201
MtrSetVal Function 202
MtrGetVal Macro 203
MtrDecVal Macro 203
MtrIncVal Macro 204
MtrSetScaleColors Macro 204
MtrSetTitleFont Macro 205
MtrSetValueFont Macro 206
METER_TYPE Macro 206
METER_DISPLAY_VALUES_ENABLE Macro 207
MTR_ACCURACY Macro 207
RESOLUTION Macro 207
MtrMsgDefault Function 208

MtrTranslateMsg Function 208
METER Structure 209
Picture Control 210
Picture States 211
PICT_DISABLED Macro 211
PICT_DRAW Macro 212
PICT_FRAME Macro 212
PICT_HIDE Macro 212
PictCreate Function 212
PictDraw Function 213
PictSetBitmap Macro 214
PictGetBitmap Macro 215
PictGetScale Macro 215
PictSetScale Macro 216
PictTranslateMsg Function 216
PICTURE Structure 217
Progress Bar 218
Progress Bar States 219
PB_DISABLED Macro 219
PB_DRAW Macro 219
PB_DRAW_BAR Macro 220
PB_HIDE Macro 220
PB_VERTICAL Macro 220
PbCreate Function 220
PbDraw Function 221
PbSetRange Function 222
PbGetRange Macro 223
PbSetPos Function 223
PbGetPos Macro 224
PbTranslateMsg Function 225
PROGRESSBAR Structure 226
Radio Button 226
Radio Button States 228
RB_CHECKED Macro 228
RB_DISABLED Macro 228
RB_DRAW Macro 228
RB_DRAW_CHECK Macro 229
RB_DRAW_FOCUS Macro 229
RB_FOCUSED Macro 229
RB_GROUP Macro 229
RB_HIDE Macro 229
RbCreate Function 230

RbDraw Function 231
RbGetCheck Function 232
RbSetCheck Function 233
RbGetText Macro 233
RbSetText Function 234
RbMsgDefault Function 235
RbTranslateMsg Function 236
RADIOBUTTON Structure 237
Slider/Scroll Bar 237
Slider States 239
SLD_DISABLED Macro 239
SLD_DRAW Macro 240
SLD_DRAW_FOCUS Macro 240
SLD_DRAW_THUMB Macro 240
SLD_FOCUSED Macro 240
SLD_HIDE Macro 240
SLD_SCROLLBAR Macro 241
SLD_VERTICAL Macro 241
SldCreate Function 241
SldDraw Function 243
SldSetPage Function 244
SldGetPage Macro 244
SldSetPos Function 245
SldGetPos Macro 246
SldSetRange Function 247
SldGetRange Macro 247
SldIncPos Macro 248
SldDecPos Macro 249
SldMsgDefault Function 250
SldTranslateMsg Function 251
SLIDER Structure 252
Static Text 252
Static Text States 254
ST_CENTER_ALIGN Macro 254
ST_DISABLED Macro 254
ST_DRAW Macro 254
ST_FRAME Macro 255
ST_HIDE Macro 255
ST_RIGHT_ALIGN Macro 255
ST_UPDATE Macro 255
StCreate Function 255
StDraw Function 256

StGetText Macro 257
StSetText Function 258
StTranslateMsg Function 258
STATICTEXT Structure 259
Text Entry 260
TextEntry States 262
TE_KEY_PRESSED Macro 262
TE_DISABLED Macro 263
TE_ECHO_HIDE Macro 263
TE_DRAW Macro 263
TE_HIDE Macro 263
TE_UPDATE_KEY Macro 263
TE_UPDATE_TEXT Macro 264
Key Command Types 264
TE_DELETE_COM Macro 264
TE_ENTER_COM Macro 264
TE_SPACE_COM Macro 265
TeCreate Function 265
TeDraw Function 266
TeGetBuffer Macro 267
TeSetBuffer Function 267
TeClearBuffer Function 268
TeGetKeyCommand Function 268
TeSetKeyCommand Function 269
TeCreateKeyMembers Function 270
TeAddChar Function 270
TeIsKeyPressed Function 271
TeSpaceChar Function 272
TeDelKeyMembers Function 272
TeSetKeyText Function 273
TeMsgDefault Function 273
TeTranslateMsg Function 275
TEXTENTRY Structure 276
KEYMEMBER Structure 277
Window 278
Window States 279
WND_DISABLED Macro 279
WND_DRAW Macro 280
WND_DRAW_CLIENT Macro 280
WND_DRAW_TITLE Macro 280
WND_FOCUSED Macro 280
WND_HIDE Macro 280

WND_TITLECENTER Macro 281
WndCreate Function 281
WndDraw Function 282
WndGetText Macro 283
WndSetText Function 283
WndTranslateMsg Function 284
WINDOW Structure 285

Object States 285
Common Object States 286
FOCUSED Macro 287
DISABLED Macro 287
HIDE Macro 287
DRAW Macro 287
DRAW_FOCUS Macro 288
DRAW_UPDATE Macro 288
GetState Macro 288
ClrState Macro 289
SetState Macro 289

Object Management 290
GOLAddObject Function 292
GOLFindObject Function 293
GOLRedraw Macro 294
GOLRedrawRec Function 295
GOLDraw Function 296
GOLDrawComplete Macro 296
GOLDrawCallback Function 297
GOLFree Function 298
GetObjType Macro 299
GetObjID Macro 299
GetObjNext Macro 300
GOLDeleteObject Function 301
GOLDeleteObjectByID Function 301
GOLNewList Macro 302
GOLGetList Macro 303
GOLSetList Macro 303
GOLSetFocus Function 304
IsObjUpdated Macro 305
GOLInit Function 306
GOLGetFocus Macro 306
GOLCanBeFocused Function 307
GOLGetFocusNext Function 307

GOLGetFocusPrev Function 308
GOLPanelDraw Macro 308
GOLPanelDrawTsk Function 310
GOLTwoTonePanelDrawTsk Function 310

GOL Messages 311
GOLMsg Function 314
GOLMsgCallback Function 314
GOL_MSG Structure 316
TRANS_MSG Enumeration 317
INPUT_DEVICE_EVENT Enumeration 319
INPUT_DEVICE_TYPE Enumeration 319
Scan Key Codes 320
SCAN_BS_PRESSED Macro 321
SCAN_BS_RELEASED Macro 321
SCAN_CR_PRESSED Macro 321
SCAN_CR_RELEASED Macro 321
SCAN_DEL_PRESSED Macro 322
SCAN_DEL_RELEASED Macro 322
SCAN_DOWN_PRESSED Macro 322
SCAN_DOWN_RELEASED Macro 322
SCAN_END_PRESSED Macro 323
SCAN_END_RELEASED Macro 323
SCAN_HOME_PRESSED Macro 323
SCAN_HOME_RELEASED Macro 323
SCAN_LEFT_PRESSED Macro 323
SCAN_LEFT_RELEASED Macro 324
SCAN_PGDOWN_PRESSED Macro 324
SCAN_PGDOWN_RELEASED Macro 324
SCAN_PGUP_PRESSED Macro 324
SCAN_PGUP_RELEASED Macro 325
SCAN_RIGHT_PRESSED Macro 325
SCAN_RIGHT_RELEASED Macro 325
SCAN_SPACE_PRESSED Macro 325
SCAN_SPACE_RELEASED Macro 325
SCAN_TAB_PRESSED Macro 326
SCAN_TAB_RELEASED Macro 326
SCAN_UP_PRESSED Macro 326
SCAN_UP_RELEASED Macro 326

Style Scheme 327
GOLCreateScheme Function 328
GOLSetScheme Macro 329

GOLGetScheme Macro 330
GOLGetSchemeDefault Macro 330
GOL_SCHEME Structure 331
Default Style Scheme Settings 331
FONTDEFAULT Variable 332
GOLFontDefault Variable 332
GOL_EMBOSS_SIZE Macro 332
GOLSchemeDefault Variable 332
RGBConvert Macro 333

GOL Global Variables 333
_pDefaultGolScheme Variable 334
_pGolObjects Variable 334
_pObjectFocused Variable 335

Object Rendering 335

Types 337
GFX_COLOR Type 337

Graphics Primitive Layer 338

Text Functions 338
FONT_HEADER Structure 339
FONT_FLASH Structure 340
FONT_EXTERNAL Type 340
SetFont Function 340
GetFontOrientation Macro 341
SetFontOrientation Macro 341
OutChar Function 342
OutText Function 343
OutTextXY Function 343
GetTextHeight Function 344
GetTextWidth Function 345
XCHAR Macro 345

Gradients 346

Line Functions 347
Line Function 347
LineRel Macro 348
LineTo Macro 349
SetLineThickness Macro 349
SetLineType Macro 350
Line Types 350
SOLID_LINE Macro 351

DASHED_LINE Macro 351
DOTTED_LINE Macro 351
Line Size 351
NORMAL_LINE Macro 351
THICK_LINE Macro 352

Rectangle Functions 352
Bar Function 352
Rectangle Macro 353
DrawPoly Function 354

Circle Functions 354
Circle Macro 355
FillCircle Macro 355
Arc Function 356
Bevel Function 357
FillBevel Function 358
SetBevelDrawType Macro 359

Graphic Cursor 359
GetX Macro 360
GetY Macro 360
MoveRel Macro 361
MoveTo Macro 361

Bitmap Functions 362
PutImage Function 362
GetImageHeight Function 363
GetImageWidth Function 363
BITMAP_HEADER Structure 364
Bitmap Settings 364
IMAGE_NORMAL Macro 365
IMAGE_X2 Macro 365
Bitmap Source 365

External Memory 365
EXTERNAL_FONT_BUFFER_SIZE Macro 366
Memory Type 366

Set Up Functions 366
ClearDevice Function 367
InitGraph Function 367

GFX_RESOURCE Enumeration 368

GFX_IMAGE_HEADER Structure 368

IMAGE_FLASH Structure 369

IMAGE_RAM Structure **370**

GFX_EXTDATA Structure **370**

GFX_GRADIENT_STYLE Structure **371**

GFX_GRADIENT_TYPE Enumeration **371**

Types **372**

IMAGE_EXTERNAL Type 372

Display Device Driver Layer 373

Display Device Driver Level Primitives **373**

GetPixel Function 374

PutPixel Function 375

GetColor Macro 376

SetColor Macro 376

GetMaxX Macro 377

GetMaxY Macro 377

SetClip Function 378

SetClipRgn Function 378

GetClipBottom Macro 379

GetClipLeft Macro 379

GetClipRight Macro 380

GetClipTop Macro 380

CLIP_DISABLE Macro 381

CLIP_ENABLE Macro 381

TransparentColorEnable Function 381

TransparentColorDisable Macro 382

GetTransparentColor Macro 382

TRANSPARENT_COLOR_DISABLE Macro 383

TRANSPARENT_COLOR_ENABLE Macro 383

DisplayBrightness Function 383

GetPageAddress Macro 384

CopyBlock Function 384

CopyPageWindow Function 385

CopyWindow Function 386

SetActivePage Function 387

SetVisualPage Function 388

Color Definition 388

BLACK Macro 389

BLUE Macro 389

BRIGHTBLUE Macro 389

BRIGHTCYAN Macro 390

BRIGHTGREEN Macro 390
BRIGHTMAGENTA Macro 390
BRIGHTRED Macro 390
BRIGHTYELLOW Macro 390
BROWN Macro 391
CYAN Macro 391
DARKGRAY Macro 391
GRAY0 Macro 391
GRAY1 Macro 392
GRAY2 Macro 392
GRAY4 Macro 392
GRAY6 Macro 392
GREEN Macro 392
LIGHTBLUE Macro 393
LIGHTCYAN Macro 393
LIGHTGRAY Macro 393
LIGHTGREEN Macro 393
LIGHTMAGENTA Macro 394
LIGHTRED Macro 394
MAGENTA Macro 394
RED Macro 394
WHITE Macro 394
YELLOW Macro 395

Display Device Driver Control 395
IsDeviceBusy Function 395
SetPalette Macro 396
ResetDevice Function 396

Adding New Device Driver 397

Advanced Display Driver Features 399

Alpha Blending 399
AlphaBlendWindow Function 400
GetPositionAddress Function 401
GFXGetPageOriginAddress Function 402
GFXGetPageXYAddress Function 402
GFXGetWindowOriginAddress Function 403
GFXGetWindowXYAddress Function 403

Transitions 403
GFXExecutePendingTransition Function 405
GFXIsTransitionPending Function 405

GFXSetupTransition Function 405
GFXTransition Function 406
GFXTranslateDirection Function 406

Microchip Graphics Controller 407
Rectangle Copy Operations 407
ROPBlock Function 417
CopyBlock Macro 419
Scroll Function 419
RCC_SRC_ADDR_CONTINUOUS Macro 420
RCC_ROP_0 Macro 421
RCC_COPY Macro 421
Double Buffering 422
SwitchOffDoubleBuffering Function 424
SwitchOnDoubleBuffering Function 424
InvalidateRectangle Function 425
RequestDisplayUpdate Function 425
UpdateDisplayNow Function 426
Decompressing DEFLATEd data 426
Decompress Function 426
Palette Mode 427
ClearPaletteChangeError Function 428
DisablePalette Function 428
EnablePalette Function 429
GetPaletteChangeError Function 429
IsPaletteEnabled Function 430
PaletteInit Function 430
RequestPaletteChange Function 430
SetPalette Function 431
SetPaletteBpp Function 432
SetPaletteFlash Function 432
PALETTE_ENTRY Union 433
PALETTE_FLASH Structure 433
PALETTE_HEADER Structure 434
PALETTE_EXTERNAL Type 434
RequestEntirePaletteChange Macro 434
SetEntirePalette Macro 435
Palette.h 435

Functions 441
CopyPageWindow Function 441

Structs, Records, Enums 442
GFX_PAGE Enumeration 442

GFX_PAGE_INFO Structure 443
GFX_WINDOW_INFO Structure 443

Variables 443
_bottom Variable 443
_delay_ms Variable 444
_destpageaddr Variable 444
_left Variable 444
_page Variable 445
_param1 Variable 445
_param2 Variable 445
_right Variable 445
_srcpageaddr Variable 446
_top Variable 446
_transitionpending Variable 446
_type Variable 447
Height Variable 447
pixelWindow Variable 447
Startx Variable 448
Starty Variable 448
Width Variable 448

Image Decoders 449

ImageDecode Function 452

ImageDecoderInit Function 453

ImageLoopCallbackRegister Function 454

ImageDecodeTask Function 454

ImageFullScreenDecode Macro 455

ImageAbort Macro 456

Image Decoders Files 457
BmpDecoder.c 457
BmpDecoder.h 464
GifDecoder.c 464
GifDecoder.h 475
ImageDecoder.c 476
ImageDecoder.h 480
JpegDecoder.c 487
JpegDecoder.h 505
jidctint.c 507

_BMPDECODER Structure **512**

_GIFDECODER Structure **512**

_JPEGDECODER Structure **513**

Miscellaneous Topics 516

Starting a New Project **516**

Changing the default Font **522**

Using Primitive Rendering Functions in Blocking and Non-Blocking Modes **523**

Using Microchip Graphics Module Color Look Up Table in Applications **524**

Files 530

GOL Layer **530**

GOL.c 531

GOL.h 552

GOLFontDefault.c 579

Graphics.h 620

ScanCodes.h 622

AnalogClock.c 625

AnalogClock.h 637

Button.c 644

Button.h 655

Chart.c 663

Chart.h 708

CheckBox.c 725

CheckBox.h 732

DigitalMeter.c 736

DigitalMeter.h 742

EditBox.c 746

EditBox.h 753

Grid.c 758

Grid.h 766

GroupBox.c 773

GroupBox.h 778

ListBox.c 782

ListBox.h 794

Meter.c 804

Meter.h 817

Picture.c 826

Picture.h 830

ProgressBar.c 834
ProgressBar.h 840
RoundDial.c 844
RoundDial.h 852
RadioButton.c 858
RadioButton.h 866
Slider.c 872
Slider.h 886
StaticText.c 895
StaticText.h 900
TextEntry.c 904
TextEntry.h 919
GraphicsConfig.h 928
Window.c 932
Window.h 937

Primitive Layer **941**
Primitive.c 941
Primitive.h 1008

Device Driver Layer **1030**
drvTFT001.c 1031
drvTFT001.h 1057
HIT1270.c 1060
HIT1270.h 1075
HX8347.c 1077
HX8347.h 1095
SH1101A_SSD1303.c 1096
SH1101A_SSD1303.h 1104
SSD1339.c 1106
SSD1339.h 1122
SSD1926.c 1124
SSD1926.h 1168
TCON_HX8238.c 1187
TCON_SSD1289.c 1192
TCON_HX8257.c 1198
CustomDisplayDriver.c 1202
UC1610.c 1204
UC1610.h 1216
TCON_Custom.c 1218

References 1220

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 3.00 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 Microchip Software Bundle License.txt

MICROCHIP IS WILLING TO LICENSE THE ACCOMPANYING SOFTWARE AND DOCUMENTATION TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE. BY DOWNLOADING AND INSTALLING THE SOFTWARE, LICENSEE AGREES TO BE BOUND BY THE TERMS OF THIS AGREEMENT.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT FOR ACCOMPANYING MICROCHIP SOFTWARE AND DOCUMENTATION

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, your heirs, successors and assigns ("Licensee") and Microchip Technology Incorporated, a Delaware corporation, with a principal place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology (Barbados) II Incorporated (collectively, "Microchip") for the accompanying Microchip software including, but not limited to, Graphics Library Software, IrDA Stack Software, MCHPFSUSB Stack Software, Memory Disk Drive File System Software, mTouch(TM) Capacitive Library Software, Smart Card Library Software, TCP/IP Stack Software, MiWi(TM) DE Software, and/or any PC programs and any updates thereto (collectively, the "Software"), and accompanying documentation, including images and any other graphic resources provided by Microchip ("Documentation").

The Software and Documentation are licensed under this Agreement and not sold. U.S. copyright laws, international copyright treaties, and other intellectual property laws and treaties protect the Software and Documentation. Microchip reserves all rights not expressly granted to Licensee in this Agreement.

(1) Definitions. As used in this Agreement, the following capitalized terms will have the meanings defined below: a. "Microchip Products" means Microchip microcontrollers and Microchip digital signal controllers. b. "Licensee Products" means Licensee products that use or incorporate Microchip Products. c. "Object Code" means the Software computer programming code that is in binary form (including related documentation, if any), and error corrections, improvements, modifications, and updates. d. "Source Code" means the Software computer programming code that may be printed out or displayed in human readable form (including related programmer comments and documentation, if any), and error corrections, improvements, modifications, and updates. e. "Third Party" means Licensee’s agents, representatives, consultants, clients, customers, or contract manufacturers. f. "Third Party Products" means Third Party products that use or incorporate Microchip Products. (2) Software License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to: a. use the Software in connection with Licensee Products and/or Third Party Products; b. if Source Code is provided, modify the Software, provided that no Open Source Components (defined in Section 5 below) are incorporated into such Software in such a way that would affect Microchip’s right to distribute the Software with the limitations set forth herein and provided that Licensee clearly notifies Third Parties regarding such modifications; c. distribute the Software to Third Parties for use in Third Party Products, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept") and this Agreement accompanies such distribution; d. sublicense to a Third Party to use the Software, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept"); e. with respect to the TCP/IP Stack Software, Licensee may port the ENC28J60.c, ENC28J60.h, ENCX24J600.c, and ENCX24J600.h driver source files to a non-Microchip Product used in conjunction with a Microchip ethernet controller; f. with respect to the MiWi (TM) DE Software, Licensee may only exercise its rights when the Software is embedded on a Microchip Product and used with a Microchip radio frequency transceiver or UBEC UZ2400 radio frequency transceiver which are integrated into Licensee Products or Third Party Products. For purposes of clarity, Licensee may NOT embed the Software on a non-Microchip Product, except as described in this Section. (3) Documentation License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to use the Documentation in support of Licensee's authorized use of the Software (4) Third Party Requirements. Licensee acknowledges that it is Licensee’s responsibility to comply with any third party license terms or requirements applicable to the use of such third party software, specifications, systems, or tools. Microchip is not responsible and will not be held responsible in any manner for Licensee’s failure to comply with such applicable terms or requirements. (5) Open Source Components. Notwithstanding the license grant in Section 1 above, Licensee further acknowledges that certain components of the Software may be covered by so-called "open source" software licenses ("Open Source Components"). Open Source Components means any software

licenses approved as open source licenses by the Open Source Initiative or any substantially similar licenses, including without limitation any license that, as a condition of distribution of the software licensed under such license, requires that the distributor make the software available in source code format. To the extent required by the licenses covering Open Source Components, the terms of such license will apply in lieu of the terms of this Agreement, and Microchip hereby represents and warrants that the licenses granted to such Open Source Components will be no less broad than the license granted in Section 1. To the extent the terms of the licenses applicable to Open Source Components prohibit any of the restrictions in this Agreement with respect to such Open Source Components, such restrictions will not apply to such Open Source Component. (6) Licensee Obligations.

a. Licensee will ensure Third Party compliance with the terms of this Agreement, and will be liable for any breach of this Agreement committed by such Third Party. b. Licensee will not: (i) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (ii) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products or Third Party Products; or (iii) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof. c. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation. d. Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless from and against any and all claims, costs, damages, expenses (including reasonable attorney's fees), liabilities, and losses, including without limitation product liability claims, directly or indirectly arising from or related to: (i) the use, modification, disclosure or distribution of the Software, Documentation, or any intellectual property rights related thereto; (ii) the use, sale and distribution of Licensee Products or Third Party Products; and (iii) breach of this Agreement. THIS SECTION 3(d) STATES THE SOLE AND EXCLUSIVE LIABILITY OF THE PARTIES FOR INTELLECTUAL PROPERTY INFRINGEMENT. (7) Confidentiality. Licensee agrees that the Software (including but not limited to the Source Code, Object Code and library files) and its derivatives, Documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are proprietary information belonging to Microchip and its licensors ("Proprietary Information"). Except as expressly and unambiguously allowed herein, Licensee will hold in confidence and not use or disclose any Proprietary Information and will similarly bind its employees and Third Party(ies) in writing. Proprietary Information will not include information that: (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by the receiving party independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If Licensee is required to disclose Proprietary Information by law, court order, or government agency, License will give Microchip prompt notice of such requirement in order to allow Microchip to object or limit such disclosure. Licensee agrees that the provisions of this Agreement regarding unauthorized use and nondisclosure of the Software, Documentation and related Proprietary Rights are necessary to protect the legitimate business interests of Microchip and its licensors and that monetary damage alone cannot adequately compensate Microchip or its licensors if such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that Licensee or Third Party has breached or violated such provision then Microchip will have the right to petition for injunctive relief, without the requirement for the posting of a bond, in addition to all other remedies at law or in equity. (8) Ownership of Proprietary Rights. Microchip and its licensors retain all right, title and interest in and to the Software and Documentation ("Proprietary Rights") including, but not limited to all patent, copyright, trade secret and other intellectual property rights in the Software, Documentation, and underlying technology and all copies and derivative works thereof (by whomever produced). Further, copies and derivative works will be considered works made for hire with ownership vesting in Microchip on creation. To the extent such modifications and derivatives do not qualify as a "work for hire," Licensee hereby irrevocably transfers, assigns and conveys the exclusive copyright thereof to Microchip, free and clear of any and all liens, claims or other encumbrances, to the fullest extent permitted by law. Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in Sections this Agreement. (9) Termination of Agreement. Without prejudice to any other rights, this Agreement terminates immediately, without notice by Microchip, upon a failure by Licensee or Third Party to comply with any provision of this Agreement. Upon termination, Licensee and Third Party will immediately stop using the Software, Documentation, and derivatives thereof, and immediately destroy all such copies. (10) Warranty Disclaimers. THE 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. MICROCHIP AND ITS LICENSORS ASSUME NO RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-FREE. MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY DEFECTS IN THE SOFTWARE. LICENSEE AND THIRD PARTY ASSUME THE

ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION PROVIDED UNDER THIS AGREEMENT. (11) Limited Liability. IN NO EVENT WILL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO 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. The aggregate and cumulative liability of Microchip and its licensors for damages hereunder will in no event exceed $1000 or the amount Licensee paid Microchip for the Software and Documentation, whichever is greater. Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement. LICENSEE ACKNOWLEDGES AND AGREES THAT IT IS SOLELY RESPONSIBLE FOR THE LICENSEE PRODUCTS AND THIRD PARTY PRODUCTS, INCLUDING DETERMINING WHETHER SUCH PRODUCTS INFRINGE A PATENT, COPYRIGHT OR OTHER PROPRIETARY RIGHT OF ANY THIRD PARTY. LICENSEE AGREES THAT MICROCHIP HAS NO OBLIGATION TO INDEMNIFY OR DEFEND LICENSEE IN THE EVENT THAT A THIRD PARTY MAKES A CLAIM REGARDING LICENSEE PRODUCTS OR THIRD PARTY PRODUCTS. (12) General. THIS AGREEMENT WILL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. Licensee agrees that any disputes arising out of or related to this Agreement, Software or Documentation will be brought in the courts of the State of Arizona. The parties agree to waive their rights to a jury trial in actions relating to this Agreement. If either the Microchip or Licensee employs attorneys to enforce any rights arising out of or relating to this Agreement, the prevailing party will be entitled to recover its reasonable attorneys' fees, costs and other expenses. This Agreement will constitute the entire agreement between the parties with respect to the subject matter hereof. It will not be modified except by a written agreement signed by an authorized representative of the Microchip. Microchip’s authorized representatives will have the right to reasonably inspect Licensee's premises and to audit Licensee's records and inventory of Licensee Products in order to ensure Licensee's adherence to the terms of this Agreement. If any provision of this Agreement will be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision will be limited or eliminated to the minimum extent necessary so that this Agreement will otherwise remain in full force and effect and enforceable. No waiver of any breach of any provision of this Agreement will constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver will be effective unless made in writing and signed by an authorized representative of the waiving party. Licensee agrees to comply with all export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority. The indemnities, obligations of confidentiality, and limitations on liability described herein, and any right of action for breach of this Agreement prior to termination, will survive any termination of this Agreement. Neither this Agreement nor any rights, licenses or obligations hereunder, may be assigned by Licensee without the prior written approval of Microchip except pursuant to a merger, sale of all assets of Licensee or other corporate reorganization, provided that assignee agrees in writing to be bound by the Agreement. Any prohibited assignment will be null and void. Use, duplication or disclosure by the United States Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause of FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199.

If Licensee has any questions about this Agreement, please write to Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA. ATTN: Marketing.

Copyright (c) 2011 Microchip Technology Inc. All rights reserved.

License Rev. No. 03-060111

License Rev. No. 03-060111

3 Release Notes

Microchip Graphics Library Release Notes

V3.00

June-2011

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:

- 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
- For detailed usage, please refer to the External Programmer help file
- Added Analog Clock (see page 53) 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 (see page 362)() functions in Primitive Layer. For Driver Layer, this feature is enabled in the following drivers:
- mchpGfxDrv - Microchip Graphics Controller Driver
- S1D13517 - Epson Display 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 (see page 383)() 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 dir>/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
- Graphics Resource Converter output is used.
- 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 (see page 28)" directory in the demos to "Precompiled Hex"
- Combined all application note demos (AN1136, AN1182, AN1227 and AN1246) into one demo project (AppNotes).
- Moved the location of the COLOR_DEPTH (see page 41) setting from the HardwareProfile.h to the GraphicsConfig.h (see page 928)
- 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 (see page 28).
- 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 (see page 397) for new requirements.
- Modified Resistive Touch Driver calibration
- In the PIC24FJ256DA210 driver, CopyWindow (see page 386)() is modified to CopyBlock (see page 419)().

Fixes:

- Fixed issue on vertical Progress Bar (see page 352) rendering.
- Updated demos Google map and JPEG to use the proper GFX_RESOURCE (see page 368) 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 (see page 362)() 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 (see page 368)
- EXTDATA - replaced by GFX_EXTDATA (see page 370)
- BITMAP_FLASH - replaced by IMAGE_FLASH (see page 369)
- BITMAP_RAM - replaced by IMAGE_RAM (see page 370)
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA (see page 370)

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 (see page 1124) 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 (see page 41) macro has been moved from the hardware profile header file to the GraphicsConfig.h (see page 928) header file.
- For project migration please refer the graphics demos for examples.

Known Issues:

- PutImage (see page 362)() does not work when using PIC24FJ256DA210 and look up table is used on images located at EDS memory with color depth less than 8bpp.
- 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.

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

Technical Briefs

- TB3012 Conversion of Graphics PICtail™ Plus Board 2 for Compatibility with USB PICtail Plus

PIC Family

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

Development Tools

This graphics library release (v3.00) was tested with IDEs MPLAB v8.70 and MPLAB X beta 6; compilers C30 v3.24 and C32 v1.12a.

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.

There is a known compiler issue with C30 v3.30 ELF libraries, which is the only library format supported by MPLAB X beta 6. Using C30 v3.30 COFF libraries in MPLAB 8.xx will work.

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 (see page 373) section. Following graphics controllers are supported in this version:

Display Module	Interface	File Names
Microchip Display Driver MCHP_DA210 (for the future product PIC24FJ256DA210 Development Board)	8/16 bit EPMP	MicrochipGraphicsModule.c, MicrochipGraphicsModule.h
Samsung S6D0129/S6D0139	8/16 bit PMP	drvTFT001.c (see page 1031), drvTFT001.h (see page 1057)
LG LGDP4531	8/16 bit PMP	drvTFT001.c (see page 1031), drvTFT001.h (see page 1057)
Renesas R61505U/R61580	8/16 bit PMP	drvTFT001.c (see page 1031), drvTFT001.h (see page 1057)
Orise Technology SPDF5408	8/16 bit PMP	drvTFT001.c (see page 1031), drvTFT001.h (see page 1057)
Epson S1D13517	8/16 bit PMP	S1D13517.c, S1D13517.h

Solomon Systech SSD1926	8/16 bit PMP	SSD1926.c (see page 1124), SSD1926.h (see page 1168)
Solomon Systech SSD1289	8/16 bit PMP	drvTFT002.c, drvTFT002.h
Solomon Systech SSD1339 for OLED displays	8 bit PMP	SSD1339.c (see page 1106), SSD1339.h (see page 1122)
Solomon Systech SSD1303 for OLED displays	8 bit PMP	SH1101A_SSD1303.c (see page 1096), SH1101A_SSD1303.h (see page 1104)
Solomon Systech SSD2119	8/16 bit PMP	drvTFT002.c, drvTFT002.h
Sino Wealth SH1101A for OLED displays	8 bit PMP	SH1101A_SSD1303.c (see page 1096), SH1101A_SSD1303.h (see page 1104)
Sitronix ST7529	8 bit PMP	ST7529.c, ST7529.h
Hitech Electronics HIT1270	8 bit PMP	HIT1270.c (see page 1060), HIT1270.h (see page 1075)
Ilitek ILI9320	8/16 bit PMP	drvTFT001.c (see page 1031), drvTFT001.h (see page 1057)
Himax HX8347	8/16 bit PMP	HX8347.c (see page 1077), HX8347.h (see page 1095)
UltraChip UC1610	8 bit PMP	UC1610.c (see page 1204), UC1610.h (see page 1216)

Please refer to Adding New Device Driver (see page 397) section to get more information on adding support for another LCD controller.

Widgets

In this version the following widgets are implemented:

- Analog Clock (see page 53)
- Button (see page 65)
- Chart (see page 77)
- Checkbox (see page 115)
- Dial (see page 124)
- Digital Meter (see page 135)
- Edit Box (see page 143)
- Grid (see page 153)
- Group Box (see page 170)
- List Box (see page 177)
- Meter (see page 196)
- Picture Control (see page 210)
- Progress Bar (see page 218)
- Radio Button (see page 226)
- Slider (see page 237) / Scroll Bar (see page 237)
- Static Text (see page 252)
- Text Entry (see page 260)
- Window (see page 278)

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 (see page 338)	0	0	68	53	3375	8868
GOL (see page 48)	20 (per style scheme)	24 (per style scheme)	32	28	2076	5400
Button (see page 65)	28 (per instance)	44 (per instance)	8	12	1002	2748
Chart (see page 77)	48 (per instance)	76 (per instance)	94	104	11427	26364
Check Box (see page 115)	22 (per instance)	36 (per instance)	2	4	894	2320
Dial (see page 124)	30 (per instance)	40 (per instance)	8	12	1065	3228
Digital Meter (see page 135)	28 (per instance)	56 (per instance)	2	4	1125	2202
Edit Box (see page 143)	26 (per instance)	40 (per instance)	2	4	822	2332
Group Box (see page 170)	24 (per instance)	36 (per instance)	8	12	903	2164
List Box (see page 177)	28 (per instance), 12 (per item)	44 (per instance)	6	12	1809	2580
Meter (see page 196)	52 (per instance)	68 (per instance)	36	40	2778	6788
Picture Control (see page 210)	22 (per instance)	36 (per instance)	10	12	645	1512
Progress Bar (see page 218)	24 (per instance)	36 (per instance)	12	16	1050	2452
Radio Button (see page 226)	26 (per instance)	44 (per instance)	14	20	993	2632
Slider (see page 237) / Scroll Bar (see page 237)	32 (per instance)	44 (per instance)	20	24	2094	5720
Static Text (see page 252)	22 (per instance)	36 (per instance)	8	12	747	1884
Text Entry (see page 260)	34 (per instance), (24 per key)	52 (per instance)	22	28	2484	6376
Window (see page 278)	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

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 (see page 368) structure. This structure defines how the image(s) are accessed and processed by the Graphics Library.
- Added a third option (#define XCHAR (see page 345) unsigned char) on the XCHAR (see page 345) 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 GOLFontDefault (see page 332) in the library with any user defined fonts. Refer to "Changing the default Font (see page 522)" 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 (see page 368) type enumeration and expanded the enumeration for graphics resources (such as images and fonts). GFX_RESOURCE (see page 368) 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 (see page 48) structure OBJ_HEADER (see page 52). These function pointers makes it easier to add user created objects in the Graphics Library.
- DRAW_FUNC (see page 52) - function pointer to object drawing function.
- FREE_FUNC (see page 53) - 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 (see page 53) - function pointer to object message function.
- MSG_DEFAULT_FUNC (see page 53) - 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 (see page 295)().
- Added an option in XCHAR (see page 345) to use unsigned char.

Deprecated Items:

- TYPE_MEMORY - replaced by GFX_RESOURCE (see page 368)
- EXTDATA - replaced by GFX_EXTDATA (see page 370)
- BITMAP_FLASH - replaced by IMAGE_FLASH (see page 369)
- BITMAP_RAM - replaced by IMAGE_RAM (see page 370)
- BITMAP_EXTERNAL - replaced by GFX_EXTDATA (see page 370)

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 (see page 368) structure for images and new GFX_RESOURCE (see page 368) 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 (see page 153) object.
- Added Vertical Mode to Progress Bar (see page 352).
- 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 gfxpmp.c and gfxepmp.c to be inline functions in gfxpmp.h and gfxepmp.h respectively.
- Changed HX8347A.c to HX8347.c (see page 1077) (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 (see page 928) 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 (see page 43)(size) malloc(size)
- #define GFX_free (see page 43)(ptr) free(ptr)
- Merged GOL (see page 48) 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 (see page 237) increment/decrement issue on Keyboard messages.
- Fixed GOLGetFocusNext (see page 307)() 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 gfxpmp.h and gfxepmp.h source files in the project.
- gfxepmp.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 (see page 41) 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 (see page 620) 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 (see page 196) object.
- Fixed ListBox LbSetFocusedItem (see page 189)() 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 gfxepmp.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 (see page 28) - 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 (see page 928) to HardwareProfile.h.HardwareProfile.h references a hardware profile file in "Alternative Configurations" directory based on the PIC device selected.

Fixes:

- Fixed BtnSetText (see page 72)() 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 (see page 928) 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 (see page 306)() 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 (see page 928) 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 (see page 1124).
- Fixed remainder error in JPEG decoding in JpegDecoder.c (see page 487).
- 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 (see page 135) 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 (see page 65) Widget for new options:
- Use multi-line text (set USE_BUTTON_MULTI_LINE (see page 32))
- 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 (see page 42) is enabled, the accelerated primitive must be checked if it succesfully rendered. Refer to the coding example (see page 523) for details.
- Replaced LGDP4531_R61505_S6D0129_S6D0139_SPFD5408.c and LGDP4531_R61505_S6D0129_S6D0139_SPFD5408.h files with drvTFT001.c (see page 1031) and drvTFT001.h (see page 1057) 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 (see page 196) Widget for new options:
- Set Title and Value Font at creation time
- Added GOLGetFocusPrev (see page 308)() for focus control on GOL (see page 48) 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 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 3.5in 320x240 Board Rev A.pdf
- Schematic for Truly TOD9M0043.pdf

Fixes:

- Fixed drawing bug on TextEntry Widget
- Added missing documentation on Chart (see page 77) and Text Entry (see page 260) Widgets

Migration Changes:

- none

v1.60 Release (December 3, 2008)

Changes:

- Added TextEntry Widget.
- Modified Meter (see page 196) 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 (see page 196).
- Meter (see page 196) range is not defined by a minimum value and a maximum value.
- Added feature to a the Button (see page 65) Widget to allow cancelling of press by moving away the touch and releasing from the Button (see page 65)'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 (see page 928) 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 (see page 531) and GOL.h (see page 552) to include processing of TextEntry object when enabled by application.

v1.52 Release (August 29, 2008)

Changes:

- Added Chart (see page 77) Widget.
- Added Property State for Window (see page 278) 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 (see page 41) to define color depth of the display.
- Added GOL_EMBOSS_SIZE (see page 332) to be user defined in GraphicsConfig.h (see page 928).
- Simplified initialization code for SSD1906 driver.

Fixes:

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

Migration Changes:

- USE_GOL (see page 35) 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 (see page 332) can now be defined by the user in GraphicsConfig.h (see page 928). If user does not define this in GraphicsConfig.h (see page 928) the default value in GOL.h (see page 552) 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 (see page 374) 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 (see page 210) 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 (see page 37) compile switch to define XCHAR (see page 345) 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 LbDelItemsList (see page 193)() 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 (see page 352)() function erroneous call to MoveTo (see page 361)() 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 (see page 377)( ) and GetMaxY (see page 377)( ) macros.

v1.0 Release (November 1, 2007)

Changes:

- Edit Box, List Box, Meter (see page 196), Dial (see page 124) widgets are added.
- Button (see page 65) is modified. New options for the object are added.
- Modified OBJ_REMOVE definition to OBJ_HIDE (example BTN_REMOVE to BTN_HIDE (see page 67)).
- Modified GOLPanelDraw (see page 308)() function to include rounded panels.
- External memory support for the fonts and bitmaps is implemented.
- SSD1339 and LGDP4531 controllers support is added.
- Bevel (see page 357)(), FillBevel (see page 358)() and Arc (see page 356)() functions are added.
- Modified Graphics Object Layer Demo.
- Added Graphics External Memory Demo & Graphics PICtail™ 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 (see page 362)() 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 (see page 314)() doesn't check object drawing status. It should be called if GOL (see page 48) drawing is completed.
- GOLStartNewList() is replaced with GOLNewList (see page 302)().
- Line() function calls are replaced with Bar (see page 352)() function calls for vertical and horizontal lines.
- Parameter "change" is removed for SldIncVal() and SldDecVal() .
- Some optimization and cleanup.
- Slider (see page 237) API is changed:
- SldSetVal() is changed to SldSetPos (see page 245)()
- SldGetVal() is changed to SldGetPos (see page 246)()
- SldIncVal() is changed to SldIncPos (see page 248)()
- SldDecVal() is changed to SldDecPos (see page 249)()
- SldCreate (see page 241)() input parameter are changed:
- "delta" changed to "res"

Fixes:

- PutImage (see page 362)().
- Line().
- FillCircle (see page 355)().
- 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 (see page 363)(), GetImageHeight (see page 363)() are added.
- OutText (see page 343)(), OutTextXY (see page 343)() and GetTextWidth (see page 345)() functions are terminated by control characters (< 32).
- Graphics Objects Layer (GOL (see page 48)) is added.
- Button (see page 65), Slider (see page 237), Checkbox (see page 115), Radio Button (see page 65), Static Text, Picture (see page 210) control, Progress Bar (see page 352), Window (see page 278), Group Box are implemented.
- Touch screen support is added.

Fixes:

- ReadFlashByte().
- GetTextWidth (see page 345)().

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

4 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:

1. AC164127-3 - Graphics PICtail Plus Daughter Board with Truly 3.2" Display Kit

2. 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.

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

3. Graphics Display Boards available:

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

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

3. AC164127-8 - Graphics Display Truly 5.7" 640x480 Board
4. AC164127-9 - Graphics Display Truly 7" 800x480 Board

5. 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.

7. 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

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

```
#define PIC24FJ256DA210_DEV_BOARD
```

2. In the 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 (see page 84) 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
```

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 display controller
#define GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210
// set the display panel
#define GFX_USE_SIPLAY_PANEL_TFT_G240320LTSW_118W_E
// 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
```

4. In the the project's GraphicsConfig.h (see page 928) 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 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 (see page 84) 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 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 display panel
// set the PMP interface
#define USE_8BIT_PMP
```

4. In the the project's GraphicsConfig.h (see page 928) 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 application specific HardwareProfile.h file of your project, set the correct display controller.

```
#define GFX_USE_DISPLAY_CONTROLLER_S1D13517
```

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

4. In the the project's GraphicsConfig.h (see page 928) 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

1. 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 (see page 928) 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 *..\Microchip Solutions\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 IS0 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 (FON 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 (see page 48) as well as pages or screens of different Widgets (see page 48). Messaging and rendering of Widgets (see page 48) 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.

5 Demo Projects

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

5.1 Demo Summary

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

Description

Demo Name	Description
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.
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)
PIC24F_DA	A demo showing the graphics module in PIC24FJ256DA210.
JPEG	A demo showing Image Decoders and Graphics libraries are integrated into one application.
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.
PIC24H Starter Kit	Demo for the PIC24H Starter Kit.
Google Map	A demo showing the TCPIP Stack and Graphics Library combined in an application.

5.2 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

5.3 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.

6 Graphics Library Configuration

The library can be customized by adding or specifying the compile time options located in the application code named GraphicsConfig.h (see page 44) or the HardwareProfile.h files. The following compile time options are supported by the library.

6.1 Graphics Object Layer Configuration

Module

Graphics Library Configuration (see page 30)

Description

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

6.1.1 Input Device Selection

Macros

Name	Description
USE_KEYBOARD (see page 30)	Enable key board support.
USE_TOUCHSCREEN (see page 30)	Enable touch screen support.

Description

These compile options selects input devices used by GOL (see page 48) widgets. Remove definitions for unused input devices to reduce code size.

The USE_INPUTDEVICE options are located in the GraphicsConfig.h (see page 928) header file.

6.1.1.1 USE_KEYBOARD Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_KEYBOARD
```

Description

Enable key board support.

6.1.1.2 USE_TOUCHSCREEN Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_TOUCHSCREEN
```

Description

Enable touch screen support.

6.1.2 Focus Support Selection

Macros

Name	Description
USE_FOCUS (see page 31)	Keyboard control on some objects can be used by enabling the GOL (see page 48) Focus (USE_FOCUS)support.

Description

This compile option allows keyboard input focus. GOLSetFocus (see page 304)(), GOLGetFocus (see page 306)(), GOLCanBeFocused (see page 307)(), GOLGetFocusNext (see page 307)() functions will be available. Focus is also changed by touch screen.

The USE_FOCUS (see page 31) option is located in the GraphicsConfig.h (see page 928) header file.

6.1.2.1 USE_FOCUS Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_FOCUS
```

Overview

Keyboard control on some objects can be used by enabling the GOL (see page 48) Focus (USE_FOCUS)support.

6.1.3 Graphics Object Selection

Macros

Name	Description
USE_BUTTON (see page 32)	Enable Button (see page 65) Object.
USE_BUTTON_MULTI_LINE (see page 32)	Enable Multi-Line (see page 347) Button (see page 65) Object
USE_CHECKBOX (see page 32)	Enable Checkbox (see page 115) Object.
USE_DIGITALMETER (see page 33)	Enable DigitalMeter Object.
USE_EDITBOX (see page 33)	Enable Edit Box Object.
USE_GROUPBOX (see page 33)	Enable Group Box Object.
USE_LISTBOX (see page 33)	Enable List Box Object.
USE_METER (see page 33)	Enable Meter (see page 196) Object.
USE_PICTURE (see page 34)	Enable Picture (see page 210) Object.
USE_PROGRESSBAR (see page 34)	Enable Progress Bar (see page 352) Object.
USE_RADIOBUTTON (see page 34)	Enable Radio Button (see page 65) Object.

USE_ROUNDDIAL (see page 34)	Enable Dial (see page 124) Object.
USE_SLIDER (see page 35)	Enable Slider (see page 237) or Scroll (see page 419) Bar (see page 352) Object.
USE_STATICTEXT (see page 35)	Enable Static Text Object.
USE_WINDOW (see page 35)	Enable Window (see page 278) Object.
USE_CUSTOM (see page 35)	Enable Custom Control Object (an example to create customized Object).
USE_GOL (see page 35)	Enable Graphics Object Layer.
USE_ANALOGCLOCK (see page 36)	Enable Analog Clock Object.
USE_TEXTENTRY (see page 36)	Enable TextEntry Object.

Description

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

The USE_GOL (see page 35) and USE_OBJECT options are located in the GraphicsConfig.h (see page 928) header file. USE_GOL (see page 35) should be included if any of the objects are to be used.

6.1.3.1 USE_BUTTON Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_BUTTON
```

Description

Enable Button (see page 65) Object.

6.1.3.2 USE_BUTTON_MULTI_LINE Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_BUTTON_MULTI_LINE
```

Description

Enable Multi-Line (see page 347) Button (see page 65) Object

6.1.3.3 USE_CHECKBOX Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_CHECKBOX
```

Description

Enable Checkbox (see page 115) Object.

6.1.3.4 USE_DIGITALMETER Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_DIGITALMETER
```

Description

Enable DigitalMeter Object.

6.1.3.5 USE_EDITBOX Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_EDITBOX
```

Description

Enable Edit Box Object.

6.1.3.6 USE_GROUPBOX Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_GROUPBOX
```

Description

Enable Group Box Object.

6.1.3.7 USE_LISTBOX Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_LISTBOX
```

Description

Enable List Box Object.

6.1.3.8 USE_METER Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_METER
```

Description

Enable Meter (see page 196) Object.

6.1.3.9 USE_PICTURE Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_PICTURE
```

Description

Enable Picture (see page 210) Object.

6.1.3.10 USE_PROGRESSBAR Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_PROGRESSBAR
```

Description

Enable Progress Bar (see page 352) Object.

6.1.3.11 USE_RADIOBUTTON Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_RADIOBUTTON
```

Description

Enable Radio Button (see page 65) Object.

6.1.3.12 USE_ROUNDDIAL Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_ROUNDDIAL
```

Description

Enable Dial (see page 124) Object.

6.1.3.13 USE_SLIDER Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_SLIDER
```

Description

Enable Slider (see page 237) or Scroll (see page 419) Bar (see page 352) Object.

6.1.3.14 USE_STATICTEXT Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_STATICTEXT
```

Description

Enable Static Text Object.

6.1.3.15 USE_WINDOW Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_WINDOW
```

Description

Enable Window (see page 278) Object.

6.1.3.16 USE_CUSTOM Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_CUSTOM
```

Description

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

6.1.3.17 USE_GOL Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_GOL
```

Description

Enable Graphics Object Layer.

6.1.3.18 USE_ANALOGCLOCK Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_ANALOGCLOCK
```

Description

Enable Analog Clock Object.

6.1.3.19 USE_TEXTENTRY Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_TEXTENTRY
```

Description

Enable TextEntry Object.

6.2 Graphics Primitive Layer Configuration

Module

Graphics Library Configuration (see page 30)

Description

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

6.2.1 Font Type Selection

Macros

Name	Description
USE_MULTIBYTECHAR (see page 37)	To enable support for unicode fonts, USE_MULTIBYTECHAR must be defined. This changes XCHAR (see page 345) definition. See XCHAR (see page 345) for details.

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 USE_MULTIBYTECHAR (see page 37) - To enable support for unicode fonts, USE_MULTIBYTECHAR (see page 37) must be defined. This sets the XCHAR (see page 345) definition to use unsigned short.
- #define USE_UNSIGNED_XCHAR - To enable support for character range of 0-255, this must be defined. This sets the XCHAR (see page 345) definition to use unsigned char.
- to use the default signed char, do not define any of the two.

See XCHAR (see page 345) for details.

The font type selection (USE_MULTIBYTECHAR (see page 37) USE_UNSIGNED_CHAR) should be defined in the GraphicsConfig.h (see page 928) header file.

6.2.1.1 USE_MULTIBYTECHAR Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_MULTIBYTECHAR
```

Overview

To enable support for unicode fonts, USE_MULTIBYTECHAR must be defined. This changes XCHAR (see page 345) definition. See XCHAR (see page 345) for details.

6.3 Display Device Driver Layer Configuration

Macros

Name	Description
GFX_LCD_TYPE (see page 38)	Sets the type of display glass used. Define this in the Hardware Profile. • #define GFX_LCD_TYPE GFX_LCD_TFT (see page 39) - sets type TFT display • #define GFX_LCD_TYPE GFX_LCD_CSTN (see page 39) - sets type color STN display • #define GFX_LCD_TYPE GFX_LCD_MSTN (see page 39) - sets type mon STN display • #define GFX_LCD_TYPE GFX_LCD_OFF (see page 39) - display is turned off
STN_DISPLAY_WIDTH (see page 40)	Sets the STN glass data width. Define this in the Hardware Profile. • #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_4 (see page 40) - use 4-bit wide interface • #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_8 (see page 41) - Use 8-bit wide interface • #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_16 (see page 40) - Use 16-bit wide interface
COLOR_DEPTH (see page 41)	Specifies the color depth used in the demo.

Module

Graphics Library Configuration (see page 30)

Description

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

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 (see page 21) section.

The rest of the settings are used to specify the 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
COLOR_DEPTH (see page 41)	YES	YES	YES
DISP_DATA_WIDTH	YES	YES	YES
DISP_ORIENTATION	YES	YES	YES
DISP_HOR_FRONT_PORCH	NO	NO	YES
DISP_HOR_RESOLUTION	YES	YES	YES
DISP_VER_RESOLUTION	YES	YES	YES
DISP_HOR_BACK_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.

6.3.1 GFX_LCD_TYPE Macro

File

mchpGfxDrv.h

C

```
#define GFX_LCD_TYPE
```

Macros

Name	Description
GFX_LCD_CSTN (see page 39)	Type Color STN Display
GFX_LCD_MSTN (see page 39)	Type Mono STN Display
GFX_LCD_OFF (see page 39)	display is turned off

GFX_LCD_TFT (see page 39)	Type TFT Display

Overview

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

- #define GFX_LCD_TYPE GFX_LCD_TFT (see page 39) - sets type TFT display
- #define GFX_LCD_TYPE GFX_LCD_CSTN (see page 39) - sets type color STN display
- #define GFX_LCD_TYPE GFX_LCD_MSTN (see page 39) - sets type mon STN display
- #define GFX_LCD_TYPE GFX_LCD_OFF (see page 39) - display is turned off

6.3.1.1 GFX_LCD_CSTN Macro

File

SSD1926.h (see page 1168)

C

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

Description

Type Color STN Display

6.3.1.2 GFX_LCD_MSTN Macro

File

SSD1926.h (see page 1168)

C

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

Description

Type Mono STN Display

6.3.1.3 GFX_LCD_OFF Macro

File

mchpGfxDrv.h

C

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

Description

display is turned off

6.3.1.4 GFX_LCD_TFT Macro

File

SSD1926.h (see page 1168)

C

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

Description

Type TFT Display

6.3.2 STN_DISPLAY_WIDTH Macro

File

mchpGfxDrv.h

C

```
#define STN_DISPLAY_WIDTH
```

Macros

Name	Description
STN_DISPLAY_WIDTH_16 (see page 40)	display interface is 16 bits wide
STN_DISPLAY_WIDTH_4 (see page 40)	display interface is 4 bits wide
STN_DISPLAY_WIDTH_8 (see page 41)	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 (see page 40) - use 4-bit wide interface
- #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_8 (see page 41) - Use 8-bit wide interface
- #define STN_DISPLAY_WIDTH STN_DISPLAY_WIDTH_16 (see page 40) - Use 16-bit wide interface

6.3.2.1 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

6.3.2.2 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

6.3.2.3 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

6.3.3 COLOR_DEPTH Macro

File

GraphicsConfig.h (see page 928)

C

```
#define COLOR_DEPTH 16
```

Overview

Specifies the color depth used in the demo.

6.3.4 Graphics Hardware Platform

The following are the currently supported demo boards for the Graphics Library Demos (see page 28).

6.4 Application Configuration

Module

Graphics Library Configuration (see page 30)

Description

6.4.1 Configuration Setting

Macros

Name	Description
USE_NONBLOCKING_CONFIG (see page 42)	Comment this line to use blocking configuration

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 (see page 42) option is located in the GraphicsConfig.h (see page 928) header file.

6.4.1.1 USE_NONBLOCKING_CONFIG Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_NONBLOCKING_CONFIG
```

Description

Comment this line to use blocking configuration

6.4.2 Font Source Selection

Macros

Name	Description
USE_FONT_FLASH (see page 42)	Support for fonts located in internal flash

Description

Font data can be placed in multiple locations. Set these options in the GraphicsConfig.h (see page 928) header file.

- USE_FONT_FLASH (see page 42) - Font in internal flash memory support.

- USE_FONT_EXTERNAL - Font in external memory support.

- USE_FONT_RAM - Font in RAM or EDS support.

6.4.2.1 USE_FONT_FLASH Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_FONT_FLASH
```

Description

Support for fonts located in internal flash

6.4.3 Image Source Selection

Macros

Name	Description
USE_BITMAP_FLASH (see page 43)	Support for bitmaps located in internal flash

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 (see page 43) and USE_BITMAP_EXTERNAL options are located in the GraphicsConfig.h (see page 928) header file.

6.4.3.1 USE_BITMAP_FLASH Macro

File

GraphicsConfig.h (see page 928)

C

```
#define USE_BITMAP_FLASH
```

Description

Support for bitmaps located in internal flash

6.4.4 Miscellaneous

Macros

Name	Description
GFX_free (see page 43)	This is macro GFX_free.
GFX_malloc (see page 43)	Define the malloc() and free() for versatility on OS based systems.

Description

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

6.4.4.1 GFX_free Macro

File

GraphicsConfig.h (see page 928)

C

```
#define GFX_free(pObj) free(pObj)
```

Description

This is macro GFX_free.

6.4.4.2 GFX_malloc Macro

File

GraphicsConfig.h (see page 928)

C

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

Overview

Define the malloc() and free() for versatility on OS based systems.

6.4.5 GraphicsConfig.h Example

This is an example of the GraphicsConfig.h (see page 928) 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_ROUNDDIAL              // 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
```

6.5 Hardware Profile

Module

Graphics Library Configuration (see page 30)

Description

These functions and macros are used to determine hardware profile settings on the chosen PIC microcontroller and hardware such as demo boards used.

6.5.1 Device Driver Options

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 (see page 21) section.

The rest of the settings are used to specify the 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_FRONT_PORCH	NO	NO	YES
DISP_HOR_RESOLUTION	YES	YES	YES
DISP_VER_RESOLUTION	YES	YES	YES
DISP_HOR_BACK_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.

6.5.2 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_GCLK_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)

.....
```

7 Library Structure

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, draw functions delay the execution of program until drawing is done. For Non-Blocking configuration, draw functions do not wait for the drawing completion and release control to the program. Draw functions should be called repeatedly to complete the rendering. This allows efficient use of microcontroller time.

8 Graphics Object Layer

Types

Name	Description
GFX_COLOR (see page 337)	Data type that defines the color data.

Description

8.1 GOL Objects

The Graphics Object Layer (GOL (see page 48)) contains the Advanced Graphics Objects or commonly known as widgets.

Enumerations

Name	Description
GOL_OBJ_TYPE (see page 51)	This structure defines the Object types used in the library.

Module

Graphics Object Layer (see page 48)

Modules

Name	Description
Analog Clock (see page 53)	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 (see page 65)	Button is an Object that emulates a press and release effect when operated upon.
Chart (see page 77)	Chart is an Object that draws a bar chart or a pie chart representation of a single data or series of data.
Checkbox (see page 115)	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.
Dial (see page 124)	Dial is an Object that can be used to display emulate a turn dial that can both go in clockwise or counterclockwise.
Digital Meter (see page 135)	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 (see page 143)	Edit Box is is an Object that emulates a cell or a text area that can be edited dynamically.
Grid (see page 153)	Grid is an Object that draws a grid on the screen with each cell capable of displaying an image or a string.
Group Box (see page 170)	Group Box is an Object that can be used to group Objects together in the screen.
List Box (see page 177)	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 (see page 196)	Meter is an Object that can be used to graphically display a sampled input.
Picture Control (see page 210)	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 (see page 218)	Progress Bar (see page 352) is an Object that can be used to display the progress of a task such as a file download or transfer.
Radio Button (see page 226)	Radio Button (see page 65) 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/Scroll Bar (see page 237)	Slider or Scrollbar is an Object that can be used to display a value or scrolling location in a predefined area.
Static Text (see page 252)	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 (see page 260)	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 (see page 278)	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 (see page 352). It also has additional controls for both Title Bar (see page 352) and Client Area.

Structures

Name	Description
OBJ_HEADER (see page 52)	This structure defines the first nine fields of the Objects structure. This allows generic operations on library Objects.

Types

Name	Description
DRAW_FUNC (see page 52)	object draw function pointer typedef
FREE_FUNC (see page 53)	object free function pointer typedef
MSG_DEFAULT_FUNC (see page 53)	object default message function pointer typedef
MSG_FUNC (see page 53)	object message function pointer typedef

Description

All the GOL (see page 48) 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 (see page 48) objects.

Object	Type Definition
Button (see page 65)	OBJ_BUTTON
Chart (see page 77)	OBJ_CHART
Checkbox (see page 115)	OBJ_CHECKBOX
Dial (see page 124)	OBJ_ROUNDDIAL
Digital Meter (see page 135)	OBJ_DIGITALMETER
Edit Box (see page 143)	OBJ_EDITBOX
Grid (see page 153)	OBJ_GRID
Group Box (see page 170)	OBJ_GROUPBOX
List Box (see page 177)	OBJ_LISTBOX
Meter (see page 196)	OBJ_METER
Picture (see page 210)	OBJ_PICTURE
Progress Bar (see page 218)	OBJ_PROGRESSBAR

Radio Button (see page 226)	OBJ_RADIOBUTTON
Slider (see page 237)	OBJ_SLIDER
Static Text (see page 252)	OBJ_STATICTEXT
Text Entry (see page 260)	OBJ_TEXTENTRY
Window (see page 278)	OBJ_WINDOW

Each GOL (see page 48) 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 (see page 52)). Defining the OBJ_HEADER (see page 52) enables the common APIs to manage the objects of different types in the same manner.

The GOL (see page 48) 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 (see page 52) makes this possible. To manage the list a global pointer _pGolObjects (see page 334) 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 (see page 334) 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.

8.1.1 GOL_OBJ_TYPE Enumeration

File

GOL.h (see page 552)

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 (see page 65) Object.
OBJ_WINDOW	Type defined for Window (see page 278) Object.
OBJ_CHECKBOX	Type defined for Check Box Object.
OBJ_RADIOBUTTON	Type defined for Radio Button (see page 65) Object.
OBJ_EDITBOX	Type defined for Edit Box Object.
OBJ_LISTBOX	Type defined for List Box Object.
OBJ_SLIDER	Type defined for Slider (see page 237) and/or Scroll (see page 419) Bar (see page 352) Object.
OBJ_PROGRESSBAR	Type defined for Progress Object.
OBJ_STATICTEXT	Type defined for Static Text Object.
OBJ_PICTURE	Type defined for Picture (see page 210) or Bitmap Object.
OBJ_GROUPBOX	Type defined for Group Box Object.
OBJ_CUSTOM	Type defined for Custom Object.
OBJ_ROUNDDIAL	Type defined for Dial (see page 124) Object.
OBJ_METER	Type defined for Meter (see page 196) Object.
OBJ_GRID	Type defined for Grid (see page 153) Object.
OBJ_CHART	Type defined for Chart (see page 77) Object.
OBJ_TEXTENTRY	Type defined for Text-Entry Object.
OBJ_DIGITALMETER	Type defined for DIGITALMETER (see page 143) Object.
OBJ_ANALOGCLOCK	Type defined for ANALOGCLOCK (see page 64) Object.
OBJ_UNKNOWN	Type is undefined and not supported by the library.

Overview

This structure defines the Object types used in the library.

8.1.2 OBJ_HEADER Structure

File

GOL.h (see page 552)

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 (see page 48) 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.

8.1.3 DRAW_FUNC Type

File

GOL.h (see page 552)

C

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

Description

object draw function pointer typedef

8.1.4 FREE_FUNC Type

File

GOL.h (see page 552)

C

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

Description

object free function pointer typedef

8.1.5 MSG_DEFAULT_FUNC Type

File

GOL.h (see page 552)

C

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

Description

object default message function pointer typedef

8.1.6 MSG_FUNC Type

File

GOL.h (see page 552)

C

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

Description

object message function pointer typedef

8.1.7 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 (see page 56)	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 (see page 58)	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.
	AcHandsDraw (see page 59)	Draws the current position of the clock hands with the given thickness and color.
	AcSetHour (see page 60)	Sets the hour of the analog clock.
	AcSetMinute (see page 60)	Sets the minute of the analog clock.
	AcSetSecond (see page 61)	Sets the second of the analog clock.
	AcMsgDefault (see page 61)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	AcTranslateMsg (see page 62)	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.

Structures

Name	Description
ANALOGCLOCK (see page 64)	Defines the parameters required for a clock Object. The following relationships of the parameters determines the general shape of the button: 1. Width (see page 448) is determined by right - left. 2. Height (see page 447) 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*radius = height = width, the button is a circular button.

Description

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

8.1.7.1 Analog Clock States

Macros

Name	Description
AC_DRAW (see page 55)	Bit to indicate button must be redrawn.
AC_DISABLED (see page 55)	Bit for disabled state.
AC_HIDE (see page 55)	Bit to indicate button must be removed from screen.
AC_PRESSED (see page 55)	Bit for press state.

AC_TICK (see page 56)	Bit to tick second hand
UPDATE_HOUR (see page 56)	Bit to indicate hour hand must be redrawn
UPDATE_MINUTE (see page 56)	Bit to indicate minute hand must be redrawn
UPDATE_SECOND (see page 56)	Bit to indicate minute hand must be redrawn

Module

Analog Clock (see page 53)

8.1.7.1.1 AC_DRAW Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit to indicate button must be redrawn.

8.1.7.1.2 AC_DISABLED Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit for disabled state.

8.1.7.1.3 AC_HIDE Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit to indicate button must be removed from screen.

8.1.7.1.4 AC_PRESSED Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit for press state.

8.1.7.1.5 AC_TICK Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit to tick second hand

8.1.7.1.6 UPDATE_HOUR Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit to indicate hour hand must be redrawn

8.1.7.1.7 UPDATE_MINUTE Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit to indicate minute hand must be redrawn

8.1.7.1.8 UPDATE_SECOND Macro

File

AnalogClock.h (see page 637)

C

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

Description

Bit to indicate minute hand must be redrawn

8.1.7.2 AcCreate Function

File

AnalogClock.h (see page 637)

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 (see page 53)

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	set the current hour.
SHORT minute	set the current minute.
SHORT radius	radius of the clock.
BOOL sechand	Flag to draw cecond hand or not.
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.
GOL_SCHEME * pScheme	Pointer to the style scheme used.
pText	Pointer to the text of the button.

Returns

Returns the pointer to the object created.

Preconditions

none

Example

```
GOL_SCHEME *pScheme;
WORD state;

    pScheme = GOLCreateScheme();
    state = AC_DRAW;

    AnalogClock = AcCreate(1,20,64,50,118,0, state, NULL, "ON", pScheme);
Side Effects:none
```

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 (see page 64) *AcCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT radius, void *pBitmap, XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.7.3 AcDraw Function

File

AnalogClock.h (see page 637)

C

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

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
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 (see page 64) *pB)

8.1.7.4 AcHandsDraw Function

File

AnalogClock.h (see page 637)

C

```
WORD AcHandsDraw(
    ANALOGCLOCK * pAc,
    SHORT hand,
    SHORT thickness,
    WORD color,
    void * pBitmap
);
```

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT hand	which hand to be drawn (second, minute, hour)
SHORT thickness	thickness to draw the hand
WORD color	color to draw the hand *pBitmap - bitmap background to be redrawn

Side Effects

none

Returns

none

Preconditions

none

Overview

Draws the current position of the clock hands with the given thickness and color.

Syntax

AcHandsDraw(ANALOGCLOCK (see page 64) *pAc, SHORT hand, SHORT thickness, WORD color, void *pBitmap);

8.1.7.5 AcSetHour Function

File

AnalogClock.h (see page 637)

C

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

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT hour	current hour of the analog clock

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets the hour of the analog clock.

Syntax

AcSetHour(ANALOGCLOCK (see page 64) *pAc, SHORT hour)

8.1.7.6 AcSetMinute Function

File

AnalogClock.h (see page 637)

C

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

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT minute	current minute of the analog clock

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets the minute of the analog clock.

Syntax

AcSetMinute(ANALOGCLOCK (see page 64) *pAc, SHORT minute)

8.1.7.7 AcSetSecond Function

File

AnalogClock.h (see page 637)

C

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

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	The pointer to the object whose hands will be modified.
SHORT second	current minute of the analog clock

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets the second of the analog clock.

Syntax

AcSetMinute (see page 60)(ANALOGCLCOK *pAc, SHORT second)

8.1.7.8 AcMsgDefault Function

File

AnalogClock.h (see page 637)

C

```
void AcMsgDefault(
    WORD translatedMsg,
    ANALOGCLOCK * pAc,
```

```
    GOL_MSG * pMsg
);
```

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
ANALOGCLOCK * pAc	The pointer to the object whose state will be modified.
GOL_MSG * pMsg	The pointer to the GOL (see page 48) message.

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg (see page 75)() 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
################# ############ #####	#########
AC_MSG_PRESSED Touch Screen, Set AC_PRESSED (see page 55) Keyboard	
AC_MSG_RELEASED Touch Screen, Clear AC_PRESSED (see page 55) Keyboard	
AC_MSG_CANCELPRESS Touch Screen, Clear AC_PRESSED (see page 55)	

Syntax

AcMsgDefault(WORD translatedMsg, ANALOGCLOCK (see page 64) *pAc, GOL_MSG (see page 316)* pMsg)

8.1.7.9 AcTranslateMsg Function

File

AnalogClock.h (see page 637)

C

```
WORD AcTranslateMsg(
    ANALOGCLOCK * pAc,
    GOL_MSG * pMsg
);
```

Module

Analog Clock (see page 53)

Input Parameters

Input Parameters	Description
ANALOGCLOCK * pAc	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 (see page 48) message:

- AC_MSG_PRESSED – Clock is pressed
- AC_MSG_RELEASED – Clock is released
- AC_MSG_CANCELPRESS – Clock will be released, user cancels press action on the clock
- OBJ_MSG_INVALID – Clock 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)){
            switch(pCurrentObj->type){
                case OBJ_ANALOGCLOCK:
                    objMsg = AcTranslateMsg((ANALOGCLOCK*)pCurrentObj, pMsg);
                    if(objMsg == OBJ_MSG_INVALID)
                        break;
                    if(GOLMsgCallback(objMsg,pCurrentObj,pMsg))
                        AcMsgDefault(objMsg,(ANALOGCLOCK*)pCurrentObj);
                    break;
                case OBJ_SLIDER:
                    objMsg = SldTranslateMsg((SLIDER*)pCurrentObj, pMsg);
                    if(objMsg == OBJ_MSG_INVALID)
                        break;
                    if(GOLMsgCallback(objMsg,pCurrentObj,pMsg))
                        SldMsgDefault(objMsg,(SLIDER*)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 and keyboard inputs.

Translated Message Input Source Set/Clear State Bit	Description
################# ############ #######################	##
AC_MSG_PRESSED Touch Screen EVENT_PRESS, EVENT_MOVE Keyboard EVENT_KEYSCAN	
AC_MSG_STILLPRESSED Touch Screen EVENT_STILLPRESS	
AC_MSG_RELEASED Touch Screen EVENT_RELEASE Keyboard EVENT_KEYSCAN	
AC_MSG_CANCELPRESS Touch Screen EVENT_MOVE	
AC_MSG_INVALID Any Any	

Syntax

AcTranslateMsg(BUTTON (see page 77) *pB, GOL_MSG (see page 316) *pMsg)

8.1.7.10 ANALOGCLOCK Structure

File

AnalogClock.h (see page 637)

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 (see page 52)).
SHORT radius;	Radius of the clock.
void * pBitmap;	Pointer to bitmap used.

Module

Analog Clock (see page 53)

Overview

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

1. Width (see page 448) is determined by right - left.
2. Height (see page 447) 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*radius = height = width, the button is a circular button.

8.1.8 Button

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

Functions

	Name	Description
	BtnCreate (see page 69)	This function creates a BUTTON (see page 77) 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 (see page 70)	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 (see page 70)
	BtnSetText (see page 72)	This function sets the string used for the object.
	BtnMsgDefault (see page 74)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	BtnTranslateMsg (see page 75)	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
BtnGetText (see page 72)	This macro returns the address of the current text string used for the object.
BtnGetBitmap (see page 73)	This macro returns the location of the currently used bitmap for the object.
BtnSetBitmap (see page 74)	This macro sets the bitmap used in the object. The size of the bitmap must match the face of the button.

Structures

Name	Description
BUTTON (see page 77)	Defines the parameters required for a button Object. The following relationships of the parameters determines the general shape of the button: 1. Width (see page 448) is determined by right - left. 2. Height (see page 447) 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*radius = height = 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.

8.1.8.1 Button States

Macros

Name	Description
BTN_DISABLED (see page 66)	Bit for disabled state.
BTN_DRAW (see page 67)	Bit to indicate button must be redrawn.
BTN_DRAW_FOCUS (see page 67)	Bit to indicate focus must be redrawn.
BTN_FOCUSED (see page 67)	Bit for focus state.
BTN_HIDE (see page 67)	Bit to indicate button must be removed from screen.
BTN_PRESSED (see page 67)	Bit for press state.
BTN_TEXTBOTTOM (see page 68)	Bit to indicate text is top aligned.
BTN_TEXTLEFT (see page 68)	Bit to indicate text is left aligned.
BTN_TEXTRIGHT (see page 68)	Bit to indicate text is right aligned.
BTN_TEXTTOP (see page 68)	Bit to indicate text is bottom aligned.
BTN_TOGGLE (see page 68)	Bit to indicate button will have a toggle behavior.
BTN_TWOTONE (see page 69)	Bit to indicate the button is a two tone type.
BTN_NOPANEL (see page 69)	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 (see page 65)

Description

List of Button (see page 65) bit states.

8.1.8.1.1 BTN_DISABLED Macro

File

Button.h (see page 655)

C

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

Description

Bit for disabled state.

8.1.8.1.2 BTN_DRAW Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate button must be redrawn.

8.1.8.1.3 BTN_DRAW_FOCUS Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate focus must be redrawn.

8.1.8.1.4 BTN_FOCUSED Macro

File

Button.h (see page 655)

C

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

Description

Bit for focus state.

8.1.8.1.5 BTN_HIDE Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate button must be removed from screen.

8.1.8.1.6 BTN_PRESSED Macro

File

Button.h (see page 655)

C

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

Description

Bit for press state.

8.1.8.1.7 BTN_TEXTBOTTOM Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate text is top aligned.

8.1.8.1.8 BTN_TEXTLEFT Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate text is left aligned.

8.1.8.1.9 BTN_TEXTRIGHT Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate text is right aligned.

8.1.8.1.10 BTN_TEXTTOP Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate text is bottom aligned.

8.1.8.1.11 BTN_TOGGLE Macro

File

Button.h (see page 655)

C

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

Description

Bit to indicate button will have a toggle behavior.

8.1.8.1.12 BTN_TWOTONE Macro

File

Button.h (see page 655)

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.

8.1.8.1.13 BTN_NOPANEL Macro

File

Button.h (see page 655)

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).

8.1.8.2 BtnCreate Function

File

Button.h (see page 655)

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 (see page 65)

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 (see page 77) 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 (see page 77) *BtnCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, SHORT radius, void *pBitmap, XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.8.3 BtnDraw Function

File

Button.h (see page 655)

C

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

Module

Button (see page 65)

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)

8.1.8.4 BtnGetText Macro

File

Button.h (see page 655)

C

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

Module

Button (see page 65)

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)

8.1.8.5 BtnSetText Function

File

Button.h (see page 655)

C

```
void BtnSetText(
    BUTTON * pB,
    XCHAR * pText
);
```

Module

Button (see page 65)

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 (see page 77) *pB, XCHAR (see page 345) *pText)

8.1.8.6 BtnGetBitmap Macro

File

Button.h (see page 655)

C

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

Module

Button (see page 65)

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)

8.1.8.7 BtnSetBitmap Macro

File

Button.h (see page 655)

C

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

Module

Button (see page 65)

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)

8.1.8.8 BtnMsgDefault Function

File

Button.h (see page 655)

C

```
void BtnMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Button (see page 65)

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 (see page 48) message.

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg (see page 75)() 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 (see page 67)	Button (see page 65) will be redrawn in the pressed state.
BTN_MSG_RELEASED	Touch Screen, Keyboard	Clear BTN_PRESSED (see page 67)	Button (see page 65) will be redrawn in the unpressed state.
BTN_MSG_CANCELPRESS	Touch Screen,	Clear BTN_PRESSED (see page 67)	Button (see page 65) will be redrawn in the unpressed state.

Syntax

BtnMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.8.9 BtnTranslateMsg Function

File

Button.h (see page 655)

C

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

Module

Button (see page 65)

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 (see page 48) message:

- BTN_MSG_PRESSED – Button (see page 65) is pressed
- BTN_MSG_RELEASED – Button (see page 65) is released
- BTN_MSG_CANCELPRESS – Button (see page 65) will be released, user cancels press action on the button
- OBJ_MSG_INVALID – Button (see page 65) 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 (see page 321) or SCAN_SPACE_PRESSED (see page 325) while the button is not pressed.
BTN_MSG_STILLPRESSED	Touch Screen	EVENT_STILLPRESS	If event occurs and and the x,y position falls in the face of the button. Current state of the button is not checked.
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 (see page 321) or SCAN_SPACE_RELEASED (see page 325) 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 (see page 316) *pMsg)

8.1.8.10 BUTTON Structure

File

Button.h (see page 655)

C

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

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (see page 52)).
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 (see page 65)

Overview

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

1. Width (see page 448) is determined by right - left.
2. Height (see page 447) 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*radius = height = width, the button is a circular button.

8.1.9 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 (see page 86)	This function creates a CHART (see page 110) 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 (see page 87)	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 (see page 102)() function. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is... more (see page 87)
	ChAddDataSeries (see page 88)	This function creates a DATASERIES (see page 110) object and populates the structure with the given parameters.
	ChRemoveDataSeries (see page 89)	This function removes DATASERIES (see page 110) object from the list of DATASERIES (see page 110) 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 (see page 95)	This function sets the minimum and maximum range of values that the bar chart will show. The criteria is that min <= max.
	ChSetPercentRange (see page 98)	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 (see page 99)	This function sets the sample start and sample end when drawing the chart. Together with the data series' SHOW_DATA (see page 84) flags the different way of displaying the chart data is achieved.
	ChFreeDataSeries (see page 108)	This function removes DATASERIES (see page 110) object from the list of DATASERIES (see page 110) objects and frees the memory used of that removed object.
	ChTranslateMsg (see page 109)	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 (see page 90)	This macro sets the specified data series number show flag to be set to SHOW_DATA (see page 84).
ChHideSeries (see page 91)	This macro sets the specified data series number show flag to be set to HIDE_DATA (see page 84).
ChGetShowSeriesCount (see page 91)	This macro shows the number of data series that has its show flag set to SHOW_DATA (see page 84)
ChGetShowSeriesStatus (see page 92)	This macro returns the show ID status of the DATASERIES (see page 110).
ChSetValueLabel (see page 92)	This macro sets the address of the current text string used for the value axis label of the bar chart.
ChGetValueLabel (see page 93)	This macro returns the address of the current text string used for the value axis label of the bar chart.
ChGetValueMax (see page 94)	This macro returns the current maximum value that will be drawn for bar charts.
ChGetValueMin (see page 94)	This macro returns the current minimum value that will be drawn for bar charts.
ChGetValueRange (see page 95)	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 (see page 94)() and to get the maximum use ChGetValueMax (see page 94)().
ChSetSampleLabel (see page 96)	This macro sets the address of the current text string used for the sample axis label of the bar chart.

ChGetSampleLabel (see page 97)	This macro returns the address of the current text string used for the sample axis label of the bar chart.
ChGetSampleStart (see page 97)	This macro returns the sampling start value.
ChGetSampleEnd (see page 98)	This macro returns the sampling end value.
ChGetPercentRange (see page 99)	This macro gets the percentage range for bar charts. The value returned is calculated from percentage max - min. To get the minimum use ChGetPercentMin (see page 102)() and to get the maximum use ChGetPercentMax (see page 101)().
ChGetSampleRange (see page 100)	This macro gets the sample range for pie or bar charts. The value returned is calculated from smplEnd - smplStart.
ChGetPercentMax (see page 101)	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 (see page 102)	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 (see page 102)	This macro sets the color table used to draw the data in pie and bar charts.
ChGetColorTable (see page 103)	This macro returns the current color table used for the pie and bar charts.
ChSetTitle (see page 103)	This macro sets the address of the current text string used for the title of the chart.
ChGetTitle (see page 104)	This macro returns the address of the current text string used for the title of the chart.
ChSetTitleFont (see page 104)	This macro sets the location of the font used for the title of the chart.
ChGetTitleFont (see page 105)	This macro returns the location of the font used for the title of the chart.
ChGetAxisLabelFont (see page 106)	This macro returns the location of the font used for the X and Y axis labels of the chart.
ChSetAxisLabelFont (see page 106)	This macro sets the location of the font used for the X and Y axis labels of the chart.
ChGetGridLabelFont (see page 107)	This macro returns the location of the font used for the X and Y axis grid labels of the chart.
ChSetGridLabelFont (see page 107)	This macro sets the location of the font used for the X and Y axis grid labels of the chart.

Structures

Name	Description
CHART (see page 110)	Defines the parameters required for a chart Object.
DATASERIES (see page 110)	Defines a variable for the CHART (see page 110) 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 (see page 111)	Defines the parameters for the CHART (see page 110) 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.

8.1.9.1 Chart States

Macros

Name	Description
CH_DISABLED (see page 81)	Bit for disabled state.
CH_DRAW (see page 81)	Bit to indicate chart must be redrawn.
CH_DRAW_DATA (see page 81)	Bit to indicate data portion of the chart must be redrawn.
CH_3D_ENABLE (see page 81)	Bit to indicate that bar charts are to be drawn with 3-D effect
CH_BAR (see page 82)	Bit to indicate the chart is type bar. If both PIE and BAR types
CH_BAR_HOR (see page 82)	These bits (with CH_BAR (see page 82) bit set), sets the bar chart to
CH_DONUT (see page 82)	These bits (with CH_PIE (see page 83) bit set), sets the pie chart to
CH_LEGEND (see page 82)	Bit to indicate that legend is to be shown. Usable only when seriesCount > 1.
CH_NUMERIC (see page 82)	This bit is used only for bar charts. If this bit is set, it indicates that the
CH_PERCENT (see page 83)	Bit to indicate that the pie chart will be drawn with percentage

CH_PIE (see page 83)	Bit to indicate the chart is type pie. If both PIE and BAR types
CH_VALUE (see page 83)	Bit to indicate that the values of the bar chart data or pie chart
CH_HIDE (see page 83)	Bit to indicate chart must be removed from screen.

Module

Chart (see page 77)

Description

List of Chart (see page 77) bit states.

8.1.9.1.1 CH_DISABLED Macro

File

Chart.h (see page 708)

C

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

Description

Bit for disabled state.

8.1.9.1.2 CH_DRAW Macro

File

Chart.h (see page 708)

C

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

Description

Bit to indicate chart must be redrawn.

8.1.9.1.3 CH_DRAW_DATA Macro

File

Chart.h (see page 708)

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.

8.1.9.1.4 CH_3D_ENABLE Macro

File

Chart.h (see page 708)

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

8.1.9.1.5 CH_BAR Macro

File

Chart.h (see page 708)

C

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

Description

Bit to indicate the chart is type bar. If both PIE and BAR types

8.1.9.1.6 CH_BAR_HOR Macro

File

Chart.h (see page 708)

C

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

Description

These bits (with CH_BAR (see page 82) bit set), sets the bar chart to

8.1.9.1.7 CH_DONUT Macro

File

Chart.h (see page 708)

C

```
#define CH_DONUT 0x0140          // These bits (with CH_PIE bit set), sets the pie chart to
```

Description

These bits (with CH_PIE (see page 83) bit set), sets the pie chart to

8.1.9.1.8 CH_LEGEND Macro

File

Chart.h (see page 708)

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.

8.1.9.1.9 CH_NUMERIC Macro

File

Chart.h (see page 708)

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

8.1.9.1.10 CH_PERCENT Macro

File

Chart.h (see page 708)

C

```
#define CH_PERCENT 0x0010          // Bit to indicate that the pie chart will be drawn with
percentage
```

Description

Bit to indicate that the pie chart will be drawn with percentage

8.1.9.1.11 CH_PIE Macro

File

Chart.h (see page 708)

C

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

Description

Bit to indicate the chart is type pie. If both PIE and BAR types

8.1.9.1.12 CH_VALUE Macro

File

Chart.h (see page 708)

C

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

Description

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

8.1.9.1.13 CH_HIDE Macro

File

Chart.h (see page 708)

C

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

Description

Bit to indicate chart must be removed from screen.

8.1.9.2 Data Series Status Settings

Macros

Name	Description
HIDE_DATA (see page 84)	Macro used to reset the data series show flag or indicate that the data series will be not be shown when the chart is drawn.
SHOW_DATA (see page 84)	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 (see page 77)

Description

Data Series show status flag settings.

8.1.9.2.1 HIDE_DATA Macro

File

Chart.h (see page 708)

C

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

8.1.9.2.2 SHOW_DATA Macro

File

Chart.h (see page 708)

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.

8.1.9.3 Chart Examples

Examples of generated bar charts based on settings.

Module

Chart (see page 77)

Description

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

Example 1

CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series

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

ChSetSampleRange(1,2);

smplStart smplEnd

Example 2

CH_NUMERIC = 0
SERIESDATA Series
SERIESDATA Series

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

smplStart smplEnd

Example 3

CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series

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

smplStart smplEnd

Example 4

CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series

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

ChSetSampleRange(1,1);

smplStart
smplEnd

Example 5

CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series

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

smplStart
smplEnd

More Customization...

CH_NUMERIC = 1
SERIESDATA Series
SERIESDATA Series

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

smplStart smplEnd

*pTitle	Animal Trend
*pSmplLabel	Year
*pValLabel	Number of Animal

8.1.9.4 ChCreate Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 GOLSmallFont;
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(),
                      GetMaxY(),
                      state,                        // state of the chart
                      NULL,                         // data not initialized yet
                      NULL,                         // no parameters yet
                      pScheme);                     // style scheme used

    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*)&GOLSmallFont);
    ChSetAxisLabelFont(pChart, (void*)&GOLSmallFont);

    // 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 (see page 110) 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 (see page 110) *ChCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, DATASERIES (see page 110) *pData, CHARTPARAM (see page 111) *pParam, GOL_SCHEME (see page 331) *pScheme)

8.1.9.5 ChDraw Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 102)() 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)

8.1.9.6 ChAddDataSeries Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 110) that the chart is operating on. By default, the show flag of the newly added data series is set to SHOW_DATA (see page 84) or enabled.

Returns

Returns the pointer to the data variable (DATASERIES (see page 110)) 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 (see page 86)() example.

Overview

This function creates a DATASERIES (see page 110) object and populates the structure with the given parameters.

Syntax

ChAddDataSeries(CHART (see page 110) *pCh, WORD nSamples, WORD *pData, XCHAR (see page 345) *pName)

8.1.9.7 ChRemoveDataSeries Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 110) object from the list of DATASERIES (see page 110) 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 (see page 110) *pCh, WORD number)

8.1.9.8 ChShowSeries Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 84).

Side Effects

none

Returns

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

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 (see page 84).

Syntax

SHORT ChShowSeries(CHART (see page 110) *pCh, WORD seriesNum)

8.1.9.9 ChHideSeries Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 84).

Side Effects

none

Returns

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

Preconditions

none

Example

See ChShowSeries (see page 90)() example.

Overview

This macro sets the specified data series number show flag to be set to HIDE_DATA (see page 84).

Syntax

SHORT ChHideSeries(CHART (see page 110) *pCh, WORD seriesNum)

8.1.9.10 ChGetShowSeriesCount Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 84).

Preconditions

none

Overview

This macro shows the number of data series that has its show flag set to SHOW_DATA (see page 84)

Syntax

ChGetShowSeriesCount(pCh)

8.1.9.11 ChGetShowSeriesStatus Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

Input Parameters

Input Parameters	Description
pDSeries	Pointer to the data series(DATASERIES (see page 110)) that is being checked.

Side Effects

none

Returns

Returns the status of the show flag. 1 - (SHOW_DATA (see page 84)) means that the show status flag is set. 0 - (HIDE_DATA (see page 84)) means that the show status flag is not set.

Preconditions

none

Overview

This macro returns the show ID status of the DATASERIES (see page 110).

Syntax

ChGetShowSeriesStatus(pDSeries)

8.1.9.12 ChSetValueLabel Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() 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)

8.1.9.13 ChGetValueLabel Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.14 ChGetValueMax Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.15 ChGetValueMin Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.16 ChSetValueRange Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 110) *pCh, WORD min, WORD max)

8.1.9.17 ChGetValueRange Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 94)() and to get the maximum use ChGetValueMax (see page 94)().

Syntax

ChGetValueRange(pCh)

8.1.9.18 ChSetSampleLabel Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() 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)

8.1.9.19 ChGetSampleLabel Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.20 ChGetSampleStart Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.21 ChGetSampleEnd Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.22 ChSetPercentRange Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 110) *pCh, WORD min, WORD max)

8.1.9.23 ChGetPercentRange Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 102)() and to get the maximum use ChGetPercentMax (see page 101)().

Syntax

ChGetPercentRange(pCh)

8.1.9.24 ChSetSampleRange Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() example.

Overview

This function sets the sample start and sample end when drawing the chart. Together with the data series' SHOW_DATA (see page 84) flags the different way of displaying the chart data is achieved.

Start & End Value	The # of Data Series Flag Set	Chart (see page 77) Description
Start <= End	1	Show the data indicated by Start and End points of the DATASERIES (see page 110) with the flag set
Start = End	1	Show the data indicated by Start or End points of the DATASERIES (see page 110) with the flag set
Start, End = don't care	> 1	Show the data indicated by Start point of the DATASERIES (see page 110) with the flag set. Each samples of all checked data series are grouped together according to sample number.

Syntax

ChSetSampleRange(CHART (see page 110) *pCh, WORD start, WORD end)

8.1.9.25 ChGetSampleRange Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.26 ChGetPercentMax Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.27 ChGetPercentMin Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.28 ChSetColorTable Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.29 ChGetColorTable Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.30 ChSetTitle Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() example.

Overview

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

Syntax

ChSetTitle(pCh, pNewTitle)

8.1.9.31 ChGetTitle Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.32 ChSetTitleFont Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() example.

Overview

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

Syntax

ChSetTitleFont(pCh, pNewFont)

8.1.9.33 ChGetTitleFont Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.34 ChGetAxisLabelFont Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.35 ChSetAxisLabelFont Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() 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)

8.1.9.36 ChGetGridLabelFont Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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)

8.1.9.37 ChSetGridLabelFont Macro

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 86)() 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)

8.1.9.38 ChFreeDataSeries Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 110) object from the list of DATASERIES (see page 110) objects and frees the memory used of that removed object.

Syntax

void ChFreeDataSeries(void *pObj)

8.1.9.39 ChTranslateMsg Function

File

Chart.h (see page 708)

C

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

Module

Chart (see page 77)

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 (see page 48) message:

- CH_MSG_SELECTED – Chart (see page 77) area is selected
- OBJ_MSG_INVALID – Chart (see page 77) 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 (see page 316) *pMsg)

8.1.9.40 CHART Structure

File

Chart.h (see page 708)

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 (see page 52)).
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 (see page 77)

Overview

Defines the parameters required for a chart Object.

8.1.9.41 DATASERIES Structure

File

Chart.h (see page 708)

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 (see page 84), the data series will be shown. If HIDE_DATA (see page 84), 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 (see page 77)

Overview

Defines a variable for the CHART (see page 110) 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.

8.1.9.42 CHARTPARAM Structure

File

Chart.h (see page 708)

C

```
typedef struct {
  XCHAR * pTitle;
  XCHAR * pSmplLabel;
  XCHAR * pValLabel;
  SHORT seriesCount;
  WORD smplStart;
  WORD smplEnd;
  WORD valMax;
  WORD valMin;
  WORD perMax;
  WORD perMin;
  WORD * 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.
WORD * 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.
void * pAxisLabelsFont;	Pointer to the font used for X and Y axis labels.
void * pGridLabelsFont;	Pointer to the font used for X and Y axis grid labels.

Module

Chart (see page 77)

Overview

Defines the parameters for the CHART (see page 110) object.

8.1.9.43 Color Table

Macros

Name	Description
CH_CLR0 (see page 112)	Bright Blue

CH_CLR1 (see page 112)	Bright Red
CH_CLR2 (see page 112)	Bright Green
CH_CLR3 (see page 113)	Bright Yellow
CH_CLR4 (see page 113)	Orange
CH_CLR5 (see page 113)	Blue
CH_CLR6 (see page 113)	Red
CH_CLR7 (see page 113)	Green
CH_CLR8 (see page 114)	Yellow
CH_CLR9 (see page 114)	Dark Orange
CH_CLR10 (see page 114)	Light Orange
CH_CLR11 (see page 114)	Light Red
CH_CLR12 (see page 114)	Light Green
CH_CLR13 (see page 115)	Light Yellow
CH_CLR14 (see page 115)	Light Orange
CH_CLR15 (see page 115)	Gold

Module

Chart (see page 77)

Description

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

8.1.9.43.1 CH_CLR0 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR0 BRIGHTBLUE      // Bright Blue
```

Description

Bright Blue

8.1.9.43.2 CH_CLR1 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR1 BRIGHTRED       // Bright Red
```

Description

Bright Red

8.1.9.43.3 CH_CLR2 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR2 BRIGHTGREEN     // Bright Green
```

Description

Bright Green

8.1.9.43.4 CH_CLR3 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR3 BRIGHTYELLOW    // Bright Yellow
```

Description

Bright Yellow

8.1.9.43.5 CH_CLR4 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR4 RGBConvert(0xFF, 0xBB, 0x4C) // Orange
```

Description

Orange

8.1.9.43.6 CH_CLR5 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR5 BLUE                            // Blue
```

Description

Blue

8.1.9.43.7 CH_CLR6 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR6 RED                             // Red
```

Description

Red

8.1.9.43.8 CH_CLR7 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR7 GREEN                          // Green
```

Description

Green

8.1.9.43.9 CH_CLR8 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR8 YELLOW                         // Yellow
```

Description

Yellow

8.1.9.43.10 CH_CLR9 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR9 RGBConvert(255, 140, 0)      // Dark Orange
```

Description

Dark Orange

8.1.9.43.11 CH_CLR10 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR10 LIGHTBLUE                      // Light Orange
```

Description

Light Orange

8.1.9.43.12 CH_CLR11 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR11 LIGHTRED                       // Light Red
```

Description

Light Red

8.1.9.43.13 CH_CLR12 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR12 LIGHTGREEN                    // Light Green
```

Description

Light Green

8.1.9.43.14 CH_CLR13 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR13 RGBConvert(255, 255, 150)    // Light Yellow
```

Description

Light Yellow

8.1.9.43.15 CH_CLR14 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR14 RGBConvert(255, 200, 0)      // Light Orange
```

Description

Light Orange

8.1.9.43.16 CH_CLR15 Macro

File

Chart.h (see page 708)

C

```
#define CH_CLR15 RGBConvert(255, 215, 0)      // Gold
```

Description

Gold

8.1.10 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 (see page 119)	This function creates a CHECKBOX (see page 124) 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 (see page 120)	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 (see page 121)	This function sets the text that will be used.
	CbMsgDefault (see page 122)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	CbTranslateMsg (see page 123)	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 (see page 121)	This macro returns the location of the text used for the check box.

Structures

Name	Description
CHECKBOX (see page 124)	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 unchecked.

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

8.1.10.1 Check Box States

Macros

Name	Description
CB_CHECKED (see page 117)	Checked state
CB_DISABLED (see page 118)	Disabled state
CB_DRAW (see page 118)	Whole check box must be redrawn
CB_DRAW_CHECK (see page 118)	Check box mark should be redrawn
CB_DRAW_FOCUS (see page 118)	Focus must be redrawn
CB_FOCUSED (see page 118)	Focus state
CB_HIDE (see page 119)	Check box must be removed from screen

Module

Checkbox (see page 115)

Description

List of Checkbox (see page 115) bit states.

8.1.10.1.1 CB_CHECKED Macro

File

CheckBox.h (see page 732)

C

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

Description

Checked state

8.1.10.1.2 CB_DISABLED Macro

File

CheckBox.h (see page 732)

C

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

Description

Disabled state

8.1.10.1.3 CB_DRAW Macro

File

CheckBox.h (see page 732)

C

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

Description

Whole check box must be redrawn

8.1.10.1.4 CB_DRAW_CHECK Macro

File

CheckBox.h (see page 732)

C

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

Description

Check box mark should be redrawn

8.1.10.1.5 CB_DRAW_FOCUS Macro

File

CheckBox.h (see page 732)

C

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

Description

Focus must be redrawn

8.1.10.1.6 CB_FOCUSED Macro

File

CheckBox.h (see page 732)

C

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

Description

Focus state

8.1.10.1.7 CB_HIDE Macro

File

CheckBox.h (see page 732)

C

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

Description

Check box must be removed from screen

8.1.10.2 CbCreate Function

File

CheckBox.h (see page 732)

C

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

Module

Checkbox (see page 115)

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

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

Overview

This function creates a CHECKBOX (see page 124) 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 (see page 124) *CbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.10.3 CbDraw Function

File

CheckBox.h (see page 732)

C

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

Module

Checkbox (see page 115)

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 (see page 119)() 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)

8.1.10.4 CbGetText Macro

File

CheckBox.h (see page 732)

C

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

Module

Checkbox (see page 115)

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)

8.1.10.5 CbSetText Function

File

CheckBox.h (see page 732)

C

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

Module

Checkbox (see page 115)

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 (see page 124) *pCb, XCHAR (see page 345) *pText)

8.1.10.6 CbMsgDefault Function

File

CheckBox.h (see page 732)

C

```
void CbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Checkbox (see page 115)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL (see page 48) 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 (see page 117)	Check Box will be redrawn in checked state.
CB_MSG_UNCHECKED	Touch Screen, Keyboard	Clear CB_CHECKED (see page 117)	Check Box will be redrawn in un-checked state.

Syntax

CbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.10.7 CbTranslateMsg Function

File

CheckBox.h (see page 732)

C

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

Module

Checkbox (see page 115)

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 (see page 48) 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 (see page 75)() 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 (see page 321) or SCAN_SPACE_PRESSED (see page 325) 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 (see page 321) or SCAN_SPACE_PRESSED (see page 325) 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 (see page 316) *pMsg)

8.1.10.8 CHECKBOX Structure

File

CheckBox.h (see page 732)

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 (see page 52)).
SHORT textHeight;	Pre-computed text height
XCHAR * pText;	Pointer to text

Module

Checkbox (see page 115)

Overview

The structure contains check box data

8.1.11 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 (see page 127)	This function creates a ROUNDDIAL (see page 134) 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 (see page 128)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the center (x,y) postion 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 (see page 132)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	RdiaTranslateMsg (see page 132)	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
RdiaIncVal (see page 129)	Used to directly increment the value. The delta change used is the resolution setting (res).
RdiaDecVal (see page 130)	Used to directly decrement the value. The delta change used is the resolution setting (res).
RdiaGetVal (see page 130)	Returns the current dial value. Value is always in the 0-max range inclusive.
RdiaSetVal (see page 131)	Sets the value to the given new value. Value set must be in 0-max range inclusive.

Structures

Name	Description
ROUNDDIAL (see page 134)	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.

8.1.11.1 Dial States

Macros

Name	Description
RDIA_DISABLED (see page 126)	Bit for disabled state.
RDIA_DRAW (see page 126)	Bit to indicate object must be redrawn.
RDIA_HIDE (see page 126)	Bit to indicate object must be removed from screen.
RDIA_ROT_CCW (see page 127)	Bit for rotate counter clockwise state.
RDIA_ROT_CW (see page 127)	Bit for rotate clockwise state.

Module

Dial (see page 124)

Description

List of Dial (see page 124) bit states.

8.1.11.1.1 RDIA_DISABLED Macro

File

RoundDial.h (see page 852)

C

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

Description

Bit for disabled state.

8.1.11.1.2 RDIA_DRAW Macro

File

RoundDial.h (see page 852)

C

```
#define RDIA_DRAW 0x4000  // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

8.1.11.1.3 RDIA_HIDE Macro

File

RoundDial.h (see page 852)

C

```
#define RDIA_HIDE 0x8000  // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

8.1.11.1.4 RDIA_ROT_CCW Macro

File

RoundDial.h (see page 852)

C

```
#define RDIA_ROT_CCW 0x0008  // Bit for rotate counter clockwise state.
```

Description

Bit for rotate counter clockwise state.

8.1.11.1.5 RDIA_ROT_CW Macro

File

RoundDial.h (see page 852)

C

```
#define RDIA_ROT_CW 0x0004  // Bit for rotate clockwise state.
```

Description

Bit for rotate clockwise state.

8.1.11.2 RdiaCreate Function

File

RoundDial.h (see page 852)

C

```
ROUNDDIAL * RdiaCreate(
    WORD ID,
    SHORT x,
    SHORT y,
    SHORT radius,
    WORD state,
    SHORT res,
    SHORT value,
    SHORT max,
    GOL_SCHEME * pScheme
);
```

Module

Dial (see page 124)

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 (see page 134) 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 (see page 134) *RdiaCreate(WORD ID, SHORT x, SHORT y, SHORT radius, WORD state, SHORT res, SHORT value, SHORT max, GOL_SCHEME (see page 331) *pScheme);

8.1.11.3 RdiaDraw Function

File

RoundDial.h (see page 852)

C

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

Module

Dial (see page 124)

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) postion 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)

8.1.11.4 RdiaIncVal Macro

File

RoundDial.h (see page 852)

C

```
#define RdiaIncVal(pDia) RdiaSetVal(pDia, (pDia->val + pDia->res))
```

Module

Dial (see page 124)

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)

8.1.11.5 RdiaDecVal Macro

File

RoundDial.h (see page 852)

C

```
#define RdiaDecVal(pDia) RdiaSetVal(pDia, (pDia->pos - pDia->res))
```

Module

Dial (see page 124)

Input Parameters

Input Parameters	Description
pDia	Pointer to the object.

Side Effects

none

Returns

none

Preconditions

none

Example

Refer to RdiaIncVal (see page 129)() example.

Overview

Used to directly decrement the value. The delta change used is the resolution setting (res).

Syntax

RdiaDecVal(pDia)

8.1.11.6 RdiaGetVal Macro

File

RoundDial.h (see page 852)

C

```
#define RdiaGetVal(pDia) (pDia)->value
```

Module

Dial (see page 124)

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)

8.1.11.7 RdiaSetVal Macro

File

RoundDial.h (see page 852)

C

```
#define RdiaSetVal(pDia, newVal) (pDia)->value = newVal
```

Module

Dial (see page 124)

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)

8.1.11.8 RdiaMsgDefault Function

File

RoundDial.h (see page 852)

C

```
void RdiaMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Dial (see page 124)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL (see page 48) message
pDia	The pointer to the object whose state will be modified

Side Effects

none

Returns

none

Preconditions

none

Example

See RdiaTranslateMsg (see page 132)() 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 (see page 127), Set RDIA_DRAW (see page 126)	Dial (see page 124) will be redrawn with clockwise update.
RD_MSG_CTR_CLOCKWISE	Touch Screen	Set RDIA_ROT_CCW (see page 127), Set RDIA_DRAW (see page 126)	Dial (see page 124) will be redrawn with counter clockwise update.

Syntax

RdiaMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.11.9 RdiaTranslateMsg Function

File

RoundDial.h (see page 852)

C

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

Module

Dial (see page 124)

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 (see page 48) message:

- RD_MSG_CLOCKWISE – Dial (see page 124) is moved in a clockwise direction.
- RD_MSG_CTR_CLOCKWISE – Dial (see page 124) is moved in a counter clockwise direction.
- OBJ_MSG_INVALID – Dial (see page 124) 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_ROUNDIAL:
                    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 (see page 124) 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 (see page 124) 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 (see page 316) *pMsg)

8.1.11.10 ROUNDDIAL Structure

File

RoundDial.h (see page 852)

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 (see page 52)).
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

Dial (see page 124)

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.

8.1.11.11 Files

8.1.12 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 (see page 138)	This function creates a DIGITALMETER (see page 143) 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 (see page 139)	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 (see page 140)	This function sets the value that will be used for the object.
	DmTranslateMsg (see page 142)	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 (see page 140)	This macro returns the current value used for the object.
DmDecVal (see page 141)	This macro is used to directly decrement the value.
DmIncVal (see page 141)	This macro is used to directly increment the value.

Structures

Name	Description
DIGITALMETER (see page 143)	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.

8.1.12.1 Digital Meter States

Macros

Name	Description
DM_DISABLED (see page 136)	Bit for disabled state.
DM_DRAW (see page 136)	Bit to indicate object must be redrawn.
DM_HIDE (see page 137)	Bit to remove object from screen.
DM_CENTER_ALIGN (see page 137)	Bit to indicate value is center aligned.
DM_RIGHT_ALIGN (see page 137)	Bit to indicate value is left aligned.
DM_FRAME (see page 137)	Bit to indicate frame is displayed.
DM_UPDATE (see page 137)	Bit to indicate that only text must be redrawn.

Module

Digital Meter (see page 135)

Description

List of Digital Meter (see page 196) bit states.

8.1.12.1.1 DM_DISABLED Macro

File

DigitalMeter.h (see page 742)

C

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

Description

Bit for disabled state.

8.1.12.1.2 DM_DRAW Macro

File

DigitalMeter.h (see page 742)

C

```
#define DM_DRAW 0x4000  // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

8.1.12.1.3 DM_HIDE Macro

File

DigitalMeter.h (see page 742)

C

```
#define DM_HIDE 0x8000  // Bit to remove object from screen.
```

Description

Bit to remove object from screen.

8.1.12.1.4 DM_CENTER_ALIGN Macro

File

DigitalMeter.h (see page 742)

C

```
#define DM_CENTER_ALIGN 0x0008  // Bit to indicate value is center aligned.
```

Description

Bit to indicate value is center aligned.

8.1.12.1.5 DM_RIGHT_ALIGN Macro

File

DigitalMeter.h (see page 742)

C

```
#define DM_RIGHT_ALIGN 0x0004  // Bit to indicate value is left aligned.
```

Description

Bit to indicate value is left aligned.

8.1.12.1.6 DM_FRAME Macro

File

DigitalMeter.h (see page 742)

C

```
#define DM_FRAME 0x0010  // Bit to indicate frame is displayed.
```

Description

Bit to indicate frame is displayed.

8.1.12.1.7 DM_UPDATE Macro

File

DigitalMeter.h (see page 742)

C

```
#define DM_UPDATE 0x2000  // Bit to indicate that only text must be redrawn.
```

Description

Bit to indicate that only text must be redrawn.

8.1.12.2 DmCreate Function

File

DigitalMeter.h (see page 742)

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 (see page 135)

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 (see page 143) 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 (see page 143) *DmCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state , DWORD Value, BYTE NoOfDigits, BYTE DotPos, GOL_SCHEME (see page 331) *pScheme)

8.1.12.3 DmDraw Function

File

DigitalMeter.h (see page 742)

C

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

Module

Digital Meter (see page 135)

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 (see page 138)() 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)

8.1.12.4 DmGetValue Macro

File

DigitalMeter.h (see page 742)

C

```
#define DmGetValue(pDm) pDm->Cvalue
```

Module

Digital Meter (see page 135)

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)

8.1.12.5 DmSetValue Function

File

DigitalMeter.h (see page 742)

C

```
void DmSetValue(
    DIGITALMETER * pDm,
    DWORD Value
);
```

Module

Digital Meter (see page 135)

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 (see page 135).

Side Effects

none

Returns

none

Preconditions

none

Overview

This function sets the value that will be used for the object.

Syntax

DmSetValue(DIGITALMETER (see page 143) *pDm, DWORD Value)

8.1.12.6 DmDecVal Macro

File

DigitalMeter.h (see page 742)

C

```
#define DmDecVal(pDm, deltaValue) DmSetValue(pDm, (pDm->Cvalue - deltaValue))
```

Module

Digital Meter (see page 135)

Input Parameters

Input Parameters	Description
pDm	Pointer to the object.
deltaValue	Number to be subtracted to the current Digital Meter (see page 135) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly decrement the value.

Syntax

DmDecVal(pDm, deltaValue)

8.1.12.7 DmIncVal Macro

File

DigitalMeter.h (see page 742)

C

```
#define DmIncVal(pDm, deltaValue) DmSetValue(pDm, (pDm->Cvalue + deltaValue))
```

Module

Digital Meter (see page 135)

Input Parameters

Input Parameters	Description
pDm	Pointer to the object.

deltaValue	Number to be added to the current Digital Meter (see page 135) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly increment the value.

Syntax

DmIncVal(pDm, deltaValue)

8.1.12.8 DmTranslateMsg Function

File

DigitalMeter.h (see page 742)

C

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

Module

Digital Meter (see page 135)

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 (see page 48) message:

- DM_MSG_SELECTED – Digital Meter (see page 135) is selected
- OBJ_MSG_INVALID – Digital Meter (see page 135) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 135).
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

WORD DmTranslateMsg(void *pObj, GOL_MSG (see page 316) *pMsg)

8.1.12.9 DIGITALMETER Structure

File

DigitalMeter.h (see page 742)

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 (see page 52)).
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 (see page 135)

Description

Structure: DIGITALMETER

Overview

Defines the parameters required for a Digital Meter (see page 135) Object.

8.1.13 Edit Box

Edit Box is is an Object that emulates a cell or a text area that can be edited dynamically.

Functions

	Name	Description
	EbCreate (see page 147)	This function creates a EDITBOX (see page 153) 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 (see page 148)	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 (see page 149)	This function sets the text to be used for the object.
	EbAddChar (see page 150)	This function inserts a character at the end of the text used by the object.
	EbDeleteChar (see page 150)	This function removes a character at the end of the text used by the object.
	EbMsgDefault (see page 151)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	EbTranslateMsg (see page 152)	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 (see page 149)	This macro returns the address of the current text string used for the object.

Structures

Name	Description
EDITBOX (see page 153)	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

8.1.13.1 Edit Box States

Macros

Name	Description
EB_CENTER_ALIGN (see page 145)	Bit to indicate text is center aligned.
EB_DISABLED (see page 145)	Bit for disabled state.
EB_DRAW (see page 145)	Bit to indicate whole edit box must be redrawn.
EB_HIDE (see page 146)	Bit to remove object from screen.
EB_FOCUSED (see page 146)	Bit for focused state.
EB_RIGHT_ALIGN (see page 146)	Bit to indicate text is left aligned.
EB_DRAW_CARET (see page 146)	Bit to indicate the cursor caret will be redrawn if the EB_CARET (see page 146) state bit is set.
EB_CARET (see page 146)	Bit to indicate the cursor will always be shown. This bit can be set when focused and reset when object is disabled. Application can also set and clear the bit through SetState (see page 289)() or ClrState (see page 289)().

Module

Edit Box (see page 143)

Description

List of Edit Box bit states.

8.1.13.1.1 EB_CENTER_ALIGN Macro

File

EditBox.h (see page 753)

C

```
#define EB_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned.
```

Description

Bit to indicate text is center aligned.

8.1.13.1.2 EB_DISABLED Macro

File

EditBox.h (see page 753)

C

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

Description

Bit for disabled state.

8.1.13.1.3 EB_DRAW Macro

File

EditBox.h (see page 753)

C

```
#define EB_DRAW 0x4000  // Bit to indicate whole edit box must be redrawn.
```

Description

Bit to indicate whole edit box must be redrawn.

8.1.13.1.4 EB_HIDE Macro

File

EditBox.h (see page 753)

C

```
#define EB_HIDE 0x8000  // Bit to remove object from screen.
```

Description

Bit to remove object from screen.

8.1.13.1.5 EB_FOCUSED Macro

File

EditBox.h (see page 753)

C

```
#define EB_FOCUSED 0x0001  // Bit for focused state.
```

Description

Bit for focused state.

8.1.13.1.6 EB_RIGHT_ALIGN Macro

File

EditBox.h (see page 753)

C

```
#define EB_RIGHT_ALIGN 0x0004  // Bit to indicate text is left aligned.
```

Description

Bit to indicate text is left aligned.

8.1.13.1.7 EB_DRAW_CARET Macro

File

EditBox.h (see page 753)

C

```
#define EB_DRAW_CARET 0x2000  // Bit to indicate the cursor caret will be redrawn if the
EB_CARET state bit is set.
```

Description

Bit to indicate the cursor caret will be redrawn if the EB_CARET (see page 146) state bit is set.

8.1.13.1.8 EB_CARET Macro

File

EditBox.h (see page 753)

C

```
#define EB_CARET 0x0010  // Bit to indicate the cursor will always be shown.
```

Description

Bit to indicate the cursor will always be shown. This bit can be set when focused and reset when object is disabled. Application can also set and clear the bit through SetState (see page 289)() or ClrState (see page 289)().

8.1.13.2 EbCreate Function

File

EditBox.h (see page 753)

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 (see page 143)

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 (see page 153) 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 (see page 153) *EbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state , XCHAR (see page 345) *pText, WORD charMax, GOL_SCHEME (see page 331) *pScheme)

8.1.13.3 EbDraw Function

File

EditBox.h (see page 753)

C

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

Module

Edit Box (see page 143)

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)

8.1.13.4 EbGetText Macro

File

EditBox.h (see page 753)

C

```
#define EbGetText(pEb) (pEb->pBuffer)
```

Module

Edit Box (see page 143)

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)

8.1.13.5 EbSetText Function

File

EditBox.h (see page 753)

C

```
void EbSetText(
    EDITBOX * pEb,
    XCHAR * pText
);
```

Module

Edit Box (see page 143)

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 (see page 153) *pEb, XCHAR (see page 345) *pText)

8.1.13.6 EbAddChar Function

File

EditBox.h (see page 753)

C

```
void EbAddChar(
    EDITBOX * pEb,
    XCHAR ch
);
```

Module

Edit Box (see page 143)

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 (see page 153)* pEb, XCHAR (see page 345) ch)

8.1.13.7 EbDeleteChar Function

File

EditBox.h (see page 753)

C

```
void EbDeleteChar(
    EDITBOX * pEb
);
```

Module

Edit Box (see page 143)

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 (see page 153)* pEb)

8.1.13.8 EbMsgDefault Function

File

EditBox.h (see page 753)

C

```
void EbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Edit Box (see page 143)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL (see page 48) 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 (see page 74)() 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 (see page 145)	New character is added and Edit Box will be redrawn.
EB_MSG_DEL	Keyboard	Set EB_DRAW (see page 145)	Last character is removed and Edit Box will be redrawn.

Syntax

void EbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316) *pMsg)

8.1.13.9 EbTranslateMsg Function

File

EditBox.h (see page 753)

C

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

Module

Edit Box (see page 143)

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 (see page 48) 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 (see page 75)() 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 (see page 316) *pMsg)

8.1.13.10 EDITBOX Structure

File

EditBox.h (see page 753)

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 (see page 52)).
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 (see page 143)

Overview

Defines the parameters required for a Edit Box Object.

8.1.14 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 (see page 159)	This function creates a GRID (see page 169) 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 (see page 160)	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 (see page 161)	This function clears the state of the cell (or Grid Item) specified by the column and row.
	GridFreeItems (see page 163)	This function removes all grid items for the given Grid and frees the memory used.

	GridGetCell (see page 163)	This function removes all grid items for the given Grid and frees the memory used.
	GridSetCell (see page 164)	This function sets the Grid Item state and data.
	GridSetCellState (see page 165)	This function sets the state of the Grid Item or cell.
	GridSetFocus (see page 166)	This function sets the focus of the specified Grid Item or cell.
	GridMsgDefault (see page 166)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	GridTranslateMsg (see page 167)	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 (see page 161)	This macro returns the x position of the focused cell.
GridGetFocusY (see page 162)	This macro returns the y position of the focused cell.
GRID_OUT_OF_BOUNDS (see page 162)	Status of an out of bounds cell GridSetCell (see page 164)() operation.
GRID_SUCCESS (see page 163)	Status of a successful GridSetCell (see page 164)() operation.

Structures

Name	Description
GRID (see page 169)	Defines the parameters required for a grid Object. Clipping is not supported in grid object.
GRIDITEM (see page 169)	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 (see page 167)() and GridMsgDefault (see page 166)() for details.

The Grid lines are drawn using the EmbossLitColor, the string drawn using the TextColor0 and the background is drawn using the CommonBkColor.

8.1.14.1 Grid States

Macros

Name	Description
GRID_FOCUSED (see page 155)	Bit for focused state
GRID_DISABLED (see page 155)	Bit for disabled state
GRID_SHOW_LINES (see page 155)	Display grid lines
GRID_SHOW_FOCUS (see page 155)	Highlight the focused cell.

GRID_SHOW_BORDER_ONLY (see page 156)	Draw only the outside border of the grid.
GRID_SHOW_SEPARATORS_ONLY (see page 156)	Draw only the lines between cells (like Tic-tac-toe)
GRID_DRAW_ITEMS (see page 156)	Bit to indicate that at least one item must be redrawn, but not the entire grid.
GRID_DRAW_ALL (see page 156)	Bit to indicate whole edit box must be redrawn
GRID_HIDE (see page 156)	Bit to remove object from screen

Module

Grid (see page 153)

Description

List of Grid (see page 153) bit states.

8.1.14.1.1 GRID_FOCUSED Macro

File

Grid.h (see page 766)

C

```
#define GRID_FOCUSED 0x0001  // Bit for focused state
```

Description

Bit for focused state

8.1.14.1.2 GRID_DISABLED Macro

File

Grid.h (see page 766)

C

```
#define GRID_DISABLED 0x0002  // Bit for disabled state
```

Description

Bit for disabled state

8.1.14.1.3 GRID_SHOW_LINES Macro

File

Grid.h (see page 766)

C

```
#define GRID_SHOW_LINES 0x0004  // Display grid lines
```

Description

Display grid lines

8.1.14.1.4 GRID_SHOW_FOCUS Macro

File

Grid.h (see page 766)

C

```
#define GRID_SHOW_FOCUS 0x0008  // Highlight the focused cell.
```

Description

Highlight the focused cell.

8.1.14.1.5 GRID_SHOW_BORDER_ONLY Macro

File

Grid.h (see page 766)

C

```
#define GRID_SHOW_BORDER_ONLY 0x0010  // Draw only the outside border of the grid.
```

Description

Draw only the outside border of the grid.

8.1.14.1.6 GRID_SHOW_SEPARATORS_ONLY Macro

File

Grid.h (see page 766)

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)

8.1.14.1.7 GRID_DRAW_ITEMS Macro

File

Grid.h (see page 766)

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.

8.1.14.1.8 GRID_DRAW_ALL Macro

File

Grid.h (see page 766)

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

8.1.14.1.9 GRID_HIDE Macro

File

Grid.h (see page 766)

C

```
#define GRID_HIDE 0x8000  // Bit to remove object from screen
```

Description

Bit to remove object from screen

8.1.14.2 Grid Item States

Macros

Name	Description
GRIDITEM_SELECTED (see page 157)	The cell is selected.
GRIDITEM_IS_TEXT (see page 157)	The grid item is a test string.
GRIDITEM_IS_BITMAP (see page 158)	The grid item is a bitmap.
GRIDITEM_TEXTBOTTOM (see page 158)	Bit to indicate text is top aligned.
GRIDITEM_TEXTLEFT (see page 158)	Text in the cell is left aligned.
GRIDITEM_TEXTRIGHT (see page 158)	Text in the cell is right aligned.
GRIDITEM_TEXTTOP (see page 158)	Bit to indicate text is bottom aligned.
GRIDITEM_DRAW (see page 159)	Draw this cell

Module

Grid (see page 153)

Description

List of Grid (see page 153) Items bit states.

8.1.14.2.1 GRIDITEM_SELECTED Macro

File

Grid.h (see page 766)

C

```
#define GRIDITEM_SELECTED 0x0001  // The cell is selected.
```

Description

The cell is selected.

8.1.14.2.2 GRIDITEM_IS_TEXT Macro

File

Grid.h (see page 766)

C

```
#define GRIDITEM_IS_TEXT 0x0000  // The grid item is a test string.
```

Description

The grid item is a test string.

8.1.14.2.3 GRIDITEM_IS_BITMAP Macro

File

Grid.h (see page 766)

C

```
#define GRIDITEM_IS_BITMAP 0x0008  // The grid item is a bitmap.
```

Description

The grid item is a bitmap.

8.1.14.2.4 GRIDITEM_TEXTBOTTOM Macro

File

Grid.h (see page 766)

C

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

Description

Bit to indicate text is top aligned.

8.1.14.2.5 GRIDITEM_TEXTLEFT Macro

File

Grid.h (see page 766)

C

```
#define GRIDITEM_TEXTLEFT 0x0020  // Text in the cell is left aligned.
```

Description

Text in the cell is left aligned.

8.1.14.2.6 GRIDITEM_TEXTRIGHT Macro

File

Grid.h (see page 766)

C

```
#define GRIDITEM_TEXTRIGHT 0x0010  // Text in the cell is right aligned.
```

Description

Text in the cell is right aligned.

8.1.14.2.7 GRIDITEM_TEXTTOP Macro

File

Grid.h (see page 766)

C

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

Description

Bit to indicate text is bottom aligned.

8.1.14.2.8 GRIDITEM_DRAW Macro

File

Grid.h (see page 766)

C

```
#define GRIDITEM_DRAW 0x0100  // Draw this cell
```

Description

Draw this cell

8.1.14.3 GridCreate Function

File

Grid.h (see page 766)

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 (see page 153)

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 (see page 169) 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 (see page 169) *GridCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT numColumns, SHORT numRows, SHORT cellWidth, SHORT cellHeight, GOL_SCHEME (see page 331) *pScheme)

8.1.14.4 GridDraw Function

File

Grid.h (see page 766)

C

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

Module

Grid (see page 153)

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)

8.1.14.5 GridClearCellState Function

File

Grid.h (see page 766)

C

```
void GridClearCellState(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD state
);
```

Module

Grid (see page 153)

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
atate	specifies the state to be cleared. See Grid (see page 153) Item State.

Side Effects

none

Returns

none.

Preconditions

none

Overview

This function clears the state of the cell (or Grid (see page 153) Item) specified by the column and row.

Syntax

GridClearCellState(GRID (see page 169) *pGrid, SHORT column, SHORT row, WORD state)

8.1.14.6 GridGetFocusX Macro

File

Grid.h (see page 766)

C

```
#define GridGetFocusX(pGrid) pGrid->focusX
```

Module

Grid (see page 153)

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)

8.1.14.7 GridGetFocusY Macro

File

Grid.h (see page 766)

C

```
#define GridGetFocusY(pGrid) pGrid->focusY
```

Module

Grid (see page 153)

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)

8.1.14.8 GRID_OUT_OF_BOUNDS Macro

File

Grid.h (see page 766)

C

```
#define GRID_OUT_OF_BOUNDS 0x0001  // Status of an out of bounds cell GridSetCell()
operation.
```

Module

Grid (see page 153)

Description

Status of an out of bounds cell GridSetCell (see page 164)() operation.

8.1.14.9 GRID_SUCCESS Macro

File

Grid.h (see page 766)

C

```
#define GRID_SUCCESS 0x0000    // Status of a successful GridSetCell() operation.
```

Module

Grid (see page 153)

Description

Status of a successful GridSetCell (see page 164)() operation.

8.1.14.10 GridFreeItems Function

File

Grid.h (see page 766)

C

```
void GridFreeItems(
    void * pObj
);
```

Module

Grid (see page 153)

Input Parameters

Input Parameters	Description
pGrid	The pointer to the Grid (see page 153) 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 (see page 153) and frees the memory used.

Syntax

GridFreeItems(void *pObj)

8.1.14.11 GridGetCell Function

File

Grid.h (see page 766)

C

```
void * GridGetCell(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD * cellType
);
```

Module

Grid (see page 153)

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (see page 153) 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 (see page 157) or GRIDITEM_IS_BITMAP (see page 158)).

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 (see page 153) and frees the memory used.

Syntax

*GridGetCell(GRID (see page 169) *pGrid, SHORT column, SHORT row, WORD *cellType)

8.1.14.12 GridSetCell Function

File

Grid.h (see page 766)

C

```
WORD GridSetCell(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD state,
    void * data
);
```

Module

Grid (see page 153)

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (see page 153) 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 (see page 153) Item specified.

void * data	pointer to the data used for the Grid (see page 153) Item.

Side Effects

none

Returns

Returns the status of the operation

- GRID_SUCCESS (see page 163) - if the set succeeded
- GRID_OUT_OF_BOUNDS (see page 162) - 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 (see page 153) Item state and data.

Syntax

WORD GridSetCell(GRID (see page 169) *pGrid, SHORT column, SHORT row, WORD state, void *data)

8.1.14.13 GridSetCellState Function

File

Grid.h (see page 766)

C

```
void GridSetCellState(
    GRID * pGrid,
    SHORT column,
    SHORT row,
    WORD state
);
```

Module

Grid (see page 153)

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (see page 153) 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 (see page 153) 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 (see page 153) Item or cell.

Syntax

GridSetCellState(GRID (see page 169) *pGrid, SHORT column, SHORT row, WORD state)

8.1.14.14 GridSetFocus Function

File

Grid.h (see page 766)

C

```
void GridSetFocus(
    GRID * pGrid,
    SHORT column,
    SHORT row
);
```

Module

Grid (see page 153)

Input Parameters

Input Parameters	Description
GRID * pGrid	The pointer to the Grid (see page 153) 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 (see page 153) Item or cell.

Syntax

GridSetFocus(GRID (see page 169) *pGrid, SHORT column, SHORT row)

8.1.14.15 GridMsgDefault Function

File

Grid.h (see page 766)

C

```
void GridMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Grid (see page 153)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL (see page 48) 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 (see page 153) will have no state change because of this event.
GRID_MSG_ITEM_SELECTED	Keyboard	Set GRIDITEM_SELECTED (see page 157),	Grid (see page 153) Item selected will be redrawn.
		GRID_DRAW_ITEMS (see page 156)	
GRID_MSG_UP	Keyboard	Set GRIDITEM_DRAW (see page 159),	Grid (see page 153) Item above the currently focused item will be redrawn.
		GRID_DRAW_ITEMS (see page 156)	
GRID_MSG_DOWN	Keyboard	Set GRIDITEM_DRAW (see page 159),	Grid (see page 153) Item below the currently focused item will be redrawn.
		GRID_DRAW_ITEMS (see page 156)	
GRID_MSG_LEFT	Keyboard	Set GRIDITEM_DRAW (see page 159),	Grid (see page 153) Item to the left of the currently focused item will be redrawn.
		GRID_DRAW_ITEMS (see page 156)	
GRID_MSG_RIGHT	Keyboard	Set GRIDITEM_DRAW (see page 159),	Grid (see page 153) Item to the right of the currently focused item will be redrawn.
		GRID_DRAW_ITEMS (see page 156)	

Syntax

GridMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316) *pMsg)

8.1.14.16 GridTranslateMsg Function

File

Grid.h (see page 766)

C

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

Module

Grid (see page 153)

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 (see page 48) message:

- GRID_MSG_TOUCHED - when the grid object is touched.
- GRID_MSG_ITEM_SELECTED – when key scan SCAN_SPACE_PRESSED (see page 325) or SCAN_CR_PRESSED (see page 321) are detected.
- GRID_MSG_UP – when key scan SCAN_UP_PRESSED (see page 326) is detected.
- GRID_MSG_DOWN – when key scan SCAN_DOWN_PRESSED (see page 322) is detected.
- GRID_MSG_LEFT – when key scan SCAN_LEFT_PRESSED (see page 323) is detected.
- GRID_MSG_RIGHT – when key scan SCAN_RIGHT_PRESSED (see page 325) is detected.
- OBJ_MSG_INVALID – Button (see page 65) 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 (see page 325) or SCAN_CR_PRESSED (see page 321).
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 (see page 326).
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 (see page 322).
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 (see page 323).
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 (see page 325).
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

GridTranslateMsg(void *pObj, GOL_MSG (see page 316) *pMsg)

8.1.14.17 GRID Structure

File

Grid.h (see page 766)

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 (see page 52)).
SHORT numColumns;	Number of columns drawn for the Grid (see page 153).
SHORT numRows;	Number of rows drawn for the Grid (see page 153).
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 (see page 153)

Overview

Defines the parameters required for a grid Object. Clipping is not supported in grid object.

8.1.14.18 GRIDITEM Structure

File

Grid.h (see page 766)

C

```
typedef struct {
  void * data;
  WORD status;
} GRIDITEM;
```

Members

Members	Description
void * data;	Indicates if the Grid (see page 153) Item is type GRIDITEM_IS_TEXT (see page 157) or GRIDITEM_IS_BITMAP (see page 158)
WORD status;	indicates the status of the Grid (see page 153) Item

Module

Grid (see page 153)

Overview

Defines the grid item.

8.1.15 Group Box

Group Box is an Object that can be used to group Objects together in the screen.

Functions

	Name	Description
	GbCreate (see page 172)	This function creates a GROUPBOX (see page 176) 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 (see page 174)	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 (see page 175)	This function sets the text used by passing the pointer to the static string.
	GbTranslateMsg (see page 176)	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 (see page 174)	This macro returns the location of the text used.

Structures

Name	Description
GROUPBOX (see page 176)	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.

8.1.15.1 Group Box States

Macros

Name	Description
GB_CENTER_ALIGN (see page 171)	Bit to indicate text is center aligned
GB_DISABLED (see page 172)	Bit for disabled state
GB_DRAW (see page 172)	Bit to indicate group box must be redrawn
GB_HIDE (see page 172)	Bit to remove object from screen
GB_RIGHT_ALIGN (see page 172)	Bit to indicate text is right aligned

Module

Group Box (see page 170)

Description

List of Group Box bit states.

8.1.15.1.1 GB_CENTER_ALIGN Macro

File

GroupBox.h (see page 778)

C

```
#define GB_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned
```

Description

Bit to indicate text is center aligned

8.1.15.1.2 GB_DISABLED Macro

File

GroupBox.h (see page 778)

C

```
#define GB_DISABLED 0x0002  // Bit for disabled state
```

Description

Bit for disabled state

8.1.15.1.3 GB_DRAW Macro

File

GroupBox.h (see page 778)

C

```
#define GB_DRAW 0x4000          // Bit to indicate group box must be redrawn
```

Description

Bit to indicate group box must be redrawn

8.1.15.1.4 GB_HIDE Macro

File

GroupBox.h (see page 778)

C

```
#define GB_HIDE 0x8000          // Bit to remove object from screen
```

Description

Bit to remove object from screen

8.1.15.1.5 GB_RIGHT_ALIGN Macro

File

GroupBox.h (see page 778)

C

```
#define GB_RIGHT_ALIGN 0x0004  // Bit to indicate text is right aligned
```

Description

Bit to indicate text is right aligned

8.1.15.2 GbCreate Function

File

GroupBox.h (see page 778)

C

```
GROUPBOX * GbCreate(
    WORD ID,
    SHORT left,
    SHORT top,
```

```
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    GOL_SCHEME * pScheme
);
```

Module

Group Box (see page 170)

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 (see page 176) 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 (see page 176) *GbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.15.3 GbDraw Function

File

GroupBox.h (see page 778)

C

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

Module

Group Box (see page 170)

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 (see page 172)() 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)

8.1.15.4 GbGetText Macro

File

GroupBox.h (see page 778)

C

```
#define GbGetText(pB) pGb->pText
```

Module

Group Box (see page 170)

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)

8.1.15.5 GbSetText Function

File

GroupBox.h (see page 778)

C

```
void GbSetText(
    GROUPBOX * pGb,
    XCHAR * pText
);
```

Module

Group Box (see page 170)

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

none

Returns

none

Preconditions

none

Overview

This function sets the text used by passing the pointer to the static string.

Syntax

GbSetText(GROUPBOX (see page 176) *pGb, XCHAR (see page 345) *pText)

8.1.15.6 GbTranslateMsg Function

File

GroupBox.h (see page 778)

C

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

Module

Group Box (see page 170)

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 (see page 48) message:

- GB_MSG_SELECTED – Group Box is selected
- OBJ_MSG_INVALID – Group Box is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 316) *pMsg)

8.1.15.7 GROUPBOX Structure

File

GroupBox.h (see page 778)

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 (see page 52)).
SHORT textWidth;	Pre-computed text width.
SHORT textHeight;	Pre-computed text height.
XCHAR * pText;	Text string used.

Module

Group Box (see page 170)

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.

8.1.16 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 (see page 181)	This function creates a LISTBOX (see page 195) 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 (see page 183)	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 (see page 184)	This function allocates memory for the LISTITEM (see page 196) and adds it to the list box. The newly created LISTITEM (see page 196) will store the location of pText, pBitmap and other parameters describing the added item.
	LbDelItem (see page 186)	This function removes an item from the list box and frees the memory used.
	LbChangeSel (see page 186)	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 (see page 188)	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 (see page 188)	This function returns the index of the focused item in the list box.

	LbSetFocusedItem (see page 189)	This function sets the focus for the item with the given index.
	LbDelItemsList (see page 193)	This function removes all items from the list box and frees the memory used.
	LbMsgDefault (see page 193)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	LbTranslateMsg (see page 194)	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 (see page 184)	This function returns the pointer to the current item list used in the list box.
LbSetSel (see page 187)	This macro sets the selection status of an item to selected.
LbGetCount (see page 189)	This macro returns the number of items in the list box.
LbGetVisibleCount (see page 190)	This macro returns the number of items visible in the list box window.
LbClrtSel (see page 191)	This macro clears the selection of an item.
LbSetBitmap (see page 191)	This macro sets the bitmap used in the item.
LbGetBitmap (see page 192)	This macro returns the location of the currently used bitmap for the item.

Structures

Name	Description
LISTBOX (see page 195)	Defines the parameters required for a list box Object.
LISTITEM (see page 196)	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.

8.1.16.1 List Box States

Macros

Name	Description
LB_RIGHT_ALIGN (see page 179)	Bit to indicate text is left aligned
LB_SINGLE_SEL (see page 179)	Bit to indicate the only item can be selected
LB_CENTER_ALIGN (see page 179)	Bit to indicate text is center aligned
LB_DISABLED (see page 180)	Bit for disabled state
LB_DRAW (see page 180)	Bit to indicate whole edit box must be redrawn
LB_DRAW_FOCUS (see page 180)	Bit to indicate whole edit box must be redrawn
LB_DRAW_ITEMS (see page 180)	Bit to indicate whole edit box must be redrawn
LB_FOCUSED (see page 180)	Bit for focused state
LB_HIDE (see page 181)	Bit to remove object from screen

Module

List Box (see page 177)

Description

List of List Box bit states.

8.1.16.1.1 LB_RIGHT_ALIGN Macro

File

ListBox.h (see page 794)

C

```
#define LB_RIGHT_ALIGN 0x0004  // Bit to indicate text is left aligned
```

Description

Bit to indicate text is left aligned

8.1.16.1.2 LB_SINGLE_SEL Macro

File

ListBox.h (see page 794)

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

8.1.16.1.3 LB_CENTER_ALIGN Macro

File

ListBox.h (see page 794)

C

```
#define LB_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned
```

Description

Bit to indicate text is center aligned

8.1.16.1.4 LB_DISABLED Macro

File

ListBox.h (see page 794)

C

```
#define LB_DISABLED 0x0002  // Bit for disabled state
```

Description

Bit for disabled state

8.1.16.1.5 LB_DRAW Macro

File

ListBox.h (see page 794)

C

```
#define LB_DRAW 0x4000  // Bit to indicate whole edit box must be redrawn
```

Description

Bit to indicate whole edit box must be redrawn

8.1.16.1.6 LB_DRAW_FOCUS Macro

File

ListBox.h (see page 794)

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

8.1.16.1.7 LB_DRAW_ITEMS Macro

File

ListBox.h (see page 794)

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

8.1.16.1.8 LB_FOCUSED Macro

File

ListBox.h (see page 794)

C

```
#define LB_FOCUSED 0x0001  // Bit for focused state
```

Description

Bit for focused state

8.1.16.1.9 LB_HIDE Macro

File

ListBox.h (see page 794)

C

```
#define LB_HIDE 0x8000  // Bit to remove object from screen
```

Description

Bit to remove object from screen

8.1.16.2 List Item Status

Macros

Name	Description
LB_STS_SELECTED (see page 181)	Item is selected.
LB_STS_REDRAW (see page 181)	Item is to be redrawn.

Module

List Box (see page 177)

Description

List of Items status.

8.1.16.2.1 LB_STS_SELECTED Macro

File

ListBox.h (see page 794)

C

```
#define LB_STS_SELECTED 0x0001  // Item is selected.
```

Description

Item is selected.

8.1.16.2.2 LB_STS_REDRAW Macro

File

ListBox.h (see page 794)

C

```
#define LB_STS_REDRAW 0x0002  // Item is to be redrawn.
```

Description

Item is to be redrawn.

8.1.16.3 LbCreate Function

File

ListBox.h (see page 794)

C

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

Module

List Box (see page 177)

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.
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 (see page 195) 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 (see page 195) *LbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR (see page 345)* pText, GOL_SCHEME (see page 331) *pScheme)

8.1.16.4 LbDraw Function

File

ListBox.h (see page 794)

C

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

Module

List Box (see page 177)

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)

8.1.16.5 LbGetItemList Macro

File

ListBox.h (see page 794)

C

```
#define LbGetItemList(pLb) ((LISTITEM *)pLb->pItemList)
```

Module

List Box (see page 177)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box object.

Side Effects

none

Returns

Returns the pointer to the LISTITEM (see page 196) used in the list box.

Preconditions

none

Example

See LbAddItem (see page 184)() example.

Overview

This function returns the pointer to the current item list used in the list box.

Syntax

LISTITEM (see page 196) LbGetItemList(LISTBOX (see page 195)* pLb)

8.1.16.6 LbAddItem Function

File

ListBox.h (see page 794)

C

```
LISTITEM * LbAddItem(
    LISTBOX * pLb,
    LISTITEM * pPrevItem,
    XCHAR * pText,
    void * pBitmap,
    WORD status,
    WORD data
);
```

Module

List Box (see page 177)

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 (see page 181) or LB_STS_REDRAW (see page 181)). Refer to LISTITEM (see page 196) 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 (see page 196) and adds it to the list box. The newly created LISTITEM (see page 196) will store the location of pText, pBitmap and other parameters describing the added item.

Syntax

LISTITEM (see page 196)* LbAddItem(LISTBOX (see page 195) *pLb, LISTITEM (see page 196) *pPrevItem, XCHAR (see page 345) *pText, void* pBitmap, WORD status, WORD data)

8.1.16.7 LbDelItem Function

File

ListBox.h (see page 794)

C

```
void LbDelItem(
    LISTBOX * pLb,
    LISTITEM * pItem
);
```

Module

List Box (see page 177)

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 LbDelItem(LISTBOX (see page 195) *pLb, LISTITEM (see page 196) *pItem)

8.1.16.8 LbChangeSel Function

File

ListBox.h (see page 794)

C

```
void LbChangeSel(
    LISTBOX * pLb,
    LISTITEM * pItem
);
```

Module

List Box (see page 177)

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 (see page 195) *pLb, LISTITEM (see page 196) *pItem)

8.1.16.9 LbSetSel Macro

File

ListBox.h (see page 794)

C

```
#define LbSetSel(pLb, pItem) \
    if(!(pItem->status & LB_STS_SELECTED)) \
        LbChangeSel((LISTBOX *)pLb, pItem);
```

Module

List Box (see page 177)

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)

8.1.16.10 LbGetSel Function

File

ListBox.h (see page 794)

C

```
LISTITEM * LbGetSel(
    LISTBOX * pLb,
    LISTITEM * pFromItem
);
```

Module

List Box (see page 177)

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 (see page 196)* LbGetSel(LISTBOX (see page 195) *pLb, LISTITEM (see page 196) *pFromItem)

8.1.16.11 LbGetFocusedItem Function

File

ListBox.h (see page 794)

C

```
SHORT LbGetFocusedItem(
    LISTBOX * pLb
);
```

Module

List Box (see page 177)

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 (see page 195)* pLb)

8.1.16.12 LbSetFocusedItem Function

File

ListBox.h (see page 794)

C

```
void LbSetFocusedItem(
    LISTBOX * pLb,
    SHORT index
);
```

Module

List Box (see page 177)

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 (see page 195)* pLb, SHORT index)

8.1.16.13 LbGetCount Macro

File

ListBox.h (see page 794)

C

```
#define LbGetCount(pLb) ((LISTBOX *)pLb)->itemsNumber
```

Module

List Box (see page 177)

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)

8.1.16.14 LbGetVisibleCount Macro

File

ListBox.h (see page 794)

C

```
#define LbGetVisibleCount(pLb) \
        (

\
            (((LISTBOX *)pLb)->hdr.bottom - ((LISTBOX *)pLb)->hdr.top - 2 *
(GOL_EMBOSS_SIZE + LB_INDENT)) / \
                ((LISTBOX
*)pLb)->textHeight                                                          \
        )
```

Module

List Box (see page 177)

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)

8.1.16.15 LbClrtSel Macro

File

ListBox.h (see page 794)

C

```
#define LbClrtSel(pLb, pItem) \
    if(pItem->status & LB_STS_SELECTED) \
        LbChangeSel((LISTBOX *)pLb, pItem);
```

Module

List Box (see page 177)

Input Parameters

Input Parameters	Description
pLb	The pointer to the list box.
pItem	The pointer to the item the selection status should be cleared.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro clears the selection of an item.

Syntax

LbClrSel(pLb, pItem)

8.1.16.16 LbSetBitmap Macro

File

ListBox.h (see page 794)

C

```
#define LbSetBitmap(pItem, pBtmap) ((LISTITEM *)pItem)->pBitmap = pBtmap
```

Module

List Box (see page 177)

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 (see page 184)() example.

Overview

This macro sets the bitmap used in the item.

Syntax

LbSetBitmap(pItem, pBtmap)

8.1.16.17 LbGetBitmap Macro

File

ListBox.h (see page 794)

C

```
#define LbGetBitmap(pItem) ((LISTITEM *)pItem)->pBitmap
```

Module

List Box (see page 177)

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(pItem)

8.1.16.18 LbDelItemsList Function

File

ListBox.h (see page 794)

C

```
void LbDelItemsList(
    void * pObj
);
```

Module

List Box (see page 177)

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 LbDelItemsList(void *pObj)

8.1.16.19 LbMsgDefault Function

File

ListBox.h (see page 794)

C

```
void LbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

List Box (see page 177)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL (see page 48) 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 (see page 180), Set LB_DRAW_FOCUS (see page 180)	If focus is enabled, the focus state bit LB_FOCUSED (see page 180) will be set. LB_DRAW_FOCUS (see page 180) draw state bit will force the List Box to be redrawn with focus.
		Set LB_DRAW_ITEMS (see page 180)	List Box will redrawn with selected item(s).
LB_MSG_MOVE	KeyBoard	Set LB_DRAW_ITEMS (see page 180)	List Box will redrawn with focus on one item.
LB_MSG_SEL	KeyBoard	Set LB_DRAW_ITEMS (see page 180)	List Box will redrawn with selection on the current item focused.

Syntax

void LbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316) *pMsg)

8.1.16.20 LbTranslateMsg Function

File

ListBox.h (see page 794)

C

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

Module

List Box (see page 177)

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 (see page 48) 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 (see page 316) *pMsg)

8.1.16.21 LISTBOX Structure

File

ListBox.h (see page 794)

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 (see page 52)).
LISTITEM * pItemList;	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 (see page 419) displacement for the list.
SHORT textHeight;	Pre-computed text height.

Module

List Box (see page 177)

Overview

Defines the parameters required for a list box Object.

8.1.16.22 LISTITEM Structure

File

ListBox.h (see page 794)

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 next 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 (see page 177)

Overview

Defines the parameters required for a list item used in list box.

8.1.17 Meter

Meter is an Object that can be used to graphically display a sampled input.

Functions

	Name	Description
	MtrCreate (see page 200)	This function creates a METER (see page 209) 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 (see page 201)	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... more (see page 201)
	MtrSetVal (see page 202)	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 (see page 208)	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 (see page 208)	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 (see page 203)	This macro returns the current value of the meter. Value is always in the minValue-maxValue range inclusive.
MtrDecVal (see page 203)	This macro is used to directly decrement the value.
MtrIncVal (see page 204)	This macro is used to directly increment the value.
MtrSetScaleColors (see page 204)	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 (see page 356) 6 225 to 180 not used not used Arc (see page 356) 5 179 to 135 179 to 135 not used Arc (see page 356) 4 134 to 90 134 to 90 not used Arc (see page 356) 3 89 to 45 89 to 45 89 to 45 Arc (see page 356) 2 44 to 0 44... more (see page 204)
MtrSetTitleFont (see page 205)	This function sets the font of title.
MtrSetValueFont (see page 206)	This function sets the font of value.
METER_TYPE (see page 206)	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 (see page 817) file and... more (see page 206)
METER_DISPLAY_VALUES_ENABLE (see page 207)	This enables the display of the values.
MTR_ACCURACY (see page 207)	Sets the meter accuracy to one decimal places
RESOLUTION (see page 207)	Factor that the meter widget will divide minValue, maxValue

Structures

Name	Description
METER (see page 209)	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 (see page 208)() 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 (see page 204)() 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.

8.1.17.1 Meter States

Macros

Name	Description
MTR_DISABLED (see page 198)	Bit for disabled state.
MTR_DRAW (see page 199)	Bit to indicate object must be redrawn.
MTR_HIDE (see page 199)	Bit to indicate object must be removed from screen.
MTR_RING (see page 199)	Bit for ring type, scales are drawn over the ring
MTR_DRAW_UPDATE (see page 199)	Bit to indicate an update only.

Module

Meter (see page 196)

Description

List of Meter (see page 196) bit states.

8.1.17.1.1 MTR_DISABLED Macro

File

Meter.h (see page 817)

C

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

Description

Bit for disabled state.

8.1.17.1.2 MTR_DRAW Macro

File

Meter.h (see page 817)

C

```
#define MTR_DRAW 0x4000      // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

8.1.17.1.3 MTR_HIDE Macro

File

Meter.h (see page 817)

C

```
#define MTR_HIDE 0x8000      // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

8.1.17.1.4 MTR_RING Macro

File

Meter.h (see page 817)

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

8.1.17.1.5 MTR_DRAW_UPDATE Macro

File

Meter.h (see page 817)

C

```
#define MTR_DRAW_UPDATE 0x1000      // Bit to indicate an update only.
```

Description

Bit to indicate an update only.

8.1.17.2 MtrCreate Function

File

Meter.h (see page 817)

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 (see page 196)

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.
XCHAR * pText	Pointer to the text label of the meter.
GOL_SCHEME * pScheme	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 GOLSmallFont;       // 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*)&GOLSmallFont,       // 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 (see page 209) 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 (see page 209) *MtrCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT value, SHORT minValue, SHORT maxValue, XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.17.3 MtrDraw Function

File

Meter.h (see page 817)

C

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

Module

Meter (see page 196)

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 (see page 200)() 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)

8.1.17.4 MtrSetVal Function

File

Meter.h (see page 817)

C

```
void MtrSetVal(
    METER * pMtr,
    SHORT newVal
);
```

Module

Meter (see page 196)

Input Parameters

Input Parameters	Description
METER * pMtr	The pointer to the object.
SHORT newVal	New value to be set for the Meter (see page 196).

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 (see page 209) *pMtr, SHORT newVal)

8.1.17.5 MtrGetVal Macro

File

Meter.h (see page 817)

C

```
#define MtrGetVal(pMtr) ((pMtr)->value)
```

Module

Meter (see page 196)

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)

8.1.17.6 MtrDecVal Macro

File

Meter.h (see page 817)

C

```
#define MtrDecVal(pMtr, deltaValue) MtrSetVal(pMtr, ((pMtr)->value - deltaValue))
```

Module

Meter (see page 196)

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.
deltaValue	Number to be subtracted to the current Meter (see page 196) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly decrement the value.

Syntax

MtrDecVal(pMtr, deltaValue)

8.1.17.7 MtrIncVal Macro

File

Meter.h (see page 817)

C

```
#define MtrIncVal(pMtr, deltaValue) MtrSetVal(pMtr, ((pMtr)->value + deltaValue))
```

Module

Meter (see page 196)

Input Parameters

Input Parameters	Description
pMtr	Pointer to the object.
deltaValue	Number to be added to the current Meter (see page 196) value.

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro is used to directly increment the value.

Syntax

MtrIncVal(pMtr, deltaValue)

8.1.17.8 MtrSetScaleColors Macro

File

Meter.h (see page 817)

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 (see page 196)

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 (see page 200)()) 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 (see page 356) 6 225 to 180 not used not used Arc (see page 356) 5 179 to 135 179 to 135 not used Arc (see page 356) 4 134 to 90 134 to 90 not used Arc (see page 356) 3 89 to 45 89 to 45 89 to 45 Arc (see page 356) 2 44 to 0 44 to 0 44 to 0 Arc (see page 356) 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; }

8.1.17.9 MtrSetTitleFont Macro

File

Meter.h (see page 817)

C

```
#define MtrSetTitleFont(pMtr, pNewFont) (((METER *)pMtr)->pTitleFont = pNewFont)
```

Module

Meter (see page 196)

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 (see page 209)*)pMtr)->pTitleFont = pNewFont)

8.1.17.10 MtrSetValueFont Macro

File

Meter.h (see page 817)

C

```
#define MtrSetValueFont(pMtr, pNewFont) (((METER *)pMtr)->pValueFont = pNewFont)
```

Module

Meter (see page 196)

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 (see page 209)*)pMtr)->pValueFont = pNewFont)

8.1.17.11 METER_TYPE Macro

File

Meter.h (see page 817)

C

```
#define METER_TYPE MTR_WHOLE_TYPE
```

Module

Meter (see page 196)

Overview

This is a compile time setting to select the type if meter shape. There are three types:

- MTR_WHOLE_TYPE - Meter (see page 196) drawn with 6 octants used.

- MTR_HALF_TYPE - Meter (see page 196) drawn with semi circle shape.
- MTR_QUARTER_TYPE - Meter (see page 196) 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 (see page 817) file and arc colors using MtrSetScaleColors (see page 204)(pMtr, arc1, arc2, arc3, arc4, arc5, arc6) macro.

8.1.17.12 METER_DISPLAY_VALUES_ENABLE Macro

File

Meter.h (see page 817)

C

```
#define METER_DISPLAY_VALUES_ENABLE
```

Module

Meter (see page 196)

Description

This enables the display of the values.

8.1.17.13 MTR_ACCURACY Macro

File

Meter.h (see page 817)

C

```
#define MTR_ACCURACY 0x0008      // Sets the meter accuracy to one decimal places
```

Module

Meter (see page 196)

Description

Sets the meter accuracy to one decimal places

8.1.17.14 RESOLUTION Macro

File

Meter.h (see page 817)

C

```
#define RESOLUTION 10              // Factor that the meter widget will divide minValue,
maxValue
```

Module

Meter (see page 196)

Description

Factor that the meter widget will divide minValue, maxValue

8.1.17.15 MtrMsgDefault Function

File

Meter.h (see page 817)

C

```
void MtrMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Meter (see page 196)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL (see page 48) 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 (see page 196) 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 (see page 199)	Meter (see page 196) will be redrawn to update the needle position and value displayed.

Syntax

MtrMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.17.16 MtrTranslateMsg Function

File

Meter.h (see page 817)

C

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

Module

Meter (see page 196)

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 (see page 48) message:

- MTR_MSG_SET - Meter (see page 196) ID is given in parameter 1 for a TYPE_SYSTEM message.
- OBJ_MSG_INVALID - Meter (see page 196) 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 (see page 316) *pMsg)

8.1.17.17 METER Structure

File

Meter.h (see page 817)

C

```
typedef struct {
  OBJ_HEADER hdr;
  XCHAR * pText;
  SHORT value;
  SHORT minValue;
  SHORT maxValue;
  SHORT xCenter;
  SHORT yCenter;
  SHORT radius;
  SHORT xPos;
  SHORT yPos;
  WORD arcColor6;
  WORD arcColor5;
  WORD arcColor4;
  WORD arcColor3;
  WORD arcColor2;
  WORD arcColor1;
```

```
    void * pTitleFont;
    void * pValueFont;
} METER;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (see page 52)).
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.
WORD arcColor6;	Arc (see page 356) 6 color parameter.
WORD arcColor5;	Arc (see page 356) 5 color parameter
WORD arcColor4;	Arc (see page 356) 4 color parameter
WORD arcColor3;	Arc (see page 356) 3 color parameter
WORD arcColor2;	Arc (see page 356) 2 color parameter
WORD arcColor1;	Arc (see page 356) 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 (see page 196)

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.

8.1.17.18 Files

8.1.18 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 (see page 212)	This function creates a PICTURE (see page 217) 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 (see page 213)	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 (see page 216)	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 (see page 217) Object.

Macros

Name	Description
PictSetBitmap (see page 214)	This macro sets the bitmap used in the object.
PictGetBitmap (see page 215)	This macro returns the pointer to the bitmap used in the object.
PictGetScale (see page 215)	This macro returns the current scale factor used to render the bitmap.
PictSetScale (see page 216)	This macro sets the scale factor used to render the bitmap used in the object.

Structures

Name	Description
PICTURE (see page 217)	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.

8.1.18.1 Picture States

Macros

Name	Description
PICT_DISABLED (see page 211)	Bit to indicate Picture (see page 210) is in a disabled state.
PICT_DRAW (see page 212)	Bit to indicate Picture (see page 210) will be redrawn.
PICT_FRAME (see page 212)	Bit to indicate Picture (see page 210) has a frame.
PICT_HIDE (see page 212)	Bit to indicate Picture (see page 210) must be hidden.

Module

Picture Control (see page 210)

Description

List of Picture (see page 210) Control bit states.

8.1.18.1.1 PICT_DISABLED Macro

File

Picture.h (see page 830)

C

```
#define PICT_DISABLED 0x0002  // Bit to indicate Picture is in a disabled state.
```

Description

Bit to indicate Picture (see page 210) is in a disabled state.

8.1.18.1.2 PICT_DRAW Macro

File

Picture.h (see page 830)

C

```
#define PICT_DRAW 0x4000  // Bit to indicate Picture will be redrawn.
```

Description

Bit to indicate Picture (see page 210) will be redrawn.

8.1.18.1.3 PICT_FRAME Macro

File

Picture.h (see page 830)

C

```
#define PICT_FRAME 0x0004  // Bit to indicate Picture has a frame.
```

Description

Bit to indicate Picture (see page 210) has a frame.

8.1.18.1.4 PICT_HIDE Macro

File

Picture.h (see page 830)

C

```
#define PICT_HIDE 0x8000  // Bit to indicate Picture must be hidden.
```

Description

Bit to indicate Picture (see page 210) must be hidden.

8.1.18.2 PictCreate Function

File

Picture.h (see page 830)

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 (see page 210)

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 (see page 217) 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 (see page 217) *PictCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, char scale, void *pBitmap, GOL_SCHEME (see page 331) *pScheme)

8.1.18.3 PictDraw Function

File

Picture.h (see page 830)

C

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

Module

Picture Control (see page 210)

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)

8.1.18.4 PictSetBitmap Macro

File

Picture.h (see page 830)

C

```
#define PictSetBitmap(pPict, pBtmap) ((PICTURE*)pPict)->pBitmap = pBtmap
```

Module

Picture Control (see page 210)

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)

8.1.18.5 PictGetBitmap Macro

File

Picture.h (see page 830)

C

```
#define PictGetBitmap(pPict) ((PICTURE*)pPict)->pBitmap
```

Module

Picture Control (see page 210)

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)

8.1.18.6 PictGetScale Macro

File

Picture.h (see page 830)

C

```
#define PictGetScale(pPict) pPict->scale
```

Module

Picture Control (see page 210)

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)

8.1.18.7 PictSetScale Macro

File

Picture.h (see page 830)

C

```
#define PictSetScale(pPict, scl) pPict->scale = scl
```

Module

Picture Control (see page 210)

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)

8.1.18.8 PictTranslateMsg Function

File

Picture.h (see page 830)

C

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

Module

Picture Control (see page 210)

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 (see page 48) message:

- PICT_MSG_SELECTED – Picture (see page 210) is touched.
- OBJ_MSG_INVALID – Picture (see page 210) 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 (see page 217) 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 (see page 316) *pMsg)

8.1.18.9 PICTURE Structure

File

Picture.h (see page 830)

C

```
typedef struct {
  OBJ_HEADER hdr;
  char scale;
  void * pBitmap;
} PICTURE;
```

Members

Members	Description
OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER (see page 52)).
char scale;	Scale factor for the bitmap
void * pBitmap;	Pointer to the bitmap

Module

Picture Control (see page 210)

Overview

The structure contains data for picture control

8.1.19 Progress Bar

Progress Bar (see page 352) 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 (see page 220)	This function creates a PROGRESSBAR (see page 226) 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 (see page 221)	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 (see page 222)	This function sets the range of the progress bar. Calling this function also resets the position equal to the new range value.
	PbSetPos (see page 223)	This function sets the position of the progress bar. Position should be in the given range inclusive.
	PbTranslateMsg (see page 225)	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 (see page 223)	This macro returns the current range of the progress bar.
PbGetPos (see page 224)	This macro returns the current progress bar position.

Structures

Name	Description
PROGRESSBAR (see page 226)	The structure contains data for the progress bar

Description

Progress Bar (see page 352) supports only Touchscreen inputs, replying to their events with the message: PB_MSG_SELECTED.

The Progress Bar (see page 352) 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.

8.1.19.1 Progress Bar States

Macros

Name	Description
PB_DISABLED (see page 219)	Bit to indicate Progress Bar (see page 352) is in a disabled state.
PB_DRAW (see page 219)	Bit to indicate Progress Bar (see page 352) must be redrawn.
PB_DRAW_BAR (see page 220)	Bit to indicate Progress Bar (see page 352) must be redrawn.
PB_HIDE (see page 220)	Bit to indicate Progress Bar (see page 352) must be hidden.
PB_VERTICAL (see page 220)	Bit for orientation (0 - horizontal, 1 - vertical)

Module

Progress Bar (see page 218)

Description

List of Progress Bar (see page 218) bit states.

8.1.19.1.1 PB_DISABLED Macro

File

ProgressBar.h (see page 840)

C

```
#define PB_DISABLED 0x0002  // Bit to indicate Progress Bar is in a disabled state.
```

Description

Bit to indicate Progress Bar (see page 352) is in a disabled state.

8.1.19.1.2 PB_DRAW Macro

File

ProgressBar.h (see page 840)

C

```
#define PB_DRAW 0x4000  // Bit to indicate Progress Bar must be redrawn.
```

Description

Bit to indicate Progress Bar (see page 352) must be redrawn.

8.1.19.1.3 PB_DRAW_BAR Macro

File

ProgressBar.h (see page 840)

C

```
#define PB_DRAW_BAR 0x2000  // Bit to indicate Progress Bar must be redrawn.
```

Description

Bit to indicate Progress Bar (see page 352) must be redrawn.

8.1.19.1.4 PB_HIDE Macro

File

ProgressBar.h (see page 840)

C

```
#define PB_HIDE 0x8000  // Bit to indicate Progress Bar must be hidden.
```

Description

Bit to indicate Progress Bar (see page 352) must be hidden.

8.1.19.1.5 PB_VERTICAL Macro

File

ProgressBar.h (see page 840)

C

```
#define PB_VERTICAL 0x0004  // Bit for orientation  (0 - horizontal, 1 - vertical)
```

Description

Bit for orientation (0 - horizontal, 1 - vertical)

8.1.19.2 PbCreate Function

File

ProgressBar.h (see page 840)

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 (see page 218)

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,     // ID
                     50,90,270,140,       // dimension
                     PB_DRAW,             // Draw the object
                     25,                  // position
                     50,                  // set the range
                     NULL);               // use default GOL scheme
    while(!PbDraw(pPBar));
}
```

Overview

This function creates a PROGRESSBAR (see page 226) 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 (see page 226) *PbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, WORD pos, WORD range, GOL_SCHEME (see page 331) *pScheme)

8.1.19.3 PbDraw Function

File

ProgressBar.h (see page 840)

C

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

Module

Progress Bar (see page 218)

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 (see page 220)() 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)

8.1.19.4 PbSetRange Function

File

ProgressBar.h (see page 840)

C

```
void PbSetRange(
    PROGRESSBAR * pPb,
    WORD range
);
```

Module

Progress Bar (see page 218)

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 (see page 226) *pPb, WORD range)

8.1.19.5 PbGetRange Macro

File

ProgressBar.h (see page 840)

C

```
#define PbGetRange(pPb) (pPb->range)
```

Module

Progress Bar (see page 218)

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)

8.1.19.6 PbSetPos Function

File

ProgressBar.h (see page 840)

C

```
void PbSetPos(
    PROGRESSBAR * pPb,
    WORD position
);
```

Module

Progress Bar (see page 218)

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 (see page 226) *pPb, WORD position)

8.1.19.7 PbGetPos Macro

File

ProgressBar.h (see page 840)

C

```
#define PbGetPos(pPb) pPb->pos
```

Module

Progress Bar (see page 218)

Input Parameters

Input Parameters	Description
pPb	Pointer to the object

Side Effects

none

Returns

Returns the progress bar position.

Preconditions

none

Example

See PbSetPos (see page 223)() exmaple.

Overview

This macro returns the current progress bar position.

Syntax

PbGetPos(pPb)

8.1.19.8 PbTranslateMsg Function

File

ProgressBar.h (see page 840)

C

```
WORD PbTranslateMsg(
    PROGRESSBAR * pPb,
    GOL_MSG * pMsg
);
```

Module

Progress Bar (see page 218)

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 (see page 48) message:

- PB_MSG_SELECTED – Progress Bar (see page 352) is selected.
- OBJ_MSG_INVALID – Progress Bar (see page 352) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 226) *pPb, GOL_MSG (see page 316) *pMsg)

8.1.19.9 PROGRESSBAR Structure

File

ProgressBar.h (see page 840)

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 (see page 52)).
WORD pos;	Current progress position.
WORD prevPos;	Previous progress position.
WORD range;	Sets the range of the object.

Module

Progress Bar (see page 218)

Overview

The structure contains data for the progress bar

8.1.20 Radio Button

Radio Button (see page 65) 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 (see page 230)	This function creates a RADIOBUTTON (see page 237) 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 (see page 231)	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 (see page 232)	This function returns the ID of the currently checked Radio Button (see page 65) in the group.
	RbSetCheck (see page 233)	This function sets the Radio Button (see page 65) with the given ID to its checked state.
	RbSetText (see page 234)	This function sets the string used for the object.
	RbMsgDefault (see page 235)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	RbTranslateMsg (see page 236)	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 (see page 233)	This macro returns the address of the current text string used for the object.

Structures

Name	Description
RADIOBUTTON (see page 237)	the structure contains data for the Radio Button (see page 65)

Description

Radio Button (see page 65) supports both Keyboard and Touchscreen inputs, replying to their events with the message: RB_MSG_CHECKED.

The Radio Button (see page 65) Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.

8.1.20.1 Radio Button States

Macros

Name	Description
RB_CHECKED (see page 228)	Bit to indicate Radio Button (see page 65) is checked.
RB_DISABLED (see page 228)	Bit for disabled state.
RB_DRAW (see page 228)	Bit to indicate whole Radio Button (see page 65) must be redrawn.
RB_DRAW_CHECK (see page 229)	Bit to indicate check mark should be redrawn.
RB_DRAW_FOCUS (see page 229)	Bit to indicate focus must be redrawn.
RB_FOCUSED (see page 229)	Bit for focused state.
RB_GROUP (see page 229)	Bit to indicate the first Radio Button (see page 65) in the group.
RB_HIDE (see page 229)	Bit to indicate that button must be removed from screen.

Module

Radio Button (see page 226)

Description

List of Radio Button (see page 226) bit states.

8.1.20.1.1 RB_CHECKED Macro

File

RadioButton.h (see page 866)

C

```
#define RB_CHECKED 0x0004  // Bit to indicate Radio Button is checked.
```

Description

Bit to indicate Radio Button (see page 65) is checked.

8.1.20.1.2 RB_DISABLED Macro

File

RadioButton.h (see page 866)

C

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

Description

Bit for disabled state.

8.1.20.1.3 RB_DRAW Macro

File

RadioButton.h (see page 866)

C

```
#define RB_DRAW 0x4000  // Bit to indicate whole Radio Button must be redrawn.
```

Description

Bit to indicate whole Radio Button (see page 65) must be redrawn.

8.1.20.1.4 RB_DRAW_CHECK Macro

File

RadioButton.h (see page 866)

C

```
#define RB_DRAW_CHECK 0x1000  // Bit to indicate check mark should be redrawn.
```

Description

Bit to indicate check mark should be redrawn.

8.1.20.1.5 RB_DRAW_FOCUS Macro

File

RadioButton.h (see page 866)

C

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

Description

Bit to indicate focus must be redrawn.

8.1.20.1.6 RB_FOCUSED Macro

File

RadioButton.h (see page 866)

C

```
#define RB_FOCUSED 0x0001  // Bit for focused state.
```

Description

Bit for focused state.

8.1.20.1.7 RB_GROUP Macro

File

RadioButton.h (see page 866)

C

```
#define RB_GROUP 0x0008  // Bit to indicate the first Radio Button in the group.
```

Description

Bit to indicate the first Radio Button (see page 65) in the group.

8.1.20.1.8 RB_HIDE Macro

File

RadioButton.h (see page 866)

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.

8.1.20.2 RbCreate Function

File

RadioButton.h (see page 866)

C

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

Module

Radio Button (see page 226)

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 (see page 65).
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,              // ID
                      255,40,310,80,                // dimension
                      RB_DRAW|RB_GROUP|RB_CHECKED,  // will be dislayed and
                                                    // focused after creation
                                                    // first button in the group
                      "RB1",                        // text
                      pScheme);                     // scheme used

    pRb[1] = RbCreate(ID_RADIOBUTTON2,              // ID
                      255,85,310,125,               // dimension
                      RB_DRAW,                      // will be dislayed and
                                                    // checked after creation
                      "RB2",                        // text
                      pScheme);                     // scheme used

    pRb[2] = RbCreate(ID_RADIOBUTTON3,              // ID
```

```
                        255,130,310,170,            // dimension
                        RB_DRAW,                    // will be dislayed and
                                                    // disabled after creation
                        "RB3",                      // text
                        pScheme);                   // scheme used

    while(!RbDraw(pRb[0]));                         // draw the objects
    while(!RbDraw(pRb[1]));
    while(!RbDraw(pRb[2]));
```

Overview

This function creates a RADIOBUTTON (see page 237) 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 (see page 237) *RbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.20.3 RbDraw Function

File

RadioButton.h (see page 866)

C

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

Module

Radio Button (see page 226)

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 (see page 230)() 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)

8.1.20.4 RbGetCheck Function

File

RadioButton.h (see page 866)

C

```
WORD RbGetCheck(
    RADIOBUTTON * pRb
);
```

Module

Radio Button (see page 226)

Input Parameters

Input Parameters	Description
RADIOBUTTON * pRb	Pointer to the Radio Button (see page 65) 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,                // ID
                      255,40,310,80,                  // dimension
                      RB_DRAW|RB_GROUP|RB_CHECKED,    // will be dislayed and
                                                      // focused after creation
                                                      // first button in the group
                      "RB1",                          // text
                      pScheme);                       // scheme used

    pRb[1] = RbCreate(ID_RADIOBUTTON2,                // ID
                      255,85,310,125,                 // dimension
                      RB_DRAW,                        // will be dislayed and
                                                      // checked after creation
                      "RB2",                          // text
                      pScheme);                       // scheme used

    pRb[2] = RbCreate(ID_RADIOBUTTON3,                // ID
                      255,130,310,170,                // dimension
                      RB_DRAW,                        // will be dislayed and
                                                      // disabled after creation
                      "RB3",                          // text
                      pScheme);                       // 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 (see page 65) in the group.

Syntax

WORD RbGetCheck(RADIOBUTTON (see page 237) *pRb)

8.1.20.5 RbSetCheck Function

File

RadioButton.h (see page 866)

C

```
void RbSetCheck(
    RADIOBUTTON * pRb,
    WORD ID
);
```

Module

Radio Button (see page 226)

Input Parameters

Input Parameters	Description
RADIOBUTTON * pRb	Pointer to the Radio Button (see page 65) in the group.
WORD ID	ID of the object to be checked.

Side Effects

none

Returns

none

Preconditions

none

Example

See RbGetCheck (see page 232)() example.

Overview

This function sets the Radio Button (see page 65) with the given ID to its checked state.

Syntax

void RbSetCheck(RADIOBUTTON (see page 237) *pRb, WORD ID)

8.1.20.6 RbGetText Macro

File

RadioButton.h (see page 866)

C

```
#define RbGetText(pRb) pRb->pText
```

Module

Radio Button (see page 226)

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)

8.1.20.7 RbSetText Function

File

RadioButton.h (see page 866)

C

```
void RbSetText(
    RADIOBUTTON * pRb,
    XCHAR * pText
);
```

Module

Radio Button (see page 226)

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 (see page 237) *pRb, XCHAR (see page 345) *pText)

8.1.20.8 RbMsgDefault Function

File

RadioButton.h (see page 866)

C

```
void RbMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Radio Button (see page 226)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message
GOL_MSG * pMsg	The pointer to the GOL (see page 48) message
pRb	The pointer to the object whose state will be modified

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg (see page 75)() 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 (see page 228),	Depending on the current value of RB_CHECKED (see page 228) Check Box will be redrawn.
	Keyboard	Set/Clear RB_CHECKED (see page 228)	

Syntax

RbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.20.9 RbTranslateMsg Function

File

RadioButton.h (see page 866)

C

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

Module

Radio Button (see page 226)

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 (see page 48) message:

- RB_MSG_CHECKED – Radio Button (see page 65) is checked
- OBJ_MSG_INVALID – Radio Button (see page 65) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 65) while the Radio Button (see page 65) 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 (see page 321) or SCAN_SPACE_PRESSED (see page 325) while the Radio Button (see page 65) is not checked.
OBJ_MSG_INVALID	Any	Any	If the message did not affect the object.

Syntax

WORD RbTranslateMsg(void *pObj, GOL_MSG (see page 316) *pMsg)

8.1.20.10 RADIOBUTTON Structure

File

RadioButton.h (see page 866)

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 (see page 52)).
OBJ_HEADER * pHead;	Pointer to the first Radio Button (see page 65) in the group
OBJ_HEADER * pNext;	Pointer to the next Radio Button (see page 65) in the group
SHORT textHeight;	Pre-computed text height
XCHAR * pText;	Pointer to the text

Module

Radio Button (see page 226)

Overview

the structure contains data for the Radio Button (see page 65)

8.1.21 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 (see page 241)	This function creates a SLIDER (see page 252) object with the parameters given. Depending on the SLD_SCROLLBAR (see page 241) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (see page 241) 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 (see page 243)	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 (see page 244)	This sets the page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos (see page 248)() or decremented via SldDecPos (see page 249)(). Page size minimum value is 1. Maximum value is range/2.
	SldSetPos (see page 245)	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 (see page 247)	This sets the range of the thumb. If this field is changed Object must be completely redrawn to reflect the change.

	SldMsgDefault (see page 250)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	SldTranslateMsg (see page 251)	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 (see page 244)	Returns returns the current slider thumb position. Value is always in the 1 to range/2 inclusive.
SldGetPos (see page 246)	Returns returns the current position of the slider thumb.
SldGetRange (see page 247)	Returns the current slider page. Value is always in the 1-range/2 inclusive for both slider and scrollbar type.
SldIncPos (see page 248)	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 (see page 249)	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 (see page 252)	Defines the parameters required for a slider/scrollbar Object. Depending on the SLD_SCROLLBAR (see page 241) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (see page 241) 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.

8.1.21.1 Slider States

Macros

Name	Description
SLD_DISABLED (see page 239)	Bit for disabled state
SLD_DRAW (see page 240)	Bit to indicate whole slider must be redrawn
SLD_DRAW_FOCUS (see page 240)	Bit to indicate that only the focus will be redrawn
SLD_DRAW_THUMB (see page 240)	Bit to indicate that only thumb area must be redrawn
SLD_FOCUSED (see page 240)	Bit for focus state
SLD_HIDE (see page 240)	Bit to remove object from screen
SLD_SCROLLBAR (see page 241)	Bit for type usage (0 - Slider (see page 237), 1 - ScrollBar)
SLD_VERTICAL (see page 241)	Bit for orientation (0 - horizontal, 1 - vertical)

Module

Slider/Scroll Bar (see page 237)

Description

List of Slider (see page 237) bit states.

8.1.21.1.1 SLD_DISABLED Macro

File

Slider.h (see page 886)

C

```
#define SLD_DISABLED 0x0002  // Bit for disabled state
```

Description

Bit for disabled state

8.1.21.1.2 SLD_DRAW Macro

File

Slider.h (see page 886)

C

```
#define SLD_DRAW 0x4000  // Bit to indicate whole slider must be redrawn
```

Description

Bit to indicate whole slider must be redrawn

8.1.21.1.3 SLD_DRAW_FOCUS Macro

File

Slider.h (see page 886)

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

8.1.21.1.4 SLD_DRAW_THUMB Macro

File

Slider.h (see page 886)

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

8.1.21.1.5 SLD_FOCUSED Macro

File

Slider.h (see page 886)

C

```
#define SLD_FOCUSED 0x0001  // Bit for focus state
```

Description

Bit for focus state

8.1.21.1.6 SLD_HIDE Macro

File

Slider.h (see page 886)

C

```
#define SLD_HIDE 0x8000  // Bit to remove object from screen
```

Description

Bit to remove object from screen

8.1.21.1.7 SLD_SCROLLBAR Macro

File

Slider.h (see page 886)

C

```
#define SLD_SCROLLBAR 0x0010  // Bit for type usage (0 - Slider, 1 - ScrollBar)
```

Description

Bit for type usage (0 - Slider (see page 237), 1 - ScrollBar)

8.1.21.1.8 SLD_VERTICAL Macro

File

Slider.h (see page 886)

C

```
#define SLD_VERTICAL 0x0004  // Bit for orientation  (0 - horizontal, 1 - vertical)
```

Description

Bit for orientation (0 - horizontal, 1 - vertical)

8.1.21.2 SldCreate Function

File

Slider.h (see page 886)

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 (see page 237)

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 (see page 252) object with the parameters given. Depending on the SLD_SCROLLBAR (see page 241) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (see page 241) 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 (see page 252) *SldCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT range, SHORT page, SHORT pos, GOL_SCHEME (see page 331) *pScheme);

8.1.21.3 SldDraw Function

File

Slider.h (see page 886)

C

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

Module

Slider/Scroll Bar (see page 237)

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 (see page 241)() 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)

8.1.21.4 SldSetPage Function

File

Slider.h (see page 886)

C

```
void SldSetPage(
    SLIDER * pSld,
    WORD newPage
);
```

Module

Slider/Scroll Bar (see page 237)

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 (see page 248)() example.

Overview

This sets the page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos (see page 248)() or decremented via SldDecPos (see page 249)(). Page size minimum value is 1. Maximum value is range/2.

Syntax

void SldSetPage(SLIDER (see page 252) *pSld, WORD newPage)

8.1.21.5 SldGetPage Macro

File

Slider.h (see page 886)

C

```
#define SldGetPage(pSld) (((SLIDER*)pSld)->page)
```

Module

Slider/Scroll Bar (see page 237)

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 returns the current slider thumb position. Value is always in the 1 to range/2 inclusive.

Syntax

SldGetPage(pSld)

8.1.21.6 SldSetPos Function

File

Slider.h (see page 886)

C

```
void SldSetPos(
    SLIDER * pSld,
    SHORT newPos
);
```

Module

Slider/Scroll Bar (see page 237)

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 (see page 252) *pSld, SHORT newPos)

8.1.21.7 SldGetPos Macro

File

Slider.h (see page 886)

C

```
#define SldGetPos(pSld) (((SLIDER*)pSld)->pos)
```

Module

Slider/Scroll Bar (see page 237)

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)

8.1.21.8 SldSetRange Function

File

Slider.h (see page 886)

C

```
void SldSetRange(
    SLIDER * pSld,
    SHORT newRange
);
```

Module

Slider/Scroll Bar (see page 237)

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 (see page 252) *pSld, SHORT newRange)

8.1.21.9 SldGetRange Macro

File

Slider.h (see page 886)

C

```
#define SldGetRange(pSld) (((SLIDER*)pSld)->range)
```

Module

Slider/Scroll Bar (see page 237)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

Returns the current value of the slider range.

Preconditions

none

Example

```
WORD range;
SLIDER *pSld;

    range = SldGetRange(pSld);
```

Overview

Returns the current slider page. Value is always in the 1-range/2 inclusive for both slider and scrollbar type.

Syntax

SldGetRange(pSld)

8.1.21.10 SldIncPos Macro

File

Slider.h (see page 886)

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 (see page 237)

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)

8.1.21.11 SldDecPos Macro

File

Slider.h (see page 886)

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 (see page 237)

Input Parameters

Input Parameters	Description
pSld	Pointer to the object.

Side Effects

none

Returns

none

Preconditions

none

Example

See SldIncPos (see page 248)() 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)

8.1.21.12 SldMsgDefault Function

File

Slider.h (see page 886)

C

```
void SldMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Slider/Scroll Bar (see page 237)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL (see page 48) 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 (see page 75)() 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 Screen, Keyboard	Set SLD_DRAW_THUMB (see page 240)	Slider (see page 237) will be redrawn with thumb in the incremented position.
SLD_MSG_DEC	Touch Screen, Keyboard	Set SLD_DRAW_THUMB (see page 240)	Slider (see page 237) will be redrawn with thumb in the decremented position.

Syntax

void SldMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.21.13 SldTranslateMsg Function

File

Slider.h (see page 886)

C

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

Module

Slider/Scroll Bar (see page 237)

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 (see page 48) message:

- SLD_MSG_INC – Increment slider position
- SLD_MSG_DEC – Decrement slider position
- OBJ_MSG_PASSIVE – Slider (see page 237) is not affected
- OBJ_MSG_INVALID – Slider (see page 237) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 326) or SCAN_LEFT_PRESSED (see page 323).

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 (see page 322) or SCAN_RIGHT_PRESSED (see page 325).
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 (see page 316) *pMsg)

8.1.21.14 SLIDER Structure

File

Slider.h (see page 886)

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 (see page 52)).
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 (see page 237)

Overview

Defines the parameters required for a slider/scrollbar Object. Depending on the SLD_SCROLLBAR (see page 241) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (see page 241) is set, mode is scrollbar; if not set mode is slider. For scrollbar mode, focus rectangle is not drawn.

8.1.22 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 (see page 255)	This function creates a STATICTEXT (see page 259) 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 (see page 256)	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 (see page 258)	This function sets the string that will be used for the object.
	StTranslateMsg (see page 258)	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 (see page 257)	This macro returns the address of the current text string used for the object.

Structures

Name	Description
STATICTEXT (see page 259)	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.

8.1.22.1 Static Text States

Macros

Name	Description
ST_CENTER_ALIGN (see page 254)	Bit to indicate text is center aligned.
ST_DISABLED (see page 254)	Bit for disabled state.
ST_DRAW (see page 254)	Bit to indicate static text must be redrawn.
ST_FRAME (see page 255)	Bit to indicate frame is displayed.
ST_HIDE (see page 255)	Bit to remove object from screen.
ST_RIGHT_ALIGN (see page 255)	Bit to indicate text is left aligned.
ST_UPDATE (see page 255)	Bit to indicate that text area only is redrawn.

Module

Static Text (see page 252)

Description

List of Static Text bit states.

8.1.22.1.1 ST_CENTER_ALIGN Macro

File

StaticText.h (see page 900)

C

```
#define ST_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned.
```

Description

Bit to indicate text is center aligned.

8.1.22.1.2 ST_DISABLED Macro

File

StaticText.h (see page 900)

C

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

Description

Bit for disabled state.

8.1.22.1.3 ST_DRAW Macro

File

StaticText.h (see page 900)

C

```
#define ST_DRAW 0x4000  // Bit to indicate static text must be redrawn.
```

Description

Bit to indicate static text must be redrawn.

8.1.22.1.4 ST_FRAME Macro

File

StaticText.h (see page 900)

C

```
#define ST_FRAME 0x0010  // Bit to indicate frame is displayed.
```

Description

Bit to indicate frame is displayed.

8.1.22.1.5 ST_HIDE Macro

File

StaticText.h (see page 900)

C

```
#define ST_HIDE 0x8000  // Bit to remove object from screen.
```

Description

Bit to remove object from screen.

8.1.22.1.6 ST_RIGHT_ALIGN Macro

File

StaticText.h (see page 900)

C

```
#define ST_RIGHT_ALIGN 0x0004  // Bit to indicate text is left aligned.
```

Description

Bit to indicate text is left aligned.

8.1.22.1.7 ST_UPDATE Macro

File

StaticText.h (see page 900)

C

```
#define ST_UPDATE 0x2000  // Bit to indicate that text area only is redrawn.
```

Description

Bit to indicate that text area only is redrawn.

8.1.22.2 StCreate Function

File

StaticText.h (see page 900)

C

```
STATICTEXT * StCreate(
    WORD ID,
    SHORT left,
    SHORT top,
```

```
    SHORT right,
    SHORT bottom,
    WORD state,
    XCHAR * pText,
    GOL_SCHEME * pScheme
);
```

Module

Static Text (see page 252)

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 (see page 259) 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 (see page 259) *StCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state , XCHAR (see page 345) *pText, GOL_SCHEME (see page 331) *pScheme)

8.1.22.3 StDraw Function

File

StaticText.h (see page 900)

C

```
WORD StDraw(
```

```
    void * pObj
);
```

Module

Static Text (see page 252)

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 (see page 255)() 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)

8.1.22.4 StGetText Macro

File

StaticText.h (see page 900)

C

```
#define StGetText(pSt) pSt->pText
```

Module

Static Text (see page 252)

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)

8.1.22.5 StSetText Function

File

StaticText.h (see page 900)

C

```
void StSetText(
    STATICTEXT * pSt,
    XCHAR * pText
);
```

Module

Static Text (see page 252)

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 (see page 259) *pSt, XCHAR (see page 345) *pText)

8.1.22.6 StTranslateMsg Function

File

StaticText.h (see page 900)

C

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

Module

Static Text (see page 252)

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 (see page 48) message:

- ST_MSG_SELECTED – Static Text is selected
- OBJ_MSG_INVALID – Static Text is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 316) *pMsg)

8.1.22.7 STATICTEXT Structure

File

StaticText.h (see page 900)

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 (see page 52)).
SHORT textHeight;	Pre-computed text height.

XCHAR * pText;	The pointer to text used.

Module

Static Text (see page 252)

Overview

Defines the parameters required for a Static Text Object.

8.1.23 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 (see page 265)	This function creates a TEXTENTRY (see page 276) 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 (see page 266)	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 (see page 308)(). The number of keys will depend on the horizontal and vertical parameters given (horizontalKeys*verticakKeys).
	TeSetBuffer (see page 267)	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 (see page 268)	This function will clear the data in the display. You must set the drawing state bit TE_UPDATE_TEXT (see page 264) to update the TEXTENTRY (see page 276) on the screen.
	TeGetKeyCommand (see page 268)	This function will return the currently used command by a key with the given index.
	TeSetKeyCommand (see page 269)	This function will assign a command (TE_DELETE_COM (see page 264), TE_SPACE_COM (see page 265) or TE_ENTER_COM (see page 264)) to a key with the given index.
	TeCreateKeyMembers (see page 270)	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 (see page 270)	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 (see page 264) or TE_DRAW (see page 263) must be set to see the effect of this insertion.
	TeIsKeyPressed (see page 271)	This function will test if a key given by its index in the TextEntry object has been pressed.

	TeSpaceChar (see page 272)	This function will insert a space character to the end of the buffer. Drawing states TE_UPDATE_TEXT (see page 264) or TE_DRAW (see page 263) must be set to see the effect of this insertion.
	TeDelKeyMembers (see page 272)	This function will delete the KEYMEMBER (see page 277) list assigned to the object from memory. Pointer to the KEYMEMBER (see page 277) list is then initialized to NULL.
	TeSetKeyText (see page 273)	This function will set the test assigned to a key with the given index.
	TeMsgDefault (see page 273)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	TeTranslateMsg (see page 275)	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 (see page 267)	This macro will return the currently used buffer in the TextEntry object.

Structures

Name	Description
TEXTENTRY (see page 276)	Defines the parameters required for a TextEntry Object.
KEYMEMBER (see page 277)	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.

8.1.23.1 TextEntry States

Macros

Name	Description
TE_KEY_PRESSED (see page 262)	Bit for press state of one of the keys.
TE_DISABLED (see page 263)	Bit for disabled state.
TE_ECHO_HIDE (see page 263)	Bit to hide the entered characters and instead echo "*" characters to the display.
TE_DRAW (see page 263)	Bit to indicate object must be redrawn.
TE_HIDE (see page 263)	Bit to indicate object must be removed from screen.
TE_UPDATE_KEY (see page 263)	Bit to indicate redraw of a key is needed.
TE_UPDATE_TEXT (see page 264)	Bit to indicate redraw of the text displayed is needed.

Module

Text Entry (see page 260)

Description

List of Text Entry (see page 260) bit states.

8.1.23.1.1 TE_KEY_PRESSED Macro

File

TextEntry.h (see page 919)

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.

8.1.23.1.2 TE_DISABLED Macro

File

TextEntry.h (see page 919)

C

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

Description

Bit for disabled state.

8.1.23.1.3 TE_ECHO_HIDE Macro

File

TextEntry.h (see page 919)

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.

8.1.23.1.4 TE_DRAW Macro

File

TextEntry.h (see page 919)

C

```
#define TE_DRAW 0x4000  // Bit to indicate object must be redrawn.
```

Description

Bit to indicate object must be redrawn.

8.1.23.1.5 TE_HIDE Macro

File

TextEntry.h (see page 919)

C

```
#define TE_HIDE 0x8000  // Bit to indicate object must be removed from screen.
```

Description

Bit to indicate object must be removed from screen.

8.1.23.1.6 TE_UPDATE_KEY Macro

File

TextEntry.h (see page 919)

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.

8.1.23.1.7 TE_UPDATE_TEXT Macro

File

TextEntry.h (see page 919)

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.

8.1.23.2 Key Command Types

Macros

Name	Description
TE_DELETE_COM (see page 264)	This macro is used to assign a "delete" command on a key.
TE_ENTER_COM (see page 264)	This macro is used to assign an "enter" (carriage return) command on a key.
TE_SPACE_COM (see page 265)	This macro is used to assign an insert "space" command on a key.

Module

Text Entry (see page 260)

Description

List of available Key command types.

8.1.23.2.1 TE_DELETE_COM Macro

File

TextEntry.h (see page 919)

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.

8.1.23.2.2 TE_ENTER_COM Macro

File

TextEntry.h (see page 919)

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.

8.1.23.2.3 TE_SPACE_COM Macro

File

TextEntry.h (see page 919)

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.

8.1.23.3 TeCreate Function

File

TextEntry.h (see page 919)

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 (see page 260)

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.
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 CUSTOMEKEYS must be populated before calling this function.

Overview

This function creates a TEXTENTRY (see page 276) 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 (see page 276) *TeCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, SHORT horizontalKeys, SHORT verticalKeys, XCHAR (see page 345) *pText[], void *pBuffer, WORD bufferLength,void *pDisplayFont, GOL_SCHEME (see page 331) *pScheme)

8.1.23.4 TeDraw Function

File

TextEntry.h (see page 919)

C

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

Module

Text Entry (see page 260)

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 (see page 308)(). The number of keys will depend on the horizontal and vertical parameters given (horizontalKeys*verticakKeys).

Syntax

WORD TeDraw(void *pObj)

8.1.23.5 TeGetBuffer Macro

File

TextEntry.h (see page 919)

C

```
#define TeGetBuffer(pTe) (((TEXTENTRY *)pTe)->pTeOutput)
```

Module

Text Entry (see page 260)

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)

8.1.23.6 TeSetBuffer Function

File

TextEntry.h (see page 919)

C

```
void TeSetBuffer(
    TEXTENTRY * pTe,
    XCHAR * pText,
    WORD MaxSize
);
```

Module

Text Entry (see page 260)

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 (see page 276) *pTe, XCHAR (see page 345) *pText, WORD MaxSize)

8.1.23.7 TeClearBuffer Function

File

TextEntry.h (see page 919)

C

```
void TeClearBuffer(
    TEXTENTRY * pTe
);
```

Module

Text Entry (see page 260)

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 (see page 264) to update the TEXTENTRY (see page 276) on the screen.

Syntax

void TeClearBuffer (TEXTENTRY (see page 276) *pTe)

8.1.23.8 TeGetKeyCommand Function

File

TextEntry.h (see page 919)

C

```
WORD TeGetKeyCommand(
    TEXTENTRY * pTe,
```

```
    WORD index
);
```

Module

Text Entry (see page 260)

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 (see page 264) 0x02 - TE_SPACE_COM (see page 265) 0x03 - TE_ENTER_COM (see page 264)

Preconditions

none

Overview

This function will return the currently used command by a key with the given index.

Syntax

TeGetKeyCommand(pTe, index)

8.1.23.9 TeSetKeyCommand Function

File

TextEntry.h (see page 919)

C

```
BOOL TeSetKeyCommand(
    TEXTENTRY * pTe,
    WORD index,
    WORD command
);
```

Module

Text Entry (see page 260)

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 (see page 264), TE_SPACE_COM (see page 265) or TE_ENTER_COM (see page 264)) to a key with the given index.

Syntax

void TeSetKeyCommand(TEXTENTRY (see page 276) *pTe,WORD index,WORD command)

8.1.23.10 TeCreateKeyMembers Function

File

TextEntry.h (see page 919)

C

```
KEYMEMBER * TeCreateKeyMembers(
    TEXTENTRY * pTe,
    XCHAR * pText[]
);
```

Module

Text Entry (see page 260)

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 (see page 277) list. A NULL is returned if the list is not created succesfully.

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 (see page 277) *TeCreateKeyMembers(TEXTENTRY (see page 276) *pTe,XCHAR (see page 345) *pText[])

8.1.23.11 TeAddChar Function

File

TextEntry.h (see page 919)

C

```
void TeAddChar(
    TEXTENTRY * pTe
);
```

Module

Text Entry (see page 260)

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 (see page 264) or TE_DRAW (see page 263) must be set to see the effect of this insertion.

Syntax

void TeAddChar(TEXTENTRY (see page 276) *pTe)

8.1.23.12 TeIsKeyPressed Function

File

TextEntry.h (see page 919)

C

```
BOOL TeIsKeyPressed(
    TEXTENTRY * pTe,
    WORD index
);
```

Module

Text Entry (see page 260)

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 TeIsKeyPressed(TEXTENTRY (see page 276) *pTe, WORD index)

8.1.23.13 TeSpaceChar Function

File

TextEntry.h (see page 919)

C

```
void TeSpaceChar(
    TEXTENTRY * pTe
);
```

Module

Text Entry (see page 260)

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 (see page 264) or TE_DRAW (see page 263) must be set to see the effect of this insertion.

Syntax

void TeSpaceChar(TEXTENTRY (see page 276) *pTe)

8.1.23.14 TeDelKeyMembers Function

File

TextEntry.h (see page 919)

C

```
void TeDelKeyMembers(
    void * pObj
);
```

Module

Text Entry (see page 260)

Input Parameters

Input Parameters	Description
pTe	pointer to the object

Side Effects

none.

Returns

none.

Preconditions

none

Overview

This function will delete the KEYMEMBER (see page 277) list assigned to the object from memory. Pointer to the KEYMEMBER (see page 277) list is then initialized to NULL.

Syntax

void TeDelKeyMembers(void *pObj)

8.1.23.15 TeSetKeyText Function

File

TextEntry.h (see page 919)

C

```
BOOL TeSetKeyText(
    TEXTENTRY * pTe,
    WORD index,
    XCHAR * pText
);
```

Module

Text Entry (see page 260)

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 (see page 276) *pTe, WORD index, XCHAR (see page 345) *pText)

8.1.23.16 TeMsgDefault Function

File

TextEntry.h (see page 919)

C

```
void TeMsgDefault(
    WORD translatedMsg,
    void * pObj,
    GOL_MSG * pMsg
);
```

Module

Text Entry (see page 260)

Input Parameters

Input Parameters	Description
WORD translatedMsg	The translated message.
GOL_MSG * pMsg	The pointer to the GOL (see page 48) message.
pTe	The pointer to the object whose state will be modified.

Side Effects

none

Returns

none

Preconditions

none

Example

See BtnTranslateMsg (see page 75)() 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 (see page 264), TE_UPDATE_KEY (see page 263),	Add a character in the buffer and update the text displayed.
		Clear TE_KEY_PRESSED (see page 262)	
TE_MSG_SPACE	Touch Screen,	Set TE_UPDATE_TEXT (see page 264), TE_UPDATE_KEY (see page 263),	Insert a space character in the buffer and update the text displayed.
		Clear TE_KEY_PRESSED (see page 262)	
TE_MSG_DELETE	Touch Screen,	Set TE_UPDATE_TEXT (see page 264), TE_UPDATE_KEY (see page 263),	Delete the most recent character in the buffer and update the text displayed.
		Clear TE_KEY_PRESSED (see page 262)	
TE_MSG_ENTER	Touch Screen,	Set TE_UPDATE_TEXT (see page 264), TE_UPDATE_KEY (see page 263),	User can define the use of this event in the message callback. Object will just update the key.
		Clear TE_KEY_PRESSED (see page 262)	
TE_MSG_RELEASED	Touch Screen,	Clear TE_KEY_PRESSED (see page 262)	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 (see page 262) TE_UPDATE_KEY (see page 263)	A Key in the object will be redrawn in the pressed state.

Syntax

TeMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG (see page 316)* pMsg)

8.1.23.17 TeTranslateMsg Function

File

TextEntry.h (see page 919)

C

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

Module

Text Entry (see page 260)

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 (see page 48) 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 (see page 316) *pMsg)

8.1.23.18 TEXTENTRY Structure

File

TextEntry.h (see page 919)

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 (see page 52)).
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 (see page 277). This is only used by the Widget. User must not change
KEYMEMBER * pHeadOfList;	Pointer to head of the list
void * pDisplayFont;	Pointer to the font used in displaying the text.

Module

Text Entry (see page 260)

Overview

Defines the parameters required for a TextEntry Object.

8.1.23.19 KEYMEMBER Structure

File

TextEntry.h (see page 919)

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 (see page 262)) 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 (see page 264), TE_SPACE_COM (see page 265) or TE_ENTER_COM (see page 264).
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 (see page 260)

Overview

Defines the parameters and the strings assigned for each key.

8.1.24 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 (see page 352). It also has additional controls for both Title Bar (see page 352) and Client Area.

Functions

	Name	Description
	WndCreate (see page 281)	This function creates a WINDOW (see page 285) 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 (see page 282)	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 (see page 283)	This function sets the string used for the title bar.
	WndTranslateMsg (see page 284)	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 (see page 283)	This macro returns the address of the current text string used for the title bar.

Structures

Name	Description
WINDOW (see page 285)	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.

8.1.24.1 Window States

Macros

Name	Description
WND_DISABLED (see page 279)	Bit for disabled state
WND_DRAW (see page 280)	Bits to indicate whole window must be redrawn
WND_DRAW_CLIENT (see page 280)	Bit to indicate client area must be redrawn
WND_DRAW_TITLE (see page 280)	Bit to indicate title area must be redrawn
WND_FOCUSED (see page 280)	Bit for focus state
WND_HIDE (see page 280)	Bit to indicate window must be removed from screen
WND_TITLECENTER (see page 281)	Bit to center the text on the Title Area

Module

Window (see page 278)

Description

List of Window (see page 278) bit states.

8.1.24.1.1 WND_DISABLED Macro

File

Window.h (see page 937)

C

```
#define WND_DISABLED 0x0002  // Bit for disabled state
```

Description

Bit for disabled state

8.1.24.1.2 WND_DRAW Macro

File

Window.h (see page 937)

C

```
#define WND_DRAW 0x6000  // Bits to indicate whole window must be redrawn
```

Description

Bits to indicate whole window must be redrawn

8.1.24.1.3 WND_DRAW_CLIENT Macro

File

Window.h (see page 937)

C

```
#define WND_DRAW_CLIENT 0x4000  // Bit to indicate client area must be redrawn
```

Description

Bit to indicate client area must be redrawn

8.1.24.1.4 WND_DRAW_TITLE Macro

File

Window.h (see page 937)

C

```
#define WND_DRAW_TITLE 0x2000  // Bit to indicate title area must be redrawn
```

Description

Bit to indicate title area must be redrawn

8.1.24.1.5 WND_FOCUSED Macro

File

Window.h (see page 937)

C

```
#define WND_FOCUSED 0x0001  // Bit for focus state
```

Description

Bit for focus state

8.1.24.1.6 WND_HIDE Macro

File

Window.h (see page 937)

C

```
#define WND_HIDE 0x8000  // Bit to indicate window must be removed from screen
```

Description

Bit to indicate window must be removed from screen

8.1.24.1.7 WND_TITLECENTER Macro

File

Window.h (see page 937)

C

```
#define WND_TITLECENTER 0x0004  // Bit to center the text on the Title Area
```

Description

Bit to center the text on the Title Area

8.1.24.2 WndCreate Function

File

Window.h (see page 937)

C

```
WINDOW * WndCreate(
    WORD ID,
    SHORT left,
    SHORT top,
    SHORT right,
    SHORT bottom,
    WORD state,
    void * pBitmap,
    XCHAR * pText,
    GOL_SCHEME * pScheme
);
```

Module

Window (see page 278)

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 (see page 285) 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 (see page 285) *WndCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, void* pBitmap, XCHAR (see page 345)* pText, GOL_SCHEME (see page 331) *pScheme)

8.1.24.3 WndDraw Function

File

Window.h (see page 937)

C

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

Module

Window (see page 278)

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 (see page 281)() 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)

8.1.24.4 WndGetText Macro

File

Window.h (see page 937)

C

```
#define WndGetText(pW) pW->pText
```

Module

Window (see page 278)

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)

8.1.24.5 WndSetText Function

File

Window.h (see page 937)

C

```
void WndSetText(
    WINDOW * pW,
    XCHAR * pText
);
```

Module

Window (see page 278)

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 (see page 283)() example.

Overview

This function sets the string used for the title bar.

Syntax

WndSetText(WINDOW (see page 285) *pW, XCHAR (see page 345) *pText)

8.1.24.6 WndTranslateMsg Function

File

Window.h (see page 937)

C

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

Module

Window (see page 278)

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 (see page 48) message:

- WND_MSG_TITLE – Title area is selected
- WND_MSG_CLIENT – Client area is selected
- OBJ_MSG_INVALID – Window (see page 278) is not affected

Preconditions

none

Example

Usage is similar to BtnTranslateMsg (see page 75)() 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 (see page 316) *pMsg)

8.1.24.7 WINDOW Structure

File

Window.h (see page 937)

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 (see page 52)).
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 (see page 278)

Overview

The structure contains data for the window

8.2 Object States

Objects rendered on the display are based on their current Property States and the Drawing States.

Macros

Name	Description
GetState (see page 288)	This macro retrieves the current value of the state bits of an object. It is possible to get several state bits.
ClrState (see page 289)	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 (see page 289)	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.

Module

Graphics Object Layer (see page 48)

Description

The GOL (see page 48) 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 (see page 52) 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 (see page 48) object.
- P – Property states, D – Drawing states

Individual Object drawing function (e.g. BtnDraw (see page 70)(), SldDraw (see page 243)(), etc...) does not reset the draw states instead use GOLDraw (see page 296)() 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.

8.2.1 Common Object States

Macros

Name	Description
FOCUSED (see page 287)	Focus state bit.
DISABLED (see page 287)	Disabled state bit.
HIDE (see page 287)	Object hide state bit. Object will be hidden from the screen by drawing over it the common background color.
DRAW (see page 287)	Object redraw state bit. The whole Object must be redrawn.
DRAW_FOCUS (see page 288)	Focus redraw state bit. The focus rectangle must be redrawn.
DRAW_UPDATE (see page 288)	Partial Object redraw state bit. A part or parts of of the Object must be redrawn to show updated state.

Description

List of common Object bit states.

8.2.1.1 FOCUSED Macro

File

GOL.h (see page 552)

C

```
#define FOCUSED 0x0001
```

Description

Focus state bit.

8.2.1.2 DISABLED Macro

File

GOL.h (see page 552)

C

```
#define DISABLED 0x0002
```

Description

Disabled state bit.

8.2.1.3 HIDE Macro

File

GOL.h (see page 552)

C

```
#define HIDE 0x8000
```

Description

Object hide state bit. Object will be hidden from the screen by drawing over it the common background color.

8.2.1.4 DRAW Macro

File

GOL.h (see page 552)

C

```
#define DRAW 0x7C00
```

Description

Object redraw state bit. The whole Object must be redrawn.

8.2.1.5 DRAW_FOCUS Macro

File

GOL.h (see page 552)

C

```
#define DRAW_FOCUS 0x2000
```

Description

Focus redraw state bit. The focus rectangle must be redrawn.

8.2.1.6 DRAW_UPDATE Macro

File

GOL.h (see page 552)

C

```
#define DRAW_UPDATE 0x3C00
```

Description

Partial Object redraw state bit. A part or parts of of the Object must be redrawn to show updated state.

8.2.2 GetState Macro

File

GOL.h (see page 552)

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)

8.2.3 ClrState Macro

File

GOL.h (see page 552)

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 (see page 289)().

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)

8.2.4 SetState Macro

File

GOL.h (see page 552)

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)

8.3 Object Management

Functions

	Name	Description
	GOLAddObject (see page 292)	This function adds an object to the tail of the active list pointed to by _pGolObjects (see page 334). The new list tail is set to point to NULL.
	GOLFindObject (see page 293)	This function finds an object in the active list pointed to by _pGolObjects (see page 334) using the given object ID.
	GOLRedrawRec (see page 295)	This function marks all objects in the active list intersected by the given rectangular area to be redrawn.
	GOLDraw (see page 296)	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 (see page 297)() function is called by GOLDraw() when drawing of objects in the active list is completed.

	GOLDrawCallback (see page 297)	GOLDrawCallback() function MUST BE implemented by the user. This is called inside the GOLDraw (see page 296)() 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 (see page 48) if this function returns a zero. To pass drawing control to GOL (see page 48) this function must return a non-zero value. If GOL (see page 48) messaging is not using the active link list, it is safe to modify the list here.
	GOLFree (see page 298)	This function frees all the memory used by objects in the active list and initializes _pGolObjects (see page 334) pointer to NULL to start a new empty list. This function must be called only inside the GOLDrawCallback (see page 297)()function when using GOLDraw (see page 296)() and GOLMsg (see page 314)() functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.
	GOLDeleteObject (see page 301)	deletes an object to the linked list objects for the current screen.
	GOLDeleteObjectByID (see page 301)	Deletes an object in the current active linked list of objects using the ID parameter of the object.
	GOLSetFocus (see page 304)	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 (see page 287) state for the object was in focus previously, set FOCUSED (see page 287) state for the required object and marks the objects to be redrawn.
	GOLInit (see page 306)	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 (see page 48) functions can be used. It is not necessary to call GraphInit() function if this function is used.
	GOLCanBeFocused (see page 307)	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 (see page 307)	This function returns the pointer to the next object in the active linked list which is able to receive keyboard input.
	GOLGetFocusPrev (see page 308)	This function returns the pointer to the previous object in the active linked list which is able to receive keyboard input.
	GOLPanelDrawTsk (see page 310)	This function draws a panel on the screen with parameters set by GOLPanelDraw (see page 308)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.
	GOLTwoTonePanelDrawTsk (see page 310)	This function draws a two tone panel on the screen with parameters set by GOLPanelDraw (see page 308)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Macros

Name	Description
GOLRedraw (see page 294)	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 (see page 296)	This macro resets the drawing states of the object (6 MSBits of the object's state).
GetObjType (see page 299)	This macro returns the object type.
GetObjID (see page 299)	This macro returns the object ID.
GetObjNext (see page 300)	This macro returns the next object after the specified object.

GOLNewList (see page 302)	This macro starts a new linked list of objects and resets the keyboard focus to none. This macro assigns the current active list _pGolObjects (see page 334) and current receiving keyboard input _pObjectFocused (see page 335) 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 (see page 298)() function.
GOLGetList (see page 303)	This macro gets the current active list.
GOLSetList (see page 303)	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 (see page 335) pointer to NULL. If the new active list has an object's state set to focus, the _pObjectFocused (see page 335) 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 (see page 305)	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 (see page 306)	This macro returns the pointer to the current object receiving keyboard input.
GOLPanelDraw (see page 308)	This macro sets the parameters to draw a panel. Panel is not an object. It will not be added to the active list. Use GOLPanelDrawTsk (see page 310)() to display the panel on the screen. The following relationships of the parameters determines the general shape of the button: 1. Panel width is determined by right - left. 2. Panel height is determined by top - bottom. 3. Panel radius - specifies if the panel will have a rounded edge. If zero then the panel will have sharp (cornered) edge. 4. If 2*radius = height = width, the panel is circular.

Module

Graphics Object Layer (see page 48)

Description

This section describes the API functions and macros that are used to create, maintain and render individual and list of objects.

8.3.1 GOLAddObject Function

File

GOL.h (see page 552)

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 (see page 334). The new list tail is set to point to NULL.

Syntax

void GOLAddObject(OBJ_HEADER (see page 52)* object)

8.3.2 GOLFindObject Function

File

GOL.h (see page 552)

C

```
OBJ_HEADER * GOLFindObject(
    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 = GOLFindObject(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 (see page 334) using the given object ID.

Syntax

OBJ_HEADER (see page 52)* GOLFindObject(WORD ID)

8.3.3 GOLRedraw Macro

File

GOL.h (see page 552)

C

```
#define GOLRedraw(pObj) ((OBJ_HEADER *)pObj)->state |= 0x7c00;
```

Input Parameters

Input Parameters	Description
pObj	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)

8.3.4 GOLRedrawRec Function

File

GOL.h (see page 552)

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)

8.3.5 GOLDraw Function

File

GOL.h (see page 552)

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 (see page 297)() function is called by GOLDraw() when drawing of objects in the active list is completed.

Syntax

WORD GOLDraw()

8.3.6 GOLDrawComplete Macro

File

GOL.h (see page 552)

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)

8.3.7 GOLDrawCallback Function

File

GOL.h (see page 552)

C

```
WORD GOLDrawCallback();
```

Side Effects

none

Returns

Return a one if GOLDraw (see page 296)() 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 (see page 296)() 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 (see page 48) if this function returns a zero. To pass drawing control to GOL (see page 48) this function must return a non-zero value. If GOL (see page 48) messaging is not using the active link list, it is safe to modify the list here.

Syntax

WORD GOLDrawCallback()

8.3.8 GOLFree Function

File

GOL.h (see page 552)

C

```
void GOLFree();
```

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
    GolFree();                          // 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 (see page 334) pointer to NULL to start a new empty list. This function must be called only inside the GOLDrawCallback (see page 297)()function when using GOLDraw (see page 296)() and GOLMsg (see page 314)() functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.

Syntax

void GOLFree()

8.3.9 GetObjType Macro

File

GOL.h (see page 552)

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)

8.3.10 GetObjID Macro

File

GOL.h (see page 552)

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)

8.3.11 GetObjNext Macro

File

GOL.h (see page 552)

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)

8.3.12 GOLDeleteObject Function

File

GOL.h (see page 552)

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 (see page 52) * object)

8.3.13 GOLDeleteObjectByID Function

File

GOL.h (see page 552)

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)

8.3.14 GOLNewList Macro

File

GOL.h (see page 552)

C

```
#define GOLNewList \
    _pGolObjects = NULL; \
    _pObjectFocused = NULL
```

Side Effects

This macro sets the focused object pointer (_pObjectFocused (see page 335)) 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 (see page 334) and current receiving keyboard input _pObjectFocused (see page 335) 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 (see page 298)() function.

Syntax

void GOLNewList()

8.3.15 GOLGetList Macro

File

GOL.h (see page 552)

C

```
#define GOLGetList _pGolObjects
```

Side Effects

none

Returns

Returns the pointer to the current active list.

Preconditions

none

Example

See GOLNewList (see page 302)() example.

Overview

This macro gets the current active list.

Syntax

GOLGetList()

8.3.16 GOLSetList Macro

File

GOL.h (see page 552)

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 (see page 335)) to NULL. Previous active list should be saved if needed to be referenced later. If not, use GOLFree (see page 298)() 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 (see page 335) pointer to NULL. If the new active list has an object's state set to focus, the _pObjectFocused (see page 335) 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)

8.3.17 GOLSetFocus Function

File

GOL.h (see page 552)

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 (see page 287) state for the object was in focus previously, set FOCUSED (see page 287) state for the required object and marks the objects to be redrawn.

Syntax

void GOLSetFocus(OBJ_HEADER (see page 52)* object)

8.3.18 IsObjUpdated Macro

File

GOL.h (see page 552)

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)

8.3.19 GOLInit Function

File

GOL.h (see page 552)

C

```
void GOLInit();
```

Side Effects

This sets the line type to SOLID_LINE (see page 351), sets the screen to all BLACK (see page 389), sets the current drawing color to WHITE (see page 394), 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 (see page 48) functions can be used. It is not necessary to call GraphInit() function if this function is used.

Syntax

void GOLInit()

8.3.20 GOLGetFocus Macro

File

GOL.h (see page 552)

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()

8.3.21 GOLCanBeFocused Function

File

GOL.h (see page 552)

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 (see page 52)* object)

8.3.22 GOLGetFocusNext Function

File

GOL.h (see page 552)

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 (see page 52) *GOLGetFocusNext()

8.3.23 GOLGetFocusPrev Function

File

GOL.h (see page 552)

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 (see page 52) *GOLGetFocusNext (see page 307)()

8.3.24 GOLPanelDraw Macro

File

GOL.h (see page 552)

C

```
#define GOLPanelDraw(left, top, right, bottom, radius, faceClr, embossLtClr, embossDkClr,
pBitmap, embossSize) \
    _rpnlX1 =
left;                                                                    \
    _rpnlY1 =
top;                                                                     \
    _rpnlX2 =
right;                                                                   \
    _rpnlY2 =
bottom;                                                                  \
    _rpnlR =
radius;                                                                   \
    _rpnlFaceColor =
faceClr;                                                            \
    _rpnlEmbossLtColor =
embossLtClr;                                                     \
    _rpnlEmbossDkColor =
embossDkClr;                                                     \
    _pRpnlBitmap =
pBitmap;                                                                 \
    _rpnlEmbossSize = embossSize;
```

Input Parameters

Input Parameters	Description
left	Panel's left most position.
top	Panel's top most position.
right	Panel's right most position.
bottom	Panel's bottom most position.
radius	The radius of the rounded corner of the panel.
faceClr	Defines the face color of the panel.
embossLtClr	Defines the emboss light color.
embossDkClr	Defines the emboss dark color.
pBitmap	Defines the bitmap used in the face of the panel.
embossSize	Defines the emboss size of the panel.

Side Effects

none

Returns

none

Preconditions

none

Example

```
// Dimensions for signature box
#define SIG_PANEL_LEFT      10
#define SIG_PANEL_RIGHT    310
#define SIG_PANEL_TOP       50
#define SIG_PANEL_BOTTOM   170

#define SIG_STATE_SET   0
#define SIG_STATE_DRAW  1

GOL_SCHEME *altScheme;           // style scheme

// Draws box for signature
WORD PanelSignature() {
    static BYTE state = SIG_STATE_SET;

    if(state == SIG_STATE_SET){
        // set data for panel drawing (radius = 0)
        // assume altScheme is defined
        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; // change state
    }

    if(!GOLPanelDrawTsk())
        return 0; // drawing is not completed
    } else {
        state = SIG_STATE_SET; // set state to initial
        return 1; // drawing is done
    }
}
```

Overview

This macro sets the parameters to draw a panel. Panel is not an object. It will not be added to the active list. Use GOLPanelDrawTsk (see page 310)() to display the panel on the screen. The following relationships of the parameters

determines the general shape of the button:

1. Panel width is determined by right - left.
2. Panel height is determined by top - bottom.
3. Panel radius - specifies if the panel will have a rounded edge. If zero then the panel will have sharp (cornered) edge.
4. If 2*radius = height = width, the panel is circular.

Syntax

GOLPanelDraw(left, top, right, bottom, radius, faceClr, embossLtClr, embossDkClr, pBitmap, embossSize)

8.3.25 GOLPanelDrawTsk Function

File

GOL.h (see page 552)

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 (see page 308)() macro.

Example

See GOLPanelDraw (see page 308)() example.

Overview

This function draws a panel on the screen with parameters set by GOLPanelDraw (see page 308)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Syntax

WORD GOLPanelDrawTsk()

8.3.26 GOLTwoTonePanelDrawTsk Function

File

GOL.h (see page 552)

C

```
WORD GOLTwoTonePanelDrawTsk();
```

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 (see page 308)() macro.

Example

Usage is similar to GOLPanelDraw (see page 308)() example.

Overview

This function draws a two tone panel on the screen with parameters set by GOLPanelDraw (see page 308)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Syntax

WORD GOLTwoTonePanelDrawTsk()

8.4 GOL Messages

The library provides an interface to accept messages from the input devices.

Enumerations

Name	Description
TRANS_MSG (see page 317)	This structure defines the list of translated messages for GOL (see page 48) Objects used in the library.
INPUT_DEVICE_EVENT (see page 319)	This structure defines the types of GOL (see page 48) message events used in the library.
INPUT_DEVICE_TYPE (see page 319)	This structure defines the types of input devices used in the library.

Functions

	Name	Description
	GOLMsg (see page 314)	This function receives a GOL (see page 48) 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 (see page 314)() 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 (see page 48) Messages section for deatils. This function should be called when GOL (see page 48) drawing is completed. It can be done... more (see page 314)
	GOLMsgCallback (see page 314)	The user MUST implement this function. GOLMsg (see page 314)() 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 (see page 48) will not perform any action.

Module

Graphics Object Layer (see page 48)

Structures

Name	Description
GOL_MSG (see page 316)	This structure defines the GOL (see page 48) message used in the library. • The types must be one of the INPUT_DEVICE_TYPE (see page 319): • TYPE_UNKNOWN • TYPE_KEYBOARD • TYPE_TOUCHSCREEN • TYPE_MOUSE • uiEvent must be one of the INPUT_DEVICE_EVENT (see page 319). • 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 (see page 48) using the GOL (see page 48) 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 (see page 314)() 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 (see page 48) message (the GOL_MSG (see page 316) structure).
2. A loop evaluates which object is affected by the message. This is done inside GOLMsg (see page 314)() function.
3. Affected object returns the translated message based on the GOL (see page 48) 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 (see page 314)() function must be called when GOL (

see page 48) drawing is completed. In this case all objects have been drawn and it is safe to change objects states. GOLMsg (see page 314)() call can be done if GOLDraw (see page 296)() function returns non-zero or inside GOLDrawCallback (see page 297)() function.

8.4.1 GOLMsg Function

File

GOL.h (see page 552)

C

```
void GOLMsg(
    GOL_MSG * pMsg
);
```

Input Parameters

Input Parameters	Description
GOL_MSG * pMsg	Pointer to the GOL (see page 48) 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 (see page 48) 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 (see page 314)() 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 (see page 48) Messages section for deatils.

This function should be called when GOL (see page 48) drawing is completed. It can be done when GOLDraw (see page 296)() returns non-zero value or inside GOLDrawCallback (see page 297)() function.

Syntax

void GOLMsg(GOL_MSG (see page 316) *pMsg)

8.4.2 GOLMsgCallback Function

File

GOL.h (see page 552)

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 (see page 48) 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 (see page 48).

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 (see page 314)() 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 (see page 48) will not perform any action.

Syntax

WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER (see page 52)* pObj, GOL_MSG (see page 316)* pMsg)

8.4.3 GOL_MSG Structure

File

GOL.h (see page 552)

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 (see page 48) message used in the library.

- The types must be one of the INPUT_DEVICE_TYPE (see page 319):
- TYPE_UNKNOWN
- TYPE_KEYBOARD
- TYPE_TOUCHSCREEN
- TYPE_MOUSE
- uiEvent must be one of the INPUT_DEVICE_EVENT (see page 319).
- 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

8.4.4 TRANS_MSG Enumeration

File

GOL.h (see page 552)

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 (see page 65) check action ID.
WND_MSG_CLIENT	Window (see page 278) client area selected action ID.
WND_MSG_TITLE	Window (see page 278) title bar selected action ID.
BTN_MSG_PRESSED	Button (see page 65) pressed action ID.
BTN_MSG_STILLPRESSED	Button (see page 65) is continuously pressed ID.
BTN_MSG_RELEASED	Button (see page 65) released action ID.
BTN_MSG_CANCELPRESS	Button (see page 65) released action ID with button press canceled.
PICT_MSG_SELECTED	Picture (see page 210) selected action ID.
GB_MSG_SELECTED	Group Box selected action ID.
CC_MSG_SELECTED	Custom Control selected action ID.
SLD_MSG_INC	Slider (see page 237) or Scroll (see page 419) Bar (see page 352) increment action ID.
SLD_MSG_DEC	Slider (see page 237) or Scroll (see page 419) Bar (see page 352) decrement action ID.
ST_MSG_SELECTED	Static Text selected action ID.
DM_MSG_SELECTED	Digital Meter (see page 196) selected action ID.
PB_MSG_SELECTED	Progress Bar (see page 352) selected action ID.
RD_MSG_CLOCKWISE	Dial (see page 124) move clockwise action ID.
RD_MSG_CTR_CLOCKWISE	Dial (see page 124) move counter clockwise action ID.
MTR_MSG_SET	Meter (see page 196) 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 (see page 153) item touched action ID.
GRID_MSG_ITEM_SELECTED	Grid (see page 153) item selected action ID.
GRID_MSG_UP	Grid (see page 153) up action ID.
GRID_MSG_DOWN	Grid (see page 153) down action ID.
GRID_MSG_LEFT	Grid (see page 153) left action ID.
GRID_MSG_RIGHT	Grid (see page 153) right action ID.
CH_MSG_SELECTED	Chart (see page 77) 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 (see page 48) Objects used in the library.

8.4.5 INPUT_DEVICE_EVENT Enumeration

File

GOL.h (see page 552)

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 (see page 48) message events used in the library.

8.4.6 INPUT_DEVICE_TYPE Enumeration

File

GOL.h (see page 552)

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.

8.4.7 Scan Key Codes

Macros

Name	Description
SCAN_BS_PRESSED (see page 321)	Back space key pressed.
SCAN_BS_RELEASED (see page 321)	Back space key released.
SCAN_CR_PRESSED (see page 321)	Carriage return pressed.
SCAN_CR_RELEASED (see page 321)	Carriage return released.
SCAN_DEL_PRESSED (see page 322)	Delete key pressed.
SCAN_DEL_RELEASED (see page 322)	Delete key released.
SCAN_DOWN_PRESSED (see page 322)	Down key pressed.
SCAN_DOWN_RELEASED (see page 322)	Down key released.
SCAN_END_PRESSED (see page 323)	End key pressed.
SCAN_END_RELEASED (see page 323)	End key released.
SCAN_HOME_PRESSED (see page 323)	Home key pressed.
SCAN_HOME_RELEASED (see page 323)	Home key released.
SCAN_LEFT_PRESSED (see page 323)	Left key pressed.
SCAN_LEFT_RELEASED (see page 324)	Left key released.
SCAN_PGDOWN_PRESSED (see page 324)	Page down key pressed.
SCAN_PGDOWN_RELEASED (see page 324)	Page down key released.
SCAN_PGUP_PRESSED (see page 324)	Page up key pressed.
SCAN_PGUP_RELEASED (see page 325)	Page up key released.
SCAN_RIGHT_PRESSED (see page 325)	Right key pressed.
SCAN_RIGHT_RELEASED (see page 325)	Right key released.

SCAN_SPACE_PRESSED (see page 325)	Space key pressed.
SCAN_SPACE_RELEASED (see page 325)	Space key released.
SCAN_TAB_PRESSED (see page 326)	Tab key pressed.
SCAN_TAB_RELEASED (see page 326)	Tab key released.
SCAN_UP_PRESSED (see page 326)	Up key pressed.
SCAN_UP_RELEASED (see page 326)	Up key released.

Description

The defined scan codes for AT keyboard.

8.4.7.1 SCAN_BS_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_BS_PRESSED 0x0E
```

Description

Back space key pressed.

8.4.7.2 SCAN_BS_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_BS_RELEASED 0x8E
```

Description

Back space key released.

8.4.7.3 SCAN_CR_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_CR_PRESSED 0x1C
```

Description

Carriage return pressed.

8.4.7.4 SCAN_CR_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_CR_RELEASED 0x9C
```

Description

Carriage return released.

8.4.7.5 SCAN_DEL_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_DEL_PRESSED 0x53
```

Description

Delete key pressed.

8.4.7.6 SCAN_DEL_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_DEL_RELEASED 0xD3
```

Description

Delete key released.

8.4.7.7 SCAN_DOWN_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_DOWN_PRESSED 0x50
```

Description

Down key pressed.

8.4.7.8 SCAN_DOWN_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_DOWN_RELEASED 0xD0
```

Description

Down key released.

8.4.7.9 SCAN_END_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_END_PRESSED 0x4F
```

Description

End key pressed.

8.4.7.10 SCAN_END_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_END_RELEASED 0xCF
```

Description

End key released.

8.4.7.11 SCAN_HOME_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_HOME_PRESSED 0x47
```

Description

Home key pressed.

8.4.7.12 SCAN_HOME_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_HOME_RELEASED 0xC7
```

Description

Home key released.

8.4.7.13 SCAN_LEFT_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_LEFT_PRESSED 0x4B
```

Description

Left key pressed.

8.4.7.14 SCAN_LEFT_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_LEFT_RELEASED 0xCB
```

Description

Left key released.

8.4.7.15 SCAN_PGDOWN_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_PGDOWN_PRESSED 0x51
```

Description

Page down key pressed.

8.4.7.16 SCAN_PGDOWN_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_PGDOWN_RELEASED 0xD1
```

Description

Page down key released.

8.4.7.17 SCAN_PGUP_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_PGUP_PRESSED 0x49
```

Description

Page up key pressed.

8.4.7.18 SCAN_PGUP_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_PGUP_RELEASED 0xC9
```

Description

Page up key released.

8.4.7.19 SCAN_RIGHT_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_RIGHT_PRESSED 0x4D
```

Description

Right key pressed.

8.4.7.20 SCAN_RIGHT_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_RIGHT_RELEASED 0xCD
```

Description

Right key released.

8.4.7.21 SCAN_SPACE_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_SPACE_PRESSED 0x39
```

Description

Space key pressed.

8.4.7.22 SCAN_SPACE_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_SPACE_RELEASED 0xB9
```

Description

Space key released.

8.4.7.23 SCAN_TAB_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_TAB_PRESSED 0x0F
```

Description

Tab key pressed.

8.4.7.24 SCAN_TAB_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_TAB_RELEASED 0x8F
```

Description

Tab key released.

8.4.7.25 SCAN_UP_PRESSED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_UP_PRESSED 0x48
```

Description

Up key pressed.

8.4.7.26 SCAN_UP_RELEASED Macro

File

ScanCodes.h (see page 622)

C

```
#define SCAN_UP_RELEASED 0xC8
```

Description

Up key released.

8.5 Style Scheme

All objects uses a style scheme structure that defines the font and colors used.

Functions

	Name	Description
	GOLCreateScheme (see page 328)	This function creates a new style scheme object and initializes the parameters to default values.

Macros

Name	Description
GOLSetScheme (see page 329)	This macro sets the GOL (see page 48) scheme to be used for the object.
GOLGetScheme (see page 330)	This macro gets the GOL (see page 48) scheme used by the given object.
GOLGetSchemeDefault (see page 330)	This macro returns the default GOL (see page 48) scheme pointer.
GOL_EMBOSS_SIZE (see page 332)	This option defines the 3-D effect emboss size for objects. The default value of this is 3 set in GOL.h (see page 552). If it is not defined in GraphicsConfig.h (see page 928) file then the default value is used.
RGBConvert (see page 333)	This macro converts the color data to the selected COLOR_DEPTH (see page 41).

Module

Graphics Object Layer (see page 48)

Structures

Name	Description
GOL_SCHEME (see page 331)	GOL (see page 48) scheme defines the style scheme to be used by an object.

Variables

Name	Description
GOLFontDefault (see page 332)	This is variable GOLFontDefault.
GOLSchemeDefault (see page 332)	This defines a default GOL (see page 48) scheme that gets populated when an application calls the GOLCreateScheme (see page 328)(). The application is able to define a default GOL (see page 48) scheme in GraphicsConfig header file.

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;
} 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 TextColor1	Generic text colors used by the objects. Usage may vary from one object type to another.
TextColorDisabled	Text color used for objects that are disabled.
Color0 Color1	Generic colors used to render objects. Usage may vary from one object type to another.
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.

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 (see page 287), the CommonBkColor is used to fill area occupied by object.

Style scheme can be created with GOLCreateScheme (see page 328)() function that returns a pointer to the newly created GOL_SCHEME (see page 331) structure with default values automatically assigned. The default settings of the style scheme 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 (see page 332)

The default values can be changed in the GOL.h (see page 552) file.

8.5.1 GOLCreateScheme Function

File

GOL.h (see page 552)

C

```
GOL_SCHEME * GOLCreateScheme();
```

Side Effects

none

Returns

Pointer to the new GOL_SCHEME (see page 331) created.

Preconditions

none

Example

```
extern const char Font22[] __attribute__((aligned(2)));
extern const char Font16[] __attribute__((aligned(2)));

GOLSCHEME *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.

Syntax

GOL_SCHEME (see page 331) *GOLCreateScheme()

8.5.2 GOLSetScheme Macro

File

GOL.h (see page 552)

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;
GOLSCHEME *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 (see page 48) scheme to be used for the object.

Syntax

GOLSetScheme(pObj, pScheme)

8.5.3 GOLGetScheme Macro

File

GOL.h (see page 552)

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

```
GOLSCHEME *pScheme2;
BUTTON *pButton;

// assume button is created and initialized
// get the scheme assigned to pButton
pScheme2 = GOLGetScheme(pButton);
```

Overview

This macro gets the GOL (see page 48) scheme used by the given object.

Syntax

GOLGetScheme(pObj)

8.5.4 GOLGetSchemeDefault Macro

File

GOL.h (see page 552)

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 (see page 48) scheme pointer.

Syntax

GOLGetSchemeDefault()

8.5.5 GOL_SCHEME Structure

File

GOL.h (see page 552)

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;
  BYTE 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.
BYTE AlphaValue;	Alpha value used for alpha blending.
GFX_GRADIENT_STYLE gradientScheme;	Gradient Scheme for widgets

Overview

GOL (see page 48) scheme defines the style scheme to be used by an object.

8.5.6 Default Style Scheme Settings

Variables

Name	Description
FONTDEFAULT (see page 332)	Default GOL (see page 48) font.

Description

Lists the default settings for the style scheme.

8.5.6.1 FONTDEFAULT Variable

File

GOL.h (see page 552)

C

```
const FONT_FLASH FONTDEFAULT;
```

Description

Default GOL (see page 48) font.

8.5.7 GOLFontDefault Variable

File

GOLFontDefault.c (see page 579)

C

```
const FONT_FLASH GOLFontDefault = { (FLASH | COMP_NONE), __GOLFontDefault };
```

Description

This is variable GOLFontDefault.

8.5.8 GOL_EMBOSS_SIZE Macro

File

GOL.h (see page 552)

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 (see page 552). If it is not defined in GraphicsConfig.h (see page 928) file then the default value is used.

8.5.9 GOLSchemeDefault Variable

File

GOLSchemeDefault.c

C

```
const GOL_SCHEME GOLSchemeDefault = { BLACK, WHITE, WHITE, BLACK, WHITE, BLACK, WHITE,
BLACK, 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, 0, { GRAD_NONE, RGBConvert(0xA9, 0xDB, 0xEF),
```

```
RGBConvert(0x26, 0xC7, 0xF2), 50 } };
```

Description

This defines a default GOL (see page 48) scheme that gets populated when an application calls the GOLCreateScheme (see page 328)(). The application is able to define a default GOL (see page 48) scheme in GraphicsConfig header file.

8.5.10 RGBConvert Macro

File

gfxcolors.h

C

```
#define RGBConvert(red, green, blue) (GFX_COLOR) ((GFX_COLOR)red | (GFX_COLOR)green|
(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 (see page 41).

COLOR_DEPTH (see page 41)	Conversion
1	8-8-8 to 8-8-8 conversion or no conversion
8	8-8-8 to 3-3-2 conversion
16	8-8-8 to to 5-6-5 conversion
24	8-8-8 to 8-8-8 conversion or no conversion

Syntax

RGBConvert(red, green, blue)

8.6 GOL Global Variables

Module

Graphics Object Layer (see page 48)

Variables

Name	Description
_pDefaultGolScheme (see page 334)	Pointer to the GOL (see page 48) default scheme (GOL_SCHEME (see page 331)). This scheme is created in GOLInit (see page 306)() function. GOL (see page 48) scheme defines the style scheme to be used by an object. Use GOLGetSchemeDefault (see page 330)() to get this pointer.
_pGolObjects (see page 334)	Pointer to the current linked list of objects displayed and receiving messages. GOLDraw (see page 296)() and GOLMsg (see page 314)() process objects referenced by this pointer.
_pObjectFocused (see page 335)	Pointer to the object receiving keyboad input. This pointer is used or modified by the following APIs: • GOLSetFocus (see page 304)() • GOLGetFocus (see page 306)() • GOLGetFocusNext (see page 307)() • GOLGetFocusPrev (see page 308)() • GOLCanBeFocused (see page 307)()

Description

Graphics Object Layer global variables.

8.6.1 _pDefaultGolScheme Variable

File

GOL.h (see page 552)

C

```
GOL_SCHEME * _pDefaultGolScheme;
```

Overview

Pointer to the GOL (see page 48) default scheme (GOL_SCHEME (see page 331)). This scheme is created in GOLInit (see page 306)() function. GOL (see page 48) scheme defines the style scheme to be used by an object. Use GOLGetSchemeDefault (see page 330)() to get this pointer.

8.6.2 _pGolObjects Variable

File

GOL.h (see page 552)

C

```
OBJ_HEADER * _pGolObjects;
```

Overview

Pointer to the current linked list of objects displayed and receiving messages. GOLDraw (see page 296)() and GOLMsg (see page 314)() process objects referenced by this pointer.

8.6.3 _pObjectFocused Variable

File

GOL.h (see page 552)

C

```
OBJ_HEADER * _pObjectFocused;
```

Overview

Pointer to the object receiving keyboad input. This pointer is used or modified by the following APIs:

- GOLSetFocus (see page 304)()
- GOLGetFocus (see page 306)()
- GOLGetFocusNext (see page 307)()
- GOLGetFocusPrev (see page 308)()
- GOLCanBeFocused (see page 307)()

8.7 Object Rendering

Object rendering can be configured to Blocking or Non-Blocking manner.

Module

Graphics Object Layer (see page 48)

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 (see page 48) 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 (see page 296)() function. The figure below explains the rendering loop of the GOLDraw (see page 296)() function.

GOL (see page 48) 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 (see page 296)() function should be called again. The next call to GOLDraw (see page 296)() 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 (see page 296)() calls user defined GOLDrawCallback (see page 297)() function. User drawing can be done in this callback function. If the function returns a zero, drawing of GOL (see page 48) objects in the active list is suspended. In this case color, clipping region, line type and graphic cursor will not be modified by GOL (see page 48). If it returns a 1 drawing control is returned to GOL (see page 48). GOLDraw (see page 296)() resume rendering of objects in the current active list. Inside the GOLDrawCallback (see page 297)() function, the active object list is not used by GOLDraw (see page 296)(). It is safe to perform modification of the list. Please refer to Configuration Settings (see page 41) to set Blocking or Non-Blocking configuration.

8.8 Types

8.8.1 GFX_COLOR Type

File

gfxcolors.h

C

```
typedef DWORD GFX_COLOR;
```

Module

Graphics Object Layer (see page 48)

Overview

Data type that defines the color data.

9 Graphics Primitive Layer

Enumerations

Name	Description
GFX_RESOURCE (see page 368)	Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.
GFX_GRADIENT_TYPE (see page 371)	Enumeration for gradient type

Structures

Name	Description
GFX_IMAGE_HEADER (see page 368)	Structure for images stored in various system memory (Flash, External Memory (SPI, Parallel Flash, or memory in EPMP).
IMAGE_FLASH (see page 369)	Structure for images stored in FLASH memory.
IMAGE_RAM (see page 370)	Structure for images stored in RAM memory.
GFX_EXTDATA (see page 370)	This structure is used to describe external memory.
GFX_GRADIENT_STYLE (see page 371)	This structure is used to describe the gradient style.

Types

Name	Description
IMAGE_EXTERNAL (see page 372)	Structure for images stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

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.

9.1 Text Functions

Functions

	Name	Description
	SetFont (see page 340)	This function sets the current font used in OutTextXY (see page 343)(), OutText (see page 343)() and OutChar (see page 342)() functions.
	OutChar (see page 342)	This function outputs a character from the current graphic cursor position. OutChar() uses the current active font set with SetFont (see page 340)().
	OutText (see page 343)	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 (see page 340)().
	OutTextXY (see page 343)	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 (see page 340)().

	GetTextHeight (see page 344)	This macro returns the height of the specified font. All characters in a given font table have a constant height.
	GetTextWidth (see page 345)	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 (see page 341)	This macro returns font orientation (0 == horizontal, 1 == vertical).
SetFontOrientation (see page 341)	This macro sets font orientation vertical or horizontal.
XCHAR (see page 345)	This macro sets the data type for the Fonts. There are three types • #define XCHAR unsigned short // use multibyte characters (0-2^16 range) • #define XCHAR unsigned char // use unsigned char (0-255 range) • #define XCHAR char // use signed char (0-127 range)

Module

Graphics Primitive Layer (see page 338)

Structures

Name	Description
FONT_HEADER (see page 339)	Structure describing the font header.
FONT_FLASH (see page 340)	Structure for font stored in FLASH memory.

Types

Name	Description
FONT_EXTERNAL (see page 340)	Structure for font stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Description

This lists the Primitive level text functions.

9.1.1 FONT_HEADER Structure

File

Primitive.h (see page 1008)

C

```
typedef struct {
  BYTE fontID;
  BYTE res1 : 4;
  BYTE orientation : 2;
  BYTE res2 : 2;
  WORD firstChar;
  WORD lastChar;
  BYTE height;
  BYTE reserved;
} FONT_HEADER;
```

Members

Members	Description
BYTE fontID;	User assigned value
BYTE res1 : 4;	Reserved for future use (must be set to 0).

BYTE orientation : 2;	Orientation of the character glyphs (0,90,180,270 degrees) 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).
BYTE height;	Font characters height in pixels.
BYTE reserved;	Reserved for future use (must be set to 0).

Overview

Structure describing the font header.

9.1.2 FONT_FLASH Structure

File

Primitive.h (see page 1008)

C

```
typedef struct {
  GFX_RESOURCE type;
  const char * address;
} FONT_FLASH;
```

Members

Members	Description
GFX_RESOURCE type;	must be FLASH
const char * address;	font image address in FLASH

Overview

Structure for font stored in FLASH memory.

9.1.3 FONT_EXTERNAL Type

File

Primitive.h (see page 1008)

C

```
typedef GFX_EXTDATA FONT_EXTERNAL;
```

Overview

Structure for font stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

9.1.4 SetFont Function

File

Primitive.h (see page 1008)

C

```
void SetFont(
    void * font
);
```

Input Parameters

Input Parameters	Description
void * font	Pointer to the new font image to be used.

Side Effects

none

Returns

none

Example

See OutTextXY (see page 343)() example.

Overview

This function sets the current font used in OutTextXY (see page 343)(), OutText (see page 343)() and OutChar (see page 342)() functions.

Syntax

void SetFont(void* font)

9.1.5 GetFontOrientation Macro

File

Primitive.h (see page 1008)

C

```
#define GetFontOrientation _fontOrientation
```

Returns

font orientation (0 == horizontal, 1 == vertical)

Preconditions

none

Overview

This macro returns font orientation (0 == horizontal, 1 == vertical).

Syntax

GetFontOrientation()

9.1.6 SetFontOrientation Macro

File

Primitive.h (see page 1008)

C

```
#define SetFontOrientation(orient) _fontOrientation = orient;
```

Input Parameters

Input Parameters	Description
orient	should be non-zero if the font orientation is vertical

Returns

none

Preconditions

none

Overview

This macro sets font orientation vertical or horizontal.

Syntax

SetFontOrientation(orient)

9.1.7 OutChar Function

File

Primitive.h (see page 1008)

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

Overview

This function outputs a character from the current graphic cursor position. OutChar() uses the current active font set with SetFont (see page 340)().

Syntax

WORD OutChar(XCHAR (see page 345) ch)

9.1.8 OutText Function

File

Primitive.h (see page 1008)

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.

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 (see page 340)().

Syntax

WORD OutText(XCHAR (see page 345)* textString)

9.1.9 OutTextXY Function

File

Primitive.h (see page 1008)

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 (see page 340)().

Syntax

WORD OutTextXY(SHORT x, SHORT y, XCHAR (see page 345)* textString)

9.1.10 GetTextHeight Function

File

Primitive.h (see page 1008)

C

```
SHORT GetTextHeight(
    void * font
);
```

Input Parameters

Input Parameters	Description
void * font	Pointer to the font image.

Side Effects

none

Returns

Returns the font height.

Example

See OutTextXY (see page 343)() 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* font)

9.1.11 GetTextWidth Function

File

Primitive.h (see page 1008)

C

```
SHORT GetTextWidth(
    XCHAR * textString,
    void * font
);
```

Input Parameters

Input Parameters	Description
XCHAR * textString	Pointer to the string.
void * font	Pointer to the font image.

Side Effects

none

Returns

Returns the string width in the specified font.

Example

See OutTextXY (see page 343)() 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 (see page 345)* textString, void* font)

9.1.12 XCHAR Macro

File

Primitive.h (see page 1008)

C

```
#define XCHAR char
```

Overview

This macro sets the data type for the Fonts. There are three types

- #define XCHAR unsigned short // use multibyte characters (0-2^16 range)
- #define XCHAR unsigned char // use unsigned char (0-255 range)
- #define XCHAR char // use signed char (0-127 range)

9.2 Gradients

Module

Graphics Primitive Layer (see page 338)

Description

Gradients can be drawn dynamically with the Microchip Graphics Library. There are two functions created to do the gradient rendering. They are:

```
Function:  BarGradient(short left, short top, short right, short bottom, GFX_COLOR color1,
GFX_COLOR color2, DWORD length,                      BYTE direction);

 Input: 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.
         color1 - start color for the gradient
        color2 - end color for the gradient
        length - From 0-100%. How much of a gradient is wanted
        direction - Gradient Direction
 Output: Returns 1 if the rendering is done, 0 if not yet done.

 Side Effects: none

 Overview: This renders a bar onto the screen, but instead of one color a gradient is drawn
 depending on the direction, length, and colors chosen
 Function:  BevelGradient(short x1, short y1, short x2, short y2,SHORT rad, GFX_COLOR
color1, GFX_COLOR color2, DWORD                        length, BYTE direction);

 Input: x1 - x coordinate position of the upper left center of the circle that
              draws the rounded corners.
         y1 - y coordinate position of the upper left center of the circle that
              draws the rounded corners.
         x2 - x coordinate position of the lower right center of the circle that
              draws the rounded corners.
         y2 - y coordinate position of the lower right center of the circle that
              draws the rounded corners.
        rad - defines the redius of the circle, that draws the rounded corners.
         color1 - start color for the gradient
        color2 - end color for the gradient
        length - From 0-100%. How much of a gradient is wanted
        direction - Gradient Direction

 Output: none

 Overview: This renders a gradient on the screen. It works the same as the fillbevel
function,
 except a gradient out of color1 and color2 is drawn depending on the direction.
```

An example below shows the direction inputs needed for the gradients rendering.

9.3 Line Functions

Functions

	Name	Description
	Line (see page 347)	This function draws a line with the current line type from the start point to the end point.

Macros

Name	Description
LineRel (see page 348)	This macro draws a line with the current line type from the current graphic cursor position to the position defined by displacement.
LineTo (see page 349)	This macro draws a line with the current line type from the current graphic cursor position to the given x, y position.
SetLineThickness (see page 349)	This macro sets sets line thickness to 1 pixel or 3 pixels.
SetLineType (see page 350)	This macro sets the line type to draw.

Module

Graphics Primitive Layer (see page 338)

Description

This lists the Primitive line text functions.

9.3.1 Line Function

File

Primitive.h (see page 1008)

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)

9.3.2 LineRel Macro

File

Primitive.h (see page 1008)

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)

9.3.3 LineTo Macro

File

Primitive.h (see page 1008)

C

```
#define LineTo(x, y) Line(_cursorX, _cursorY, x, y)
```

Input Parameters

Input Parameters	Description
x	End point x position.
y	End point y poisiton.

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)

9.3.4 SetLineThickness Macro

File

Primitive.h (see page 1008)

C

```
#define SetLineThickness(lnThickness) _lineThickness = lnThickness;
```

Input Parameters

Input Parameters	Description
lnThickness	Line (see page 347) thickness code (0 - 1 pixel; 1 - 3 pixels)

Side Effects

none

Returns

none

Overview

This macro sets sets line thickness to 1 pixel or 3 pixels.

Syntax

SetLineThickness(lnThickness)

9.3.5 SetLineType Macro

File

Primitive.h (see page 1008)

C

```
#define SetLineType(lnType) _lineType = lnType;
```

Input Parameters

Input Parameters	Description
lnType	The type of line to be used. Supported line types: • SOLID_LINE (see page 351) • DOTTED_LINE (see page 351) • DASHED_LINE (see page 351)

Side Effects

none

Returns

none

Overview

This macro sets the line type to draw.

Syntax

SetLineType(lnType)

9.3.6 Line Types

Macros

Name	Description
SOLID_LINE (see page 351)	Solid Line (see page 347) Style
DASHED_LINE (see page 351)	Dashed Line (see page 347) Style
DOTTED_LINE (see page 351)	Dotted Line (see page 347) Style

Description

Line type definitions.

9.3.6.1 SOLID_LINE Macro

File

Primitive.h (see page 1008)

C

```
#define SOLID_LINE 0
```

Description

Solid Line (see page 347) Style

9.3.6.2 DASHED_LINE Macro

File

Primitive.h (see page 1008)

C

```
#define DASHED_LINE 4
```

Description

Dashed Line (see page 347) Style

9.3.6.3 DOTTED_LINE Macro

File

Primitive.h (see page 1008)

C

```
#define DOTTED_LINE 1
```

Description

Dotted Line (see page 347) Style

9.3.7 Line Size

Macros

Name	Description
NORMAL_LINE (see page 351)	Normal Line (see page 347) (thickness is 1 pixel)
THICK_LINE (see page 352)	Thick Line (see page 347) (thickness is 3 pixels)

Description

Line sizes definition.

9.3.7.1 NORMAL_LINE Macro

File

Primitive.h (see page 1008)

C

```
#define NORMAL_LINE 0
```

Description

Normal Line (see page 347) (thickness is 1 pixel)

9.3.7.2 THICK_LINE Macro

File

Primitive.h (see page 1008)

C

```
#define THICK_LINE 1
```

Description

Thick Line (see page 347) (thickness is 3 pixels)

9.4 Rectangle Functions

Functions

	Name	Description
	Bar (see page 352)	This function draws a bar given the left, top and right, bottom corners with the current color.
	DrawPoly (see page 354)	This function draws a polygon with the current line type using the given number of points. The polygon points are stored in an array arranged in the following order:

Macros

Name	Description
Rectangle (see page 353)	This macro draws a rectangle with the given left, top and right, bottom corners. Current line type is used.

Module

Graphics Primitive Layer (see page 338)

Description

This lists the Primitive level rectangle functions.

9.4.1 Bar Function

File

Primitive.h (see page 1008)

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 color.

Syntax

WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)

9.4.2 Rectangle Macro

File

Primitive.h (see page 1008)

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)

9.4.3 DrawPoly Function

File

Primitive.h (see page 1008)

C

```
WORD DrawPoly(
    SHORT numPoints,
    SHORT * polyPoints
);
```

Input Parameters

Input Parameters	Description
SHORT numPoints	Defines the number of points in the polygon.
SHORT * polyPoints	Pointer to the array of polygon points.

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 polygon with the current line type using the given number of points. The polygon points are stored in an array arranged in the following order:

SHORT polyPoints[numPoints] = {x0, y0, x1, y1, x2, y2 … xn, yn};
Where n = numPoints - 1

Syntax

WORD DrawPoly(SHORT numPoints, SHORT* polyPoints)

9.5 Circle Functions

Functions

	Name	Description
	Arc (see page 356)	Draws the octant arc of a beveled figure with given centers, radii and octant mask. When r1 is zero and r2 has some value, a filled circle is drawn; when the radii have values, an arc of thickness (r2-r1) is drawn; when octant = 0xFF, a full ring is drawn. When r1 and r2 are zero, a rectangular object is drawn, where xL, yT specifies the left top corner; xR, yB specifies the right bottom corner.
	Bevel (see page 357)	Draws a beveled figure on the screen. For a pure circular object x1 = x2 and y1 = y2. For a rectangular object radius = 0.

	FillBevel (see page 358)	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 (see page 355)	This macro draws a circle with the given center and radius.
FillCircle (see page 355)	This macro draws a filled circle. Uses the FillBevel (see page 358)() function.
SetBevelDrawType (see page 359)	This macro sets the fill bevel type to be drawn.

Module

Graphics Primitive Layer (see page 338)

Description

This lists the Primitive level circle functions.

9.5.1 Circle Macro

File

Primitive.h (see page 1008)

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)

9.5.2 FillCircle Macro

File

Primitive.h (see page 1008)

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 redius 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 (see page 358)() function.

Syntax

FillCircle(SHORT x1, SHORT y1, SHORT rad)

9.5.3 Arc Function

File

Primitive.h (see page 1008)

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 cicles 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 = +radius • bit0 : first octant • bit1 : second octant • bit2 : third octant • bit3 : fourth octant • bit4 : fifth octant • bit5 : sixth octant • bit6 : seventh octant • bit7 : eight octant

Side Effects

none

Returns

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

Preconditions

none

Overview

Draws the octant arc of a beveled figure with given centers, radii and octant mask. When r1 is zero and r2 has some value, a filled circle is drawn; when the radii have values, an arc of thickness (r2-r1) is drawn; when octant = 0xFF, a full ring is drawn. When r1 and r2 are zero, a rectangular object is drawn, where xL, yT specifies the left top corner; xR, yB specifies the right bottom corner.

Syntax

WORD Arc(SHORT xL, SHORT yT, SHORT xR, SHORT yB, SHORT r1, SHORT r2, BYTE octant)

9.5.4 Bevel Function

File

Primitive.h (see page 1008)

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 redius 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.

Overview

Draws a beveled figure on the screen. For a pure circular object x1 = x2 and y1 = y2. For a rectangular object radius = 0.

Syntax

WORD Bevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)

9.5.5 FillBevel Function

File

Primitive.h (see page 1008)

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 redius 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.

Overview

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.

Syntax

WORD FillBevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)

9.5.6 SetBevelDrawType Macro

File

Primitive.h (see page 1008)

C

```
#define SetBevelDrawType(type) (_bevelDrawType = type)
```

Input Parameters

Input Parameters	Description
type	is set using the following. • 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)

9.6 Graphic Cursor

Macros

Name	Description
GetX (see page 360)	This macro returns the current graphic cursor x-coordinate.
GetY (see page 360)	This macro returns the current graphic cursor y-coordinate.
MoveRel (see page 361)	This macro moves the graphic cursor relative to the current location. The given dX and dY displacement can be positive or negative numbers.
MoveTo (see page 361)	This macro moves the graphic cursor to new x,y position.

Module

Graphics Primitive Layer (see page 338)

Description

This lists the functions to control the graphics cursor.

9.6.1 GetX Macro

File

Primitive.h (see page 1008)

C

```
#define GetX _cursorX
```

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro returns the current graphic cursor x-coordinate.

Syntax

GetX()

9.6.2 GetY Macro

File

Primitive.h (see page 1008)

C

```
#define GetY _cursorY
```

Side Effects

none

Returns

none

Preconditions

none

Overview

This macro returns the current graphic cursor y-coordinate.

Syntax

GetX (see page 360)()

9.6.3 MoveRel Macro

File

Primitive.h (see page 1008)

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)

9.6.4 MoveTo Macro

File

Primitive.h (see page 1008)

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)

9.7 Bitmap Functions

Functions

	Name	Description
	PutImage (see page 362)	This function outputs image starting from left,top coordinates.
	GetImageHeight (see page 363)	This function returns the image height.
	GetImageWidth (see page 363)	This function returns the image width.

Module

Graphics Primitive Layer (see page 338)

Structures

Name	Description
BITMAP_HEADER (see page 364)	Structure describing the bitmap header.

Description

This lists the functions to display bitmaps.

9.7.1 PutImage Function

File

Primitive.h (see page 1008)

C

```
WORD PutImage(
    SHORT left,
    SHORT top,
    void * bitmap,
    BYTE stretch
);
```

Input Parameters

Input Parameters	Description
SHORT left	x coordinate position of the left top corner.
SHORT top	y coordinate position of the left top corner.
void * bitmap	pointer to the bitmap.
BYTE stretch	The image stretch factor.

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 function outputs image starting from left,top coordinates.

Syntax

WORD PutImage(SHORT left, SHORT top, void* bitmap, BYTE stretch)

9.7.2 GetImageHeight Function

File

Primitive.h (see page 1008)

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)

9.7.3 GetImageWidth Function

File

Primitive.h (see page 1008)

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)

9.7.4 BITMAP_HEADER Structure

File

Primitive.h (see page 1008)

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.

9.7.5 Bitmap Settings

Macros

Name	Description
IMAGE_NORMAL (see page 365)	Normal image stretch code
IMAGE_X2 (see page 365)	Stretched image stretch code

Description

Bitmap rendering settings.

9.7.5.1 IMAGE_NORMAL Macro

File

Primitive.h (see page 1008)

C

```
#define IMAGE_NORMAL 1
```

Description

Normal image stretch code

9.7.5.2 IMAGE_X2 Macro

File

Primitive.h (see page 1008)

C

```
#define IMAGE_X2 2
```

Description

Stretched image stretch code

9.7.6 Bitmap Source

Bitmap data structure is dependent on the location.

9.8 External Memory

This lists the external memory access functions and descriptions.

Macros

Name	Description
EXTERNAL_FONT_BUFFER_SIZE (see page 366)	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.

Module

Graphics Primitive Layer (see page 338)

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 (see page 362)() or SetFont (see page 340)()). 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.

9.8.1 EXTERNAL_FONT_BUFFER_SIZE Macro

File

Primitive.h (see page 1008)

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.

9.8.2 Memory Type

Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.

9.9 Set Up Functions

Functions

	Name	Description
	ClearDevice (see page 367)	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 (see page 367)	This function initializes the display controller, sets the line type to SOLID_LINE (see page 351), sets the screen to all BLACK (see page 389), sets the current drawing color to WHITE (see page 394), 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.

Module

Graphics Primitive Layer (see page 338)

Description

This lists the Primitive set up and initialization functions.

9.9.1 ClearDevice Function

File

Primitive.h (see page 1008)

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)

9.9.2 InitGraph Function

File

Primitive.h (see page 1008)

C

```
void InitGraph();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

This function initializes the display controller, sets the line type to SOLID_LINE (see page 351), sets the screen to all BLACK (see page 389), sets the current drawing color to WHITE (see page 394), 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)

9.10 GFX_RESOURCE Enumeration

File

Primitive.h (see page 1008)

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 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)

Module

Graphics Primitive Layer (see page 338)

Overview

Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.

9.11 GFX_IMAGE_HEADER Structure

File

Primitive.h (see page 1008)

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;	size of the IPU compressed data
DWORD param2;	size of the IPU decompressed data
WORD colorDepth;	color depth of the image

Module

Graphics Primitive Layer (see page 338)

Overview

Structure for images stored in various system memory (Flash, External Memory (SPI, Parallel Flash, or memory in EPMP).

9.12 IMAGE_FLASH Structure

File

Primitive.h (see page 1008)

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

Module

Graphics Primitive Layer (see page 338)

Overview

Structure for images stored in FLASH memory.

9.13 IMAGE_RAM Structure

File

Primitive.h (see page 1008)

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

Module

Graphics Primitive Layer (see page 338)

Overview

Structure for images stored in RAM memory.

9.14 GFX_EXTDATA Structure

File

Primitive.h (see page 1008)

C

```
typedef struct {
  GFX_RESOURCE type;
  WORD ID;
  DWORD address;
} GFX_EXTDATA;
```

Members

Members	Description
GFX_RESOURCE type;	must be EXTERNAL or 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 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)

Module

Graphics Primitive Layer (see page 338)

Overview

This structure is used to describe external memory.

9.15 GFX_GRADIENT_STYLE Structure

File

Primitive.h (see page 1008)

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

Module

Graphics Primitive Layer (see page 338)

Overview

This structure is used to describe the gradient style.

9.16 GFX_GRADIENT_TYPE Enumeration

File

Primitive.h (see page 1008)

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 (see page 346) 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

Module

Graphics Primitive Layer (see page 338)

Overview

Enumeration for gradient type

9.17 Types

9.17.1 IMAGE_EXTERNAL Type

File

Primitive.h (see page 1008)

C

```
typedef GFX_EXTDATA IMAGE_EXTERNAL;
```

Module

Graphics Primitive Layer (see page 338)

Overview

Structure for images stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

10 Display Device Driver Layer

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. It is recommended that the application keep the same format when naming the display macro, GFX_USE_DISPLAY_CONTROLLER_<DRIVER NAME>.

10.1 Display Device Driver Level Primitives

This lists the Device Level Primitive rendering functions and macros.

Functions

	Name	Description
	GetPixel (see page 374)	Returns pixel color at the given x,y coordinate position.
	PutPixel (see page 375)	Puts pixel with the given x,y coordinate position.
	SetClip (see page 378)	Enables/disables clipping.
	SetClipRgn (see page 378)	Sets clipping region.
	TransparentColorEnable (see page 381)	Sets current transparent color.

	DisplayBrightness (see page 383)	Sets the brightness of the display.
	CopyBlock (see page 384)	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 (see page 386)() and but instead of using left, top position of the source and destination, it uses offsets instead. This is a blocking function.
	CopyPageWindow (see page 385)	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 (see page 386)	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 (see page 441)() but instead of using page numbers, addresses are used for versatility. This is a blocking function.
	SetActivePage (see page 387)	Sets active graphic page.
	SetVisualPage (see page 388)	Sets graphic page to display.

Macros

Name	Description
GetColor (see page 376)	Returns current drawing color.
SetColor (see page 376)	Sets current drawing color.
GetMaxX (see page 377)	Returns maximum horizontal coordinate.
GetMaxY (see page 377)	Returns maximum vertical coordinate.
GetClipBottom (see page 379)	Returns bottom clipping border.
GetClipLeft (see page 379)	Returns left clipping border.
GetClipRight (see page 380)	Returns right clipping border.
GetClipTop (see page 380)	Returns top clipping border.
CLIP_DISABLE (see page 381)	Disables clipping.
CLIP_ENABLE (see page 381)	Enables clipping.
TransparentColorDisable (see page 382)	Disables the transparent color function.
GetTransparentColor (see page 382)	Returns the current transparent color value.
TRANSPARENT_COLOR_DISABLE (see page 383)	Check of transparent color is not performed
TRANSPARENT_COLOR_ENABLE (see page 383)	Check pixel if color is equal to transparent color, if equal do not render pixel
GetPageAddress (see page 384)	Returns the address of the given page.

Module

Display Device Driver Layer (see page 373)

10.1.1 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 (see page 337) GetPixel(SHORT x, SHORT y)

10.1.2 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)

10.1.3 GetColor Macro

File

DisplayDriver.h

C

```
#define GetColor _color
```

Side Effects

none

Returns

Color coded in 5:6:5 RGB format.

Preconditions

none

Overview

Returns current drawing color.

Syntax

GetColor()

10.1.4 SetColor Macro

File

DisplayDriver.h

C

```
#define SetColor(color) _color = (color)
```

Input Parameters

Input Parameters	Description
color	Color coded in 5:6:5 RGB format.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets current drawing color.

Syntax

SetColor(color)

10.1.5 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 dislayed 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()

10.1.6 GetMaxY Macro

File

DisplayDriver.h

C

```
#define GetMaxY (DISP_VER_RESOLUTION - 1)
```

Side Effects

none

Returns

Maximum vertical coordinate.

Preconditions

none

Example

(see GetMaxX (see page 377)()) example.

Overview

Returns maximum vertical coordinate.

Syntax

GetMaxY()

10.1.7 SetClip Function

File

DisplayDriver.h

C

```
void SetClip(
    BYTE control
);
```

Input Parameters

Input Parameters	Description
BYTE control	Enables or disables the clipping. • 0: Disable clipping • 1: Enable clipping

Side Effects

none

Returns

none

Preconditions

none

Overview

Enables/disables clipping.

Syntax

SetClip(control)

10.1.8 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)

10.1.9 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()

10.1.10 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()

10.1.11 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()

10.1.12 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()

10.1.13 CLIP_DISABLE Macro

File

DisplayDriver.h

C

```
#define CLIP_DISABLE 0   // Disables clipping.
```

Description

Disables clipping.

10.1.14 CLIP_ENABLE Macro

File

DisplayDriver.h

C

```
#define CLIP_ENABLE 1   // Enables clipping.
```

Description

Enables clipping.

10.1.15 TransparentColorEnable Function

File

DisplayDriver.h

C

```
void TransparentColorEnable(
    GFX_COLOR color
);
```

Input Parameters

Input Parameters	Description
GFX_COLOR color	Color value chosen.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets current transparent color.

Syntax

void TransparentColorEnable(GRFX_COLOR color)

10.1.16 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()

10.1.17 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()

10.1.18 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

10.1.19 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

10.1.20 DisplayBrightness Function

File

SSD1926.h (see page 1168)

C

```
void DisplayBrightness(
    WORD level
);
```

Input Parameters

Input Parameters	Description
WORD level	Brightness level. Valid values are 0 to 100. • 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)

10.1.21 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)

10.1.22 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 (see page 386)() 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)

10.1.23 CopyPageWindow Function

File

DisplayDriver.h

C

```
WORD 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)

10.1.24 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 (see page 441)() 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)

10.1.25 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.

10.1.26 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.

10.1.27 Color Definition

The device driver must also implement the definition of standard color set based on the color format used.

Macros

Name	Description
BLACK (see page 389)	USE_PALETTE
BLUE (see page 389)	This is macro BLUE.
BRIGHTBLUE (see page 389)	This is macro BRIGHTBLUE.
BRIGHTCYAN (see page 390)	This is macro BRIGHTCYAN.
BRIGHTGREEN (see page 390)	This is macro BRIGHTGREEN.
BRIGHTMAGENTA (see page 390)	This is macro BRIGHTMAGENTA.
BRIGHTRED (see page 390)	This is macro BRIGHTRED.
BRIGHTYELLOW (see page 390)	This is macro BRIGHTYELLOW.
BROWN (see page 391)	This is macro BROWN.
CYAN (see page 391)	This is macro CYAN.
DARKGRAY (see page 391)	This is macro DARKGRAY.
GRAY0 (see page 391)	This is macro GRAY0.
GRAY1 (see page 392)	This is macro GRAY1.
GRAY2 (see page 392)	This is macro GRAY2.

GRAY4 (see page 392)	This is macro GRAY4.
GRAY6 (see page 392)	This is macro GRAY6.
GREEN (see page 392)	This is macro GREEN.
LIGHTBLUE (see page 393)	This is macro LIGHTBLUE.
LIGHTCYAN (see page 393)	This is macro LIGHTCYAN.
LIGHTGRAY (see page 393)	This is macro LIGHTGRAY.
LIGHTGREEN (see page 393)	This is macro LIGHTGREEN.
LIGHTMAGENTA (see page 394)	This is macro LIGHTMAGENTA.
LIGHTRED (see page 394)	This is macro LIGHTRED.
MAGENTA (see page 394)	This is macro MAGENTA.
RED (see page 394)	This is macro RED.
WHITE (see page 394)	This is macro WHITE.
YELLOW (see page 395)	This is macro YELLOW.

Description

The following lists sample color definitions for a 16-bpp color encoding.

10.1.27.1 BLACK Macro

File

gfxcolors.h

C

```
#define BLACK RGBConvert(0, 0, 0)
```

Description

USE_PALETTE

10.1.27.2 BLUE Macro

File

gfxcolors.h

C

```
#define BLUE RGBConvert(0, 0, 128)
```

Description

This is macro BLUE.

10.1.27.3 BRIGHTBLUE Macro

File

gfxcolors.h

C

```
#define BRIGHTBLUE RGBConvert(0, 0, 255)
```

Description

This is macro BRIGHTBLUE.

10.1.27.4 BRIGHTCYAN Macro

File

gfxcolors.h

C

```
#define BRIGHTCYAN RGBConvert(0, 255, 255)
```

Description

This is macro BRIGHTCYAN.

10.1.27.5 BRIGHTGREEN Macro

File

gfxcolors.h

C

```
#define BRIGHTGREEN RGBConvert(0, 255, 0)
```

Description

This is macro BRIGHTGREEN.

10.1.27.6 BRIGHTMAGENTA Macro

File

gfxcolors.h

C

```
#define BRIGHTMAGENTA RGBConvert(255, 0, 255)
```

Description

This is macro BRIGHTMAGENTA.

10.1.27.7 BRIGHTRED Macro

File

gfxcolors.h

C

```
#define BRIGHTRED RGBConvert(255, 0, 0)
```

Description

This is macro BRIGHTRED.

10.1.27.8 BRIGHTYELLOW Macro

File

gfxcolors.h

C

```
#define BRIGHTYELLOW RGBConvert(255, 255, 0)
```

Description

This is macro BRIGHTYELLOW.

10.1.27.9 BROWN Macro

File

gfxcolors.h

C

```
#define BROWN RGBConvert(255, 128, 0)
```

Description

This is macro BROWN.

10.1.27.10 CYAN Macro

File

gfxcolors.h

C

```
#define CYAN RGBConvert(0, 128, 128)
```

Description

This is macro CYAN.

10.1.27.11 DARKGRAY Macro

File

gfxcolors.h

C

```
#define DARKGRAY RGBConvert(64, 64, 64)
```

Description

This is macro DARKGRAY.

10.1.27.12 GRAY0 Macro

File

gfxcolors.h

C

```
#define GRAY0 RGBConvert(224, 224, 224)
```

Description

This is macro GRAY0.

10.1.27.13 GRAY1 Macro

File

gfxcolors.h

C

```
#define GRAY1 RGBConvert(192, 192, 192)
```

Description

This is macro GRAY1.

10.1.27.14 GRAY2 Macro

File

gfxcolors.h

C

```
#define GRAY2 RGBConvert(160, 160, 160)
```

Description

This is macro GRAY2.

10.1.27.15 GRAY4 Macro

File

gfxcolors.h

C

```
#define GRAY4 RGBConvert(96, 96, 96)
```

Description

This is macro GRAY4.

10.1.27.16 GRAY6 Macro

File

gfxcolors.h

C

```
#define GRAY6 RGBConvert(32, 32, 32)
```

Description

This is macro GRAY6.

10.1.27.17 GREEN Macro

File

gfxcolors.h

C

```
#define GREEN RGBConvert(0, 128, 0)
```

Description

This is macro GREEN.

10.1.27.18 LIGHTBLUE Macro

File

gfxcolors.h

C

```
#define LIGHTBLUE RGBConvert(128, 128, 255)
```

Description

This is macro LIGHTBLUE.

10.1.27.19 LIGHTCYAN Macro

File

gfxcolors.h

C

```
#define LIGHTCYAN RGBConvert(128, 255, 255)
```

Description

This is macro LIGHTCYAN.

10.1.27.20 LIGHTGRAY Macro

File

gfxcolors.h

C

```
#define LIGHTGRAY RGBConvert(128, 128, 128)
```

Description

This is macro LIGHTGRAY.

10.1.27.21 LIGHTGREEN Macro

File

gfxcolors.h

C

```
#define LIGHTGREEN RGBConvert(128, 255, 128)
```

Description

This is macro LIGHTGREEN.

10.1.27.22 LIGHTMAGENTA Macro

File

gfxcolors.h

C

```
#define LIGHTMAGENTA RGBConvert(255, 128, 255)
```

Description

This is macro LIGHTMAGENTA.

10.1.27.23 LIGHTRED Macro

File

gfxcolors.h

C

```
#define LIGHTRED RGBConvert(255, 128, 128)
```

Description

This is macro LIGHTRED.

10.1.27.24 MAGENTA Macro

File

gfxcolors.h

C

```
#define MAGENTA RGBConvert(128, 0, 128)
```

Description

This is macro MAGENTA.

10.1.27.25 RED Macro

File

gfxcolors.h

C

```
#define RED RGBConvert(128, 0, 0)
```

Description

This is macro RED.

10.1.27.26 WHITE Macro

File

gfxcolors.h

C

```
#define WHITE RGBConvert(255, 255, 255)
```

Description

This is macro WHITE.

10.1.27.27 YELLOW Macro

File

gfxcolors.h

C

```
#define YELLOW RGBConvert(255, 255, 128)
```

Description

This is macro YELLOW.

10.2 Display Device Driver Control

This lists the device control functions and macros.

Functions

	Name	Description
	IsDeviceBusy (see page 395)	Returns non-zero if LCD controller is busy (previous drawing operation is not completed).
	ResetDevice (see page 396)	Initializes LCD module.

Macros

Name	Description
SetPalette (see page 396)	Sets palette register.

Module

Display Device Driver Layer (see page 373)

10.2.1 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()

10.2.2 SetPalette Macro

File

SH1101A_SSD1303.h (see page 1104)

C

```
#define SetPalette(colorNum, color)
```

Input Parameters

Input Parameters	Description
colorNum	Register number.
color	Color.

Side Effects

none

Returns

none

Preconditions

none

Overview

Sets palette register.

Syntax

SetPalette(colorNum, color)

10.2.3 ResetDevice Function

File

DisplayDriver.h

C

```
void ResetDevice();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Initializes LCD module.

Syntax

void ResetDevice()

10.3 Adding New Device Driver

This is a summary of the requirements to add a new device driver.

Module

Display Device Driver Layer (see page 373)

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 (see page 396)()	Initializes the display device.
GetMaxX (see page 377)()	Returns the maximum x-coordinate for the display.
GetMaxY (see page 377)()	Returns the maximum y-coordinate for the display.
SetColor (see page 376)()	Sets the current drawing color.
GetColor (see page 376)()	Returns the current drawing color.
SetActivePage (see page 387)()	Sets the current active graphic page (optional).
SetVisualPage (see page 388)()	Sets the current visual graphic page (optional).
PutPixel (see page 375)()	Modifies pixel on the screen.
GetPixel (see page 374)()	Returns the pixel color.
PutImage (see page 362)()	Renders an image on the screen. This function is dependent on the color format used.
SetClipRgn (see page 378)()	Set the current clipping region borders.
GetClipLeft (see page 379)(), GetClipTop (see page 380)(), GetClipRight (see page 380)(), GetClipBottom (see page 379)()	Returns the left, top, right and bottom clipping borders.
SetClip (see page 378)()	Enables or disables the clipping region.
IsDeviceBusy (see page 395)()	Checks if the display controller is busy executing the previous rendering operation.
SetPalette (see page 396)()	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.

11 Advanced Display Driver Features

This section lists advanced Display Device Driver Features implemented in select Display Device Driver.

Functions

	Name	Description
	CopyPageWindow (see page 441)	This function alphablends two source windows to a destination window

Structs, Records, Enums

Name	Description
GFX_PAGE (see page 442)	This is type GFX_PAGE.
GFX_PAGE_INFO (see page 443)	This is type GFX_PAGE_INFO.
GFX_WINDOW_INFO (see page 443)	This is type GFX_WINDOW_INFO.

Variables

Name	Description
_bottom (see page 443)	This is variable _bottom.
_delay_ms (see page 444)	This is variable _delay_ms.
_destpageaddr (see page 444)	This is variable _destpageaddr.
_left (see page 444)	This is variable _left.
_page (see page 445)	Current Page
_param1 (see page 445)	This is variable _param1.
_param2 (see page 445)	This is variable _param2.
_right (see page 445)	This is variable _right.
_srcpageaddr (see page 446)	This is variable _srcpageaddr.
_top (see page 446)	This is variable _top.
_transitionpending (see page 446)	This is variable _transitionpending.
_type (see page 447)	This is variable _type.
Height (see page 447)	This is variable Height.
pixelWindow (see page 447)	This is variable pixelWindow.
Startx (see page 448)	This is variable Startx.
Starty (see page 448)	This is variable Starty.
Width (see page 448)	This is variable Width.

11.1 Alpha Blending

The following APIs are used to implement alpha blending features in the Epson S1D13517 Controller.

Functions

	Name	Description
	AlphaBlendWindow (see page 400)	This alphablends a foreground and a background stored in frames to a destination window. The function uses windows insides frames. Each window shares the same width and height parameters.
	GetPositionAddress (see page 401)	This is function GetPositionAddress.

	GFXGetPageOriginAddress (see page 402)	This function calculates the address of a certain 0,0 location of a page in memory
	GFXGetPageXYAddress (see page 402)	This function calculates the address of a certain x,y location in memory
	GFXGetWindowOriginAddress (see page 403)	
	GFXGetWindowXYAddress (see page 403)	

Module

Advanced Display Driver Features (see page 399)

Description

Alpha Blending has certain definitions and assumptions to be used correctly. Here is a brief overview of them. A page is considered an entire graphics resolution frame. So for QVGA a page would be 320x240. A window is a certain width and height contained inside a page.

Alpha blending is a simple function, but requires a lot of CPU bandwith. Some graphics controllers (like the S1D13517) have alpha blending as an accelerated function to offload CPU rednering time. For the Microchips Graphics solution two input sources are blended together and then sent to an output window. Alpha blending is done on a pixel by pixel basis given the window sizes. A sample alpha blending equation used in the epson controller hardware is:

Alpha Blended color = Foreground Color * (Alpha) + Background Color * (1-Alpha)

An alpha blending diagram is shown below.

11.1.1 AlphaBlendWindow Function

File

Primitive.h (see page 1008)

C

```
void AlphaBlendWindow(
    DWORD foregroundWindowAddr,
    DWORD backgroundWindowAddr,
    DWORD destinationWindowAddr,
    WORD width,
    WORD height,
    BYTE alphaPercentage
);
```

Input Parameters

Input Parameters	Description
DWORD foregroundWindowAddr	the starting address of the foreground window
DWORD backgroundWindowAddr	the starting address of the background window
DWORD destinationWindowAddr	the starting address of the destination window
WORD width	the width of the alpha blend window
WORD height	the height of the alpha blend window
BYTE alphaPercentage	the amount of transparency to give the foreground Window (see page 278)

Side Effects

none

Returns

none

Notes

none

Preconditions

none

Overview

This alphablends a foreground and a background stored in frames to a destination window. The function uses windows insides frames. Each window shares the same width and height parameters.

Syntax

void AlphaBlendWindow(DWORD foregroundWindowAddr, DWORD backgroundWindowAddr, DWORD destinationWindowAddr, WORD width, WORD height, BYTE alphaPercentage)

11.1.2 GetPositionAddress Function

File

S1D13517.c

C

```
DWORD GetPositionAddress(
    SHORT page,
    WORD left,
    WORD top
);
```

Description

This is function GetPositionAddress.

11.1.3 GFXGetPageOriginAddress Function

File

Primitive.h (see page 1008)

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)

11.1.4 GFXGetPageXYAddress Function

File

Primitive.h (see page 1008)

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)

11.1.5 GFXGetWindowOriginAddress Function

File

S1D13517.c

C

```
DWORD GFXGetWindowOriginAddress(
    GFX_WINDOW_INFO* GFXWindowInfo
);
```

Syntax

DWORD GFXGetWindowOriginAddress(GFX_WINDOW_INFO (see page 443)* GFXWindowInfo)

11.1.6 GFXGetWindowXYAddress Function

File

S1D13517.c

C

```
DWORD GFXGetWindowXYAddress(
    GFX_WINDOW_INFO* GFXWindowInfo,
    WORD relX,
    WORD relY
);
```

Syntax

DWORD GFXGetWindowXYAddress(GFX_WINDOW_INFO (see page 443)* GFXWindowInfo, WORD relX, WORD relY)

11.2 Transitions

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 (see page 406)(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 (see page 297)() 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 (see page 405)(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.

Functions

	Name	Description
	GFXExecutePendingTransition (see page 405)	This function executes the pending transition and returns the value returned by GFXTransition (see page 406)(). (Used with DOUBLE_BUFFERING)
	GFXIsTransitionPending (see page 405)	This function returns the status if there is any pending transition waiting to be executed. (Used with DOUBLE_BUFFERING)
	GFXSetupTransition (see page 405)	This function saves the parameters to be used and marks the transition to be pending which is executed when GFXExecutePendingTransition (see page 405)() is called. (Used with DOUBLE_BUFFERING)
	GFXTransition (see page 406)	This function saves the parameters to be used and marks the transition to be pending which is executed when GFXExecutePendingTransition (see page 405)() is called
	GFXTranslateDirection (see page 406)	This function returns the actual direction taking the screen orientation into consideration (local function)

Module

Advanced Display Driver Features (see page 399)

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 (see page 406)(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 (see page 297)() 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 (see page

405)(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.

11.2.1 GFXExecutePendingTransition Function

File

Transitions.h

C

```
BYTE GFXExecutePendingTransition(
    DWORD srcpageaddr,
    DWORD destpageaddr
);
```

Returns

See the output of GFXTransition (see page 406)()

Overview

This function executes the pending transition and returns the value returned by GFXTransition (see page 406)(). (Used with DOUBLE_BUFFERING)

Syntax

BYTE GFXExecutePendingTransition(DWORD srcpageaddr, DWORD destpageaddr)

11.2.2 GFXIsTransitionPending Function

File

Transitions.h

C

```
BYTE GFXIsTransitionPending();
```

Returns

0 -> No 1 -> Yes

Overview

This function returns the status if there is any pending transition waiting to be executed. (Used with DOUBLE_BUFFERING)

Syntax

BYTE GFXIsTransitionPending(void)

11.2.3 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
);
```

Returns

0 -> Parameters successfully saved for the new transition -1 -> Parameters not saved as a transition is still pending

Overview

This function saves the parameters to be used and marks the transition to be pending which is executed when GFXExecutePendingTransition (see page 405)() is called. (Used with DOUBLE_BUFFERING)

Syntax

BYTE GFXSetupTransition(SHORT left, SHORT top, SHORT right, SHORT bottom, GFX_TRANSITION_TYPE type, WORD delay_ms, WORD param1, WORD param2)

11.2.4 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
);
```

Returns

0 -> Transition executed successfully -1 -> Transition not executed

Overview

This function saves the parameters to be used and marks the transition to be pending which is executed when GFXExecutePendingTransition (see page 405)() is called

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)

11.2.5 GFXTranslateDirection Function

File

Transitions.h

C

```
WORD GFXTranslateDirection(
    WORD direction,
    WORD orientation
);
```

Returns

The new direction taking the screen orientation into consideration

Overview

This function returns the actual direction taking the screen orientation into consideration (local function)

Syntax

WORD GFXTranslateDirection(WORD direction, WORD orientation)

11.3 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 (see page 399)

11.3.1 Rectangle Copy Operations

The following APIs are used move blocks of data from one memory location to another.

Functions

	Name	Description
	ROPBlock (see page 417)	Performs a Raster Operation (ROP) on source and destination. The type of ROP is decided by the rop and the copyOp parameter.
	Scroll (see page 419)	Scrolls the rectangular area defined by left, top, right, bottom by delta pixels.

Macros

Name	Description
CopyBlock (see page 419)	Copies a block of data from source specified by srcAddr and srcOffset to the destination specified by dstAddr and dstOffset.
RCC_SRC_ADDR_CONTINUOUS (see page 420)	Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (see page 353) 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 (see page 420) - source data is discontinuous... more (see page 420)

RCC_SRC_ADDR_DISCONTINUOUS (see page 420)	Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (see page 353) 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 (see page 420) - source data is discontinuous... more (see page 420)
RCC_DEST_ADDR_CONTINUOUS (see page 420)	Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (see page 353) 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 (see page 420) - source data is discontinuous... more (see page 420)
RCC_DEST_ADDR_DISCONTINUOUS (see page 420)	Source (S) and Destination (D) data type. When performing executing commands on the Rectangle (see page 353) 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 (see page 420) - source data is discontinuous... more (see page 420)
RCC_ROP_0 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_1 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_2 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_3 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_4 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_5 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_6 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_7 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_8 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_9 (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_A (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_B (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_C (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_D (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_ROP_E (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)

RCC_ROP_F (see page 421)	Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389)) • RCC_ROP_1 (see page 421) - not (S or D) • RCC_ROP_2 (see page 421) - (not S) and D • RCC_ROP_3 (see page 421) - not (S) • RCC_ROP_4 (see page 421) - S and not (D)) • RCC_ROP_5 (see page 421) - not (D) • RCC_ROP_6 (see page 421) - S xor D • RCC_ROP_7 (see page 421) - not (S and D) • RCC_ROP_8 (see page 421) - S and D • RCC_ROP_9 (see page 421) - not (S xor D) • RCC_ROP_A (see page 421) - D • RCC_ROP_B (see page 421)... more (see page 421)
RCC_COPY (see page 421)	Type of Rectangle (see page 353) 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 (see page 421) - Fills the specified rectangle with the current color set. • RCC_TRANSPARENT_COPY (see page 421) - 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 (see page 421)
RCC_SOLID_FILL (see page 421)	Type of Rectangle (see page 353) 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 (see page 421) - Fills the specified rectangle with the current color set. • RCC_TRANSPARENT_COPY (see page 421) - 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 (see page 421)

RCC_TRANSPARENT_COPY (see page 421)	Type of Rectangle (see page 353) 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 (see page 421) - Fills the specified rectangle with the current color set. • RCC_TRANSPARENT_COPY (see page 421) - 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 (see page 421)

11.3.1.1 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 (RCC_SRC_ADDR_CONTINUOUS (see page 420) or RCC_SRC_ADDR_DISCONTINUOUS (see page 420))
WORD dstType	sets the source type (RCC_DEST_ADDR_CONTINUOUS (see page 420) or RCC_DEST_ADDR_DISCONTINUOUS (see page 420))
WORD copyOp	sets the type of copy operation • RCC_SOLID_FILL (see page 421): Solid fill of the set color • RCC_COPY (see page 421): direct copy of source to destination • RCC_TRANSPARENT_COPY (see page 421): copy with transparency. Transparency color is set by color

WORD rop	sets the raster operation equation • RCC_ROP_0 (see page 421): Solid black color fill • RCC_ROP_1 (see page 421): not (Source or Destination) • RCC_ROP_2 (see page 421): (not Source) and Destination • RCC_ROP_3 (see page 421): not Source • RCC_ROP_4 (see page 421): Source and (not Destination) • RCC_ROP_5 (see page 421): not Destination • RCC_ROP_6 (see page 421): Source xor Destination • RCC_ROP_7 (see page 421): not (Source and Destination) • RCC_ROP_8 (see page 421): Source and Destination • RCC_ROP_9 (see page 421): not (Source xor Destination) • RCC_ROP_A (see page 421): Destination • RCC_ROP_B (see page 421): (not Source) or Destination • RCC_ROP_C (see page 421): Source • RCC_ROP_D (see page 421): Source or (not Destination) • RCC_ROP_E (see page 421): Source or Destination • RCC_ROP_F (see page 421): 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)

11.3.1.2 CopyBlock Macro

File

mchpGfxDrv.h

C

```
#define CopyBlock(srcAddr, dstAddr, srcOffset, dstOffset, width, height) \
                                                                          \
        ROPBlock( srcAddr, dstAddr, srcOffset, dstOffset,                  \
                  (RCC_SRC_ADDR_DISCONTINUOUS), (RCC_DEST_ADDR_DISCONTINUOUS),  \
                  RCC_COPY,  RCC_ROP_C,                                    \
                  0,  width, height)
```

Input Parameters

Input Parameters	Description
srcAddr	the base address of the data to be moved
dstAddr	the base address of the new location of the moved data
srcOffset	offset of the data to be moved with respect to the source base address.
dstOffset	offset of the new location of the moved data respect to the source base address.
width	width of the block of data to be moved
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.

Syntax

WORD CopyBlock(DWORD srcAddr, DWORD dstAddr, DWORD srcOffset, DWORD dstOffset, WORD width, WORD height)

11.3.1.3 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. • 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)

11.3.1.4 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 (see page 353) 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

11.3.1.5 RCC_ROP_0 Macro

File

mchpGfxDrv.h

C

```
#define RCC_ROP_0 0x00000000ul
#define RCC_ROP_1 0x00000008ul        // not (S or D)
#define RCC_ROP_2 0x00000010ul        // (not S) and D
#define RCC_ROP_3 0x00000018ul        // not (S)
#define RCC_ROP_4 0x00000020ul        // S and not (D)
#define RCC_ROP_5 0x00000028ul        // not (D)
#define RCC_ROP_6 0x00000030ul        // S xor D
#define RCC_ROP_7 0x00000038ul        // not (S and D)
#define RCC_ROP_8 0x00000040ul        // S and D
#define RCC_ROP_9 0x00000048ul        // not (S xor D)
#define RCC_ROP_A 0x00000050ul        // D
#define RCC_ROP_B 0x00000058ul        // not (S) or D
#define RCC_ROP_C 0x00000060ul        // S
#define RCC_ROP_D 0x00000068ul        // S or not (D)
#define RCC_ROP_E 0x00000070ul        // S or D
#define RCC_ROP_F 0x00000078ul        // 1 (WHITE)
```

Overview

Raster Operation (ROP) option. Select one of the following 16 raster operation options whenever Rectangle (see page 353) 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 (see page 389))
- 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 (see page 394))

11.3.1.6 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 (see page 353) 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

11.3.2 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).

Functions

	Name	Description
	SwitchOffDoubleBuffering (see page 424)	Switches off the double buffering. All rendering will be performed on the frame buffer. Calls to UpdateDisplayNow (see page 426)() or RequestDisplayUpdate (see page 425)() will have no effect.
	SwitchOnDoubleBuffering (see page 424)	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 (see page 426)() or RequestDisplayUpdate (see page 425)().
	InvalidateRectangle (see page 425)	Invalidates 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.
	RequestDisplayUpdate (see page 425)	Synchronizes the draw and frame buffers at next VBlank
	UpdateDisplayNow (see page 426)	Synchronizes the draw and frame buffers immediately.

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 and can cause user dissatisfaction. 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 workflow 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 (see page 928)

2. Update GFX_DISPLAY_BUFFER_LENGTH to include both the buffers in HardwareProfile.h

3. Check the jumper settings to enable the required RAM address space on the development board

If Graphics Objects Layer (GOL (see page 48)) 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 (see page 367)() is called, call the APIs InvalidateAll() followed by UpdateDisplayNow (see page 426)(). 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 (see page 425)(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 (see page 425)() or UpdateDisplayNow (see page 426)() 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 (see page 425)() is same as that of UpdateDisplayNow (see page 426)().

Even if double buffering is enabled at compile time, it can be switched off and on at run time using APIs SwitchOffDoubleBuffering (see page 424)() and SwitchOnDoubleBuffering (see page 424)(). 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 (see page 306)() 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 (see page 306)().

11.3.2.1 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 (see page 426)() or RequestDisplayUpdate (see page 425)() will have no effect.

Syntax

SwitchOffDoubleBuffering()

11.3.2.2 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 (see page 426)() or RequestDisplayUpdate (see page 425)().

Syntax

SwitchOnDoubleBuffering()

11.3.2.3 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

Invalidates 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)

11.3.2.4 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)

11.3.2.5 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)

11.3.3 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 (see page 426)	Decompresses the nbytes number of data at SrcAddress and places starting from DestAddress. (Blocking)

11.3.3.1 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);

11.3.4 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 (see page 435)	This is file Palette.h.

Functions

	Name	Description
	ClearPaletteChangeError (see page 428)	Clears the Palette change error status
	DisablePalette (see page 428)	Disables the Palette mode.
	EnablePalette (see page 429)	Enables the Palette mode.
	GetPaletteChangeError (see page 429)	Returns the Palette change error status
	IsPaletteEnabled (see page 430)	Returns if the Palette mode is enabled or not.
	PaletteInit (see page 430)	Initializes the color look up table (CLUT).
	RequestPaletteChange (see page 430)	Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.
	SetPalette (see page 431)	Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.
	SetPaletteBpp (see page 432)	Sets the color look up table (CLUT) number of valid entries.
	SetPaletteFlash (see page 432)	Loads the palettes from the flash immediately.

Macros

Name	Description
RequestEntirePaletteChange (see page 434)	Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.
SetEntirePalette (see page 435)	Programs the whole 256 entry palette with new color values from flash.

Structures

Name	Description
PALETTE_FLASH (see page 433)	Structure for the palette stored in FLASH memory.
PALETTE_HEADER (see page 434)	Structure for the palette header.

Types

Name	Description
PALETTE_EXTERNAL (see page 434)	Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Unions

Name	Description
PALETTE_ENTRY (see page 433)	Structure used for the palette entries. • For TFT: color is defined as 5-6-5 RGB format (5-bits for RED (see page 394), 6-bits for GREEN (see page 392) and 5-bits for BLUE (see page 389). • For Monochrome: 4 bits are used to represent the luma.

11.3.4.1 ClearPaletteChangeError Function

File

Palette.h (see page 435)

C

```
void ClearPaletteChangeError();
```

Side Effects

none

Returns

none

Preconditions

none

Overview

Clears the Palette change error status

Syntax

void ClearPaletteChangeError(void)

11.3.4.2 DisablePalette Function

File

Palette.h (see page 435)

C

```
void DisablePalette();
```

Returns

none

Preconditions

none

Overview

Disables the Palette mode.

Syntax

void DisablePalette(void)

11.3.4.3 EnablePalette Function

File

Palette.h (see page 435)

C

```
void EnablePalette();
```

Returns

none

Preconditions

none

Overview

Enables the Palette mode.

Syntax

void EnablePalette(void)

11.3.4.4 GetPaletteChangeError Function

File

Palette.h (see page 435)

C

```
BYTE GetPaletteChangeError();
```

Side Effects

none

Returns

Returns the palette change status. 1 - If the palette change error occured 0 - If no palette change error occured

Preconditions

none

Overview

Returns the Palette change error status

Syntax

BYTE GetPaletteChangeError(void)

11.3.4.5 IsPaletteEnabled Function

File

Palette.h (see page 435)

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)

11.3.4.6 PaletteInit Function

File

Palette.h (see page 435)

C

```
void PaletteInit();
```

Side Effects

All rendering will use the new palette entries.

Returns

none

Preconditions

none

Overview

Initializes the color look up table (CLUT).

Syntax

void PaletteInit(void)

11.3.4.7 RequestPaletteChange Function

File

Palette.h (see page 435)

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 (see page 430)().

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)

11.3.4.8 SetPalette Function

File

Palette.h (see page 435)

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 (see page 430)().

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)

11.3.4.9 SetPaletteBpp Function

File

Palette.h (see page 435)

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 (see page 430)().

Overview

Sets the color look up table (CLUT) number of valid entries.

Syntax

BYTE SetPaletteBpp(BYTE bpp)

11.3.4.10 SetPaletteFlash Function

File

Palette.h (see page 435)

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 (see page 430)().

Overview

Loads the palettes from the flash immediately.

Syntax

BYTE SetPaletteFlash(PALETTE_ENTRY (see page 433) *pPaletteEntry, WORD startEntry, WORD length)

11.3.4.11 PALETTE_ENTRY Union

File

Palette.h (see page 435)

C

```
typedef union {
  WORD value;
  struct {
    BYTE r : 5;
    BYTE g : 6;
    BYTE b : 5;
  } color;
  struct {
    BYTE luma : 4;
  } monchrome;
} PALETTE_ENTRY;
```

Members

Members	Description
WORD value;	a 16-bit value representing a color or palette entry
struct { BYTE r : 5; BYTE g : 6; BYTE b : 5; } color;	color value in 5-6-5 RGB format
BYTE r : 5;	represents the RED (see page 394) component
BYTE g : 6;	represents the GREEN (see page 392) component
BYTE b : 5;	represents the BLUE (see page 389) component
struct { BYTE luma : 4; } monchrome;	monochrome LUMA value
BYTE luma : 4;	monochrome LUMA value

Overview

Structure used for the palette entries.

- For TFT: color is defined as 5-6-5 RGB format (5-bits for RED (see page 394), 6-bits for GREEN (see page 392) and 5-bits for BLUE (see page 389).
- For Monochrome: 4 bits are used to represent the luma.

11.3.4.12 PALETTE_FLASH Structure

File

Palette.h (see page 435)

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.

11.3.4.13 PALETTE_HEADER Structure

File

Palette.h (see page 435)

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.

11.3.4.14 PALETTE_EXTERNAL Type

File

Palette.h (see page 435)

C

```
typedef GFX_EXTDATA PALETTE_EXTERNAL;
```

Overview

Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

11.3.4.15 RequestEntirePaletteChange Macro

File

Palette.h (see page 435)

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 PaletteInit (see page 430)().

Overview

Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.

Syntax

RequestEntirePaletteChange(pPalette)

11.3.4.16 SetEntirePalette Macro

File

Palette.h (see page 435)

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 PaletteInit (see page 430)().

Overview

Programs the whole 256 entry palette with new color values from flash.

Syntax

SetEntirePalette(pPalette)

11.3.4.17 Palette.h

Functions

	Name	Description
	ClearPaletteChangeError (see page 428)	Clears the Palette change error status

	DisablePalette (see page 428)	Disables the Palette mode.
	EnablePalette (see page 429)	Enables the Palette mode.
	GetPaletteChangeError (see page 429)	Returns the Palette change error status
	IsPaletteEnabled (see page 430)	Returns if the Palette mode is enabled or not.
	PaletteInit (see page 430)	Initializes the color look up table (CLUT).
	RequestPaletteChange (see page 430)	Loads the palettes from the flash during vertical blanking period if possible, otherwise loads immediately.
	SetPalette (see page 431)	Programs a block of palette entries starting from startEntry and until startEntry + length from the flash immediately.
	SetPaletteBpp (see page 432)	Sets the color look up table (CLUT) number of valid entries.
	SetPaletteFlash (see page 432)	Loads the palettes from the flash immediately.

Macros

Name	Description
RequestEntirePaletteChange (see page 434)	Loads all the palette entries from the flash during vertical blanking period if possible, otherwise loads immediately.
SetEntirePalette (see page 435)	Programs the whole 256 entry palette with new color values from flash.

Structures

Name	Description
PALETTE_FLASH (see page 433)	Structure for the palette stored in FLASH memory.
PALETTE_HEADER (see page 434)	Structure for the palette header.

Types

Name	Description
PALETTE_EXTERNAL (see page 434)	Structure for palette stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Unions

Name	Description
PALETTE_ENTRY (see page 433)	Structure used for the palette entries. • For TFT: color is defined as 5-6-5 RGB format (5-bits for RED (see page 394), 6-bits for GREEN (see page 392) and 5-bits for BLUE (see page 389). • For Monochrome: 4 bits are used to represent the luma.

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.
 ****************************************************************************/
#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
{
    WORD    value;                      // a 16-bit value representing a color or palette
entry
    struct
    {
        BYTE    r : 5;                  // represents the RED component
        BYTE    g : 6;                  // represents the GREEN component
        BYTE    b : 5;                  // represents the BLUE component
    } color;                            // color value in 5-6-5 RGB format

    struct
    {
        BYTE    luma : 4;               // monochrome LUMA value
    } monchrome;                        // monochrome LUMA value

} 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 occured
*         0 - If no palette change error occured
*
* 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
```

11.4 Functions

11.4.1 CopyPageWindow Function

File

S1D13517.h

C

```
void CopyPageWindow(
    BYTE srcPage,
    BYTE dstPage,
    WORD srcX,
    WORD srcY,
    WORD dstX,
    WORD dstY,
    WORD width,
    WORD height
);
```

Module

Advanced Display Driver Features (see page 399)

Input Parameters

Input Parameters	Description
BYTE srcPage	page of the source to be copied
BYTE dstPage	page of the destination to paste to
WORD srcX	x offset of the page
WORD srcY	y offset of the page

WORD dstX	x offset of the page
WORD dstY	y offset of the page
WORD width	width of the windows
WORD height	height of the windows

Side Effects

none

Returns

- none

Overview

This function alphablends two source windows to a destination window

Syntax

void CopyPageWindow(BYTE srcPage, BYTE dstPage, WORD srcX, WORD srcY, WORD dstX, WORD dstY, WORD width, WORD height)

11.5 Structs, Records, Enums

11.5.1 GFX_PAGE Enumeration

File

S1D13517.h

C

```
typedef enum {
  GFX_PAGE0 = 0,
  GFX_PAGE1,
  GFX_PAGE2,
  GFX_PAGE3,
  GFX_PAGE4,
  GFX_PAGE5,
  GFX_PAGE6,
  GFX_PAGE7,
  GFX_PAGE8,
  GFX_PAGE9,
  GFX_PAGE10,
  GFX_PAGE11,
  GFX_PAGE12,
  GFX_PAGE13,
  GFX_PAGE14,
  GFX_PAGE15,
  GFX_NUM_OF_PAGES
} GFX_PAGE;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is type GFX_PAGE.

11.5.2 GFX_PAGE_INFO Structure

File

S1D13517.c

C

```
typedef struct {
  DWORD page0BaseAddress;
} GFX_PAGE_INFO;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is type GFX_PAGE_INFO.

11.5.3 GFX_WINDOW_INFO Structure

File

S1D13517.c

C

```
typedef struct {
  GFX_PAGE windowPage;
  WORD xOrigin;
  WORD yOrigin;
  WORD width;
  WORD height;
} GFX_WINDOW_INFO;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is type GFX_WINDOW_INFO.

11.6 Variables

11.6.1 _bottom Variable

File

S1D13517.c

C

```
WORD _bottom;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _bottom.

11.6.2 _delay_ms Variable

File

S1D13517.c

C

```
WORD _delay_ms;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _delay_ms.

11.6.3 _destpageaddr Variable

File

S1D13517.c

C

```
DWORD _destpageaddr;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _destpageaddr.

11.6.4 _left Variable

File

S1D13517.c

C

```
WORD _left;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _left.

11.6.5 _page Variable

File

S1D13517.c

C

```
WORD _page;
```

Module

Advanced Display Driver Features (see page 399)

Description

Current Page

11.6.6 _param1 Variable

File

S1D13517.c

C

```
WORD _param1;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _param1.

11.6.7 _param2 Variable

File

S1D13517.c

C

```
WORD _param2;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _param2.

11.6.8 _right Variable

File

S1D13517.c

C

```
WORD _right;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _right.

11.6.9 _srcpageaddr Variable

File

S1D13517.c

C

```
DWORD _srcpageaddr;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _srcpageaddr.

11.6.10 _top Variable

File

S1D13517.c

C

```
WORD _top;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _top.

11.6.11 _transitionpending Variable

File

S1D13517.c

C

```
WORD _transitionpending;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _transitionpending.

11.6.12 _type Variable

File

S1D13517.c

C

```
WORD _type;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable _type.

11.6.13 Height Variable

File

S1D13517.c

C

```
WORD Height;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable Height.

11.6.14 pixelWindow Variable

File

S1D13517.c

C

```
GFX_COLOR pixelWindow[8] = { TRANSPARENTCOLOR, TRANSPARENTCOLOR, TRANSPARENTCOLOR,
TRANSPARENTCOLOR, TRANSPARENTCOLOR, TRANSPARENTCOLOR, TRANSPARENTCOLOR, TRANSPARENTCOLOR };
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable pixelWindow.

11.6.15 Startx Variable

File

S1D13517.c

C

```
WORD Startx;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable Startx.

11.6.16 Starty Variable

File

S1D13517.c

C

```
WORD Starty;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable Starty.

11.6.17 Width Variable

File

S1D13517.c

C

```
WORD Width;
```

Module

Advanced Display Driver Features (see page 399)

Description

This is variable Width.

12 Image Decoders

Demo Project

Name	Description
Image Decoder Demo (see page 457)	This demo demonstrates the decoding of images with JPEG and BMP file formats.

Functions

	Name	Description
	ImageDecode (see page 452)	This function decodes and displays the image on the screen
	ImageDecoderInit (see page 453)	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 (see page 454)	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 maintainance activities such as fetching data, updating the display, etc...
	ImageDecodeTask (see page 454)	This function completes one small part of the image decode function

Macros

Name	Description
ImageFullScreenDecode (see page 455)	This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required
ImageAbort (see page 456)	This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.

Structures

	Name	Description
	_BMPDECODER (see page 512)	DATA STRUCTURES
	_GIFDECODER (see page 512)	DATA STRUCTURES
	_JPEGDECODER (see page 513)	DATA STRUCTURES

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 (see page 480)
- Individual decoder headers
- JpegDecoder.h (see page 505)
- BmpDecoder.h (see page 464)
- GifDecoder.h (see page 475)

3. Source files

- Main Source file
- ImageDecoder.c (see page 476)
- Individual Source files
- JpegDecoder.c (see page 487)
- BmpDecoder.c (see page 457)
- GifDecoder.c (see page 464)

4. Support files

- jidctint.c (see page 507)

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 (see page 480)) 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 (see page 475) file. Enabling this will have an additional effect on the GIF decoding time.

12.1 ImageDecode Function

File

ImageDecoder.h (see page 480)

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
);
```

Module

Image Decoders (see page 449)

Side Effects

None

Returns

Error code -> 0 means no error

Example

```
void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
            <- Error handling ->
    }
    IMG_bDecode(pImageFile, IMG_JPEG, 0, 0, 320, 240, 0, NULL, NULL);
    IMG_FCLOSE(pImageFile);
    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)

12.2 ImageDecoderInit Function

File

ImageDecoder.h (see page 480)

C

```
void ImageDecoderInit();
```

Module

Image Decoders (see page 449)

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)

12.3 ImageLoopCallbackRegister Function

File

ImageDecoder.h (see page 480)

C

```
void ImageLoopCallbackRegister(
    IMG_LOOP_CALLBACK pFn
);
```

Module

Image Decoders (see page 449)

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 maintainance activities such as fetching data, updating the display, etc...

Syntax

void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pLoopCallbackFn)

12.4 ImageDecodeTask Function

File

ImageDecoder.h (see page 480)

C

```
BYTE ImageDecodeTask();
```

Module

Image Decoders (see page 449)

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)

12.5 ImageFullScreenDecode Macro

File

ImageDecoder.h (see page 480)

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)
```

Module

Image Decoders (see page 449)

Side Effects

None

Returns

Error code -> 0 means no error

Example

```
void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
            <- Error handling ->
    }
    IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(pImageFile);
    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 *pImageFile, IMG_FILE_FORMAT eImgFormat, IMG_FILE_SYSTEM_API pFileAPIs, IMG_PIXEL_OUTPUT pPixelOutput)

12.6 ImageAbort Macro

File

ImageDecoder.h (see page 480)

C

```
#define ImageAbort IMG_blAbortImageDecoding = 1;
```

Module

Image Decoders (see page 449)

Side Effects

None

Returns

None

Example

```
void callback(void);
void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    ImageLoopCallbackRegister(callback);
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
            <- Error handling ->
    }
    IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(pImageFile);
    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)

12.7 Demo Project

12.7.1 Image Decoder Demo

This demo demonstrates the decoding of images with JPEG and BMP file formats.

Module

Image Decoders (see page 449)

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.

12.8 Image Decoders Files

Files

Name	Description
BmpDecoder.c (see page 457)	This is file BmpDecoder.c.
BmpDecoder.h (see page 464)	This is file BmpDecoder.h.
GifDecoder.c (see page 464)	This is file GifDecoder.c.
GifDecoder.h (see page 475)	This is file GifDecoder.h.
ImageDecoder.c (see page 476)	This is file ImageDecoder.c.
ImageDecoder.h (see page 480)	This is file ImageDecoder.h.
JpegDecoder.c (see page 487)	This is file JpegDecoder.c.
JpegDecoder.h (see page 505)	This is file JpegDecoder.h.
jidctint.c (see page 507)	This is file jidctint.c.

Module

Image Decoders (see page 449)

Description

Lists all files describing the Image Decoders.

12.8.1 BmpDecoder.c

Structures

	Name	Description
	_BMPDECODER (see page 512)	DATA STRUCTURES

Description

This is file BmpDecoder.c.

Body Source

```
#define __BMPDECODER_C__
/*****************************************************************************

* FileName:        BmpDecoder.c
* Dependencies:    Image decoding library; project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
* Compiler:        C30 v2.01/C32 v0.00.18
* Company:         Microchip Technology, Inc.

 * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
*******************************************************************************/

#include "Image Decoders/ImageDecoder.h"

#ifdef IMG_SUPPORT_BMP

/*************************/
/**** DATA STRUCTURES ****/
/*************************/
typedef struct _BMPDECODER
{
        IMG_FILE *pImageFile;                       /* Image file pointer */
        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]; /* Each palette entry has RGB */
} BMPDECODER;

/*************************/
/******* FUNCTIONS  *******/
/*************************/

/*****************************************************************************
```

```
Function:       void BDEC_vResetData(BMPDECODER *pBmpDec)

Precondition:   None

Overview:       This function resets the variables so that new Bitmap image
                can be decoded

Input:          Bitmap decoder's data structure

Output:         None
***************************************************************************/
void BDEC_vResetData(BMPDECODER *pBmpDec)
{
    pBmpDec->pImageFile = NULL;
    pBmpDec->lWidth = 0;
    pBmpDec->lHeight = 0;
    pBmpDec->lImageOffset = 0;
    pBmpDec->wPaletteEntries = 0;
    pBmpDec->bBitsPerPixel = 0;
    pBmpDec->bHeaderType = 0;
    pBmpDec->blBmMarkerFlag = 0;
    pBmpDec->blCompressionType = 0;
    pBmpDec->bNumOfPlanes = 0;
    pBmpDec->b16bit565flag = 0;
}

/***************************************************************************
Function:       BYTE BDEC_bReadHeader(BMPDECODER *pBmpDec)

Precondition:   None

Overview:       This function reads the Bitmap file header and
                fills the data structure

Input:          Bitmap decoder's data structure

Output:         Error code - '0' means no error
***************************************************************************/
BYTE BDEC_bReadHeader(BMPDECODER *pBmpDec)
{
        BYTE bByte1, bByte2;
        WORD wWord;
        LONG lLong;

        IMG_FREAD(&bByte1, sizeof(bByte1), 1, pBmpDec->pImageFile);  /* Marker */
        IMG_FREAD(&bByte2, sizeof(bByte2), 1, pBmpDec->pImageFile);  /* Marker */

        if(bByte1 == 'B' && bByte2 == 'M')
        {
                pBmpDec->blBmMarkerFlag = 1;
        }
        else
        {
                return(100);
        }

        IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* File length */
        IMG_FREAD(&wWord, sizeof(wWord), 1, pBmpDec->pImageFile);  /* Reserved */
        IMG_FREAD(&wWord, sizeof(wWord), 1, pBmpDec->pImageFile);  /* Reserved */

        IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Image offset */
        pBmpDec->lImageOffset = lLong;

        IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Header length */
        pBmpDec->bHeaderType = (BYTE)lLong;

        if(pBmpDec->bHeaderType >= 40)
        {
                IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Image Width
*/
                pBmpDec->lWidth = lLong;
```

```
                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Image
Height */
                    pBmpDec->lHeight = lLong;

                    IMG_FREAD(&wWord, sizeof(wWord), 1, pBmpDec->pImageFile);  /* Number of
Planes */
                    pBmpDec->bNumOfPlanes = (BYTE)wWord;

                    IMG_FREAD(&wWord, sizeof(wWord), 1, pBmpDec->pImageFile);  /* Bits per
Pixel */
                    pBmpDec->bBitsPerPixel = (BYTE)wWord;

                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Compression
info */
                    pBmpDec->blCompressionType = (BYTE)lLong;

                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Image
length */
                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* xPixels per
metre */
                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* yPixels per
metre */

                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Palette
entries */
                    pBmpDec->wPaletteEntries = (WORD)lLong;

                    if(pBmpDec->wPaletteEntries == 0)
                    {
                            WORD wTemp = (WORD)(pBmpDec->lImageOffset - 14 - 40)/4;
                            if(wTemp > 0)
                            {
                                     pBmpDec->wPaletteEntries = wTemp; /* This is because of
a bug in MSPAINT */
                            }
                    }

                    IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Important
colors */
                    if(pBmpDec->bBitsPerPixel == 16 && pBmpDec->bHeaderType > 40)
                    {
                            IMG_FREAD(&lLong, sizeof(lLong), 1, pBmpDec->pImageFile);  /* Red
mask */
                            if((WORD)lLong == 0xF800)
                            {
                                      pBmpDec->b16bit565flag = 1;
                            }
                    }

                    IMG_FSEEK(pBmpDec->pImageFile, pBmpDec->bHeaderType + 14, 0);

                    if(pBmpDec->wPaletteEntries <= 256)
                    {
                            WORD wCounter;
                            for(wCounter = 0; wCounter < pBmpDec->wPaletteEntries; wCounter++)
                            {
                                      IMG_FREAD(&pBmpDec->aPalette[wCounter][2],
sizeof(BYTE), 1, pBmpDec->pImageFile); /* R */
                                      IMG_FREAD(&pBmpDec->aPalette[wCounter][1],
sizeof(BYTE), 1, pBmpDec->pImageFile); /* G */
                                      IMG_FREAD(&pBmpDec->aPalette[wCounter][0],
sizeof(BYTE), 1, pBmpDec->pImageFile); /* B */
                                      IMG_FREAD(&wWord, sizeof(BYTE), 1,
pBmpDec->pImageFile); /* Dummy */
                            }
                    }
            }
            return(0);
}

/*****************************************************************************
Function:       BYTE BMP_bDecode(IMG_FILE *pFile)
```

```
  Precondition:    None

  Overview:        This function decodes and displays a Bitmap image

  Input:           Image file

  Output:          Error code - '0' means no error
*************************************************************************/
BYTE BMP_bDecode(IMG_FILE *pFile)
{
        BMPDECODER BmpDec;
        WORD wX, wY;
        BYTE bPadding;

        BDEC_vResetData(&BmpDec);
        BmpDec.pImageFile = pFile;
        BDEC_bReadHeader(&BmpDec);
        if(BmpDec.blBmMarkerFlag == 0 || BmpDec.bHeaderType < 40 ||
(BmpDec.blCompressionType != 0 && BmpDec.blCompressionType != 3))
        {
            return 100;
        }
        IMG_wImageWidth = (WORD)BmpDec.lWidth;
        IMG_wImageHeight = (WORD)BmpDec.lHeight;
        IMG_vSetboundaries();

        IMG_FSEEK(pFile, BmpDec.lImageOffset, 0);
        if(BmpDec.wPaletteEntries == 0 && BmpDec.bBitsPerPixel == 8) /* Grayscale Image */
        {
                bPadding = (4 - (BmpDec.lWidth % 4))%4;
                for(wY = 0; wY < BmpDec.lHeight; wY++)
                {
                        IMG_vLoopCallback();
                        IMG_vCheckAndAbort();
                        for(wX = 0; wX < BmpDec.lWidth; wX++)
                        {
                                BYTE bY;
                                IMG_FREAD(&bY, sizeof(bY), 1, BmpDec.pImageFile); /* Y */
                                IMG_vSetColor(bY, bY, bY);
                                IMG_vPutPixel(wX, BmpDec.lHeight - wY - 1);
                        }
                        for(wX = 0; wX < bPadding; wX++)
                        {
                                BYTE bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                        }
                }
        }
        else if(BmpDec.bBitsPerPixel == 16) /* 16-bit Color Image */
        {
                bPadding = (4 - ((BmpDec.lWidth*2) % 4))%4;
                for(wY = 0; wY < BmpDec.lHeight; wY++)
                {
                        IMG_vLoopCallback();
                        IMG_vCheckAndAbort();
                        for(wX = 0; wX < BmpDec.lWidth; wX++)
                        {
                                WORD wColor;
                                BYTE bR, bG, bB;
                                IMG_FREAD(&wColor, sizeof(wColor), 1,
BmpDec.pImageFile); /* RGB */
                                if(BmpDec.b16bit565flag == 1)
                                {
                                        bR = (wColor >> 11) << 3;
                                        bG = ((wColor & 0x07E0) >> 5) << 2;
                                        bB = (wColor & 0x001F) << 3;
                                }
                                else
                                {
                                        bR = ((wColor & 0x7FFF) >> 10) << 3;
                                        bG = ((wColor & 0x03E0) >> 5) << 3;
```

```
                                        bB = (wColor & 0x001F) << 3;
                                }
                                IMG_vSetColor(bR, bG, bB);
                                IMG_vPutPixel(wX, BmpDec.lHeight - wY - 1);
                        }
                        for(wX = 0; wX < bPadding; wX++)
                        {
                                BYTE bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                        }
                }
        }
        else if(BmpDec.bBitsPerPixel == 24) /* True color Image */
        {
                bPadding = (4 - ((BmpDec.lWidth*3) % 4))%4;
                for(wY = 0; wY < BmpDec.lHeight; wY++)
                {
                        IMG_vLoopCallback();
                        IMG_vCheckAndAbort();
                        for(wX = 0; wX < BmpDec.lWidth; wX++)
                        {
                                BYTE bR, bG, bB;
                                IMG_FREAD(&bB, sizeof(bB), 1, BmpDec.pImageFile); /* B */
                                IMG_FREAD(&bG, sizeof(bG), 1, BmpDec.pImageFile); /* G */
                                IMG_FREAD(&bR, sizeof(bR), 1, BmpDec.pImageFile); /* R */
                                IMG_vSetColor(bR, bG, bB);
                                IMG_vPutPixel(wX, BmpDec.lHeight - wY - 1);
                        }
                        for(wX = 0; wX < bPadding; wX++)
                        {
                                BYTE bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                        }
                }
        }
        else if(BmpDec.wPaletteEntries != 0 && BmpDec.bBitsPerPixel == 1) /* B/W Image */
        {
                WORD wBytesPerRow = BmpDec.lWidth/8;
                BYTE bAdditionalBitsPerRow = BmpDec.lWidth % 8;
                bPadding = (4 - ((wBytesPerRow + (bAdditionalBitsPerRow?1:0)) % 4))%4;
                for(wY = 0; wY < BmpDec.lHeight; wY++)
                {
                        BYTE bBits, bValue;
                        IMG_vLoopCallback();
                        IMG_vCheckAndAbort();
                        for(wX = 0; wX < wBytesPerRow; wX++)
                        {
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);

                                for(bBits = 0; bBits < 8; bBits++)
                                {
                                        BYTE bIndex = (bValue & (0x80 >> bBits))?1:0;
                                        IMG_vSetColor(BmpDec.aPalette[bIndex][0],
BmpDec.aPalette[bIndex][1], BmpDec.aPalette[bIndex][2]);
                                        IMG_vPutPixel(wX*8+bBits, BmpDec.lHeight - wY
- 1);
                                }
                        }
                        if(bAdditionalBitsPerRow > 0)
                        {
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);

                                for(bBits = 0; bBits < bAdditionalBitsPerRow; bBits++)
                                {
                                        BYTE bIndex = (bValue & (0x80 >> bBits))?1:0;
                                        IMG_vSetColor(BmpDec.aPalette[bIndex][0],
BmpDec.aPalette[bIndex][1], BmpDec.aPalette[bIndex][2]);
                                        IMG_vPutPixel(wX*8+bBits, BmpDec.lHeight - wY
- 1);
                                }
                        }
                        for(wX = 0; wX < bPadding; wX++)
```

```
                        {
                                BYTE bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                        }
                }
        }
        else if(BmpDec.wPaletteEntries != 0 && BmpDec.bBitsPerPixel == 4) /* 16 colors
Image */
        {
                WORD wBytesPerRow = BmpDec.lWidth/2;
                BYTE bAdditionalNibblePerRow = BmpDec.lWidth % 2;
                bPadding = (4 - ((wBytesPerRow + bAdditionalNibblePerRow) % 4))%4;
                for(wY = 0; wY < BmpDec.lHeight; wY++)
                {
                        IMG_vLoopCallback();
                        IMG_vCheckAndAbort();
                        for(wX = 0; wX < wBytesPerRow; wX++)
                        {
                                BYTE bIndex, bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                                bIndex = bValue >> 4;
                                IMG_vSetColor(BmpDec.aPalette[bIndex][0],
BmpDec.aPalette[bIndex][1], BmpDec.aPalette[bIndex][2]);
                                IMG_vPutPixel(wX*2, BmpDec.lHeight - wY - 1);
                                bIndex = bValue & 0x0F;
                                IMG_vSetColor(BmpDec.aPalette[bIndex][0],
BmpDec.aPalette[bIndex][1], BmpDec.aPalette[bIndex][2]);
                                IMG_vPutPixel(wX*2+1, BmpDec.lHeight - wY - 1);
                        }
                        if(bAdditionalNibblePerRow)
                        {
                                BYTE bIndex, bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1,
BmpDec.pImageFile); /* Bits8 */
                                bIndex = bValue >> 4;
                                IMG_vSetColor(BmpDec.aPalette[bIndex][0],
BmpDec.aPalette[bIndex][1], BmpDec.aPalette[bIndex][2]);
                                IMG_vPutPixel(wX*2, BmpDec.lHeight - wY - 1);
                        }
                        for(wX = 0; wX < bPadding; wX++)
                        {
                                BYTE bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                        }
                }
        }
        else if(BmpDec.wPaletteEntries != 0 && BmpDec.bBitsPerPixel == 8) /* 256 colors
Image */
        {
                bPadding = (4 - (BmpDec.lWidth % 4))%4;
                for(wY = 0; wY < BmpDec.lHeight; wY++)
                {
                        IMG_vLoopCallback();
                        IMG_vCheckAndAbort();
                        for(wX = 0; wX < BmpDec.lWidth; wX++)
                        {
                                BYTE bIndex;
                                IMG_FREAD(&bIndex, sizeof(bIndex), 1, BmpDec.pImageFile);
                                IMG_vSetColor(BmpDec.aPalette[bIndex][0],
BmpDec.aPalette[bIndex][1], BmpDec.aPalette[bIndex][2]);
                                IMG_vPutPixel(wX, BmpDec.lHeight - wY - 1);
                        }
                        for(wX = 0; wX < bPadding; wX++)
                        {
                                BYTE bValue;
                                IMG_FREAD(&bValue, sizeof(bValue), 1, BmpDec.pImageFile);
                        }
                }
        }

        return 0;
}
```

```
#endif /* #ifdef IMG_SUPPORT_BMP */

#undef __BMPDECODER_C__
```

12.8.2 BmpDecoder.h

This is file BmpDecoder.h.

Body Source

```
#ifndef __BMPDECODER_H__
 #define __BMPDECODER_H__

/*****************************************************************************
* FileName:        BmpDecoder.h
* Dependencies:    Image decoding library; project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
* Compiler:        C30 v2.01/C32 v0.00.18
* Company:         Microchip Technology, Inc.

 * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
*****************************************************************************/

/* Function prototype */
/* This function must be called after setting proper values in the global variables of
ImageDecoder.c */
BYTE BMP_bDecode(IMG_FILE *pFile);

#endif
```

12.8.3 GifDecoder.c

Structures

	Name	Description
	_GIFDECODER (see page 512)	DATA STRUCTURES

Description

This is file GifDecoder.c.

Body Source

```
#define __GIFDECODER_C__
/*****************************************************************************

* FileName:        GifDecoder.c
* Dependencies:    Image decoding library; project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
* Compiler:        C30 v2.01/C32 v0.00.18
* Company:         Microchip Technology, Inc.

 * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
*****************************************************************************/

#include "Image Decoders/ImageDecoder.h"

#ifdef IMG_SUPPORT_GIF

/*************************/
/**** DATA STRUCTURES ****/
/*************************/
typedef struct _GIFDECODER
{
        IMG_FILE *pImageFile;                       /* Image file pointer */
        WORD wImageWidth;
        WORD wImageHeight;
        WORD wImageX;
        WORD wImageY;
        WORD wScreenWidth;
        WORD wScreenHeight;
        WORD wGlobalPaletteEntries;
        WORD wLocalPaletteEntries;
        BYTE bBgColorIndex;
        BYTE bPixelAspectRatio;
        BYTE blGifMarkerFlag : 1;
        BYTE blGloabalColorTableFlag : 1;
        BYTE blLocalColorTableFlag : 1;
        BYTE blInterlacedFlag : 1;
        BYTE blFirstcodeFlag : 1;
        BYTE bInterlacePass : 3;
        #if GIF_USE_16_BITS_PER_PIXEL == 0
        BYTE aPalette[256][3]; /* Each palette entry has RGB */
```

```
        #else
        WORD awPalette[256]; /* Each palette entry has RGB */
        #endif
        /* For decoding */
        BYTE abSymbol[4096];

        #if GIF_CRUSH_PREV_SYMBOL_PTR_TABLE == 0
        WORD awPrevSymbolPtr[4096];
        #else
        WORD awPrevSymbolPtr[(4096 * 3)/4];
        #endif

        WORD wInitialSymbols;
        WORD wMaxSymbol;
        BYTE bInitialSymbolBits;
        BYTE bMaxSymbolBits;
        LONG lGlobalColorTablePos;
        /* Work memory */
        BYTE bWorkBits;
        BYTE bRemainingDataInBlock;
        BYTE bRemainingBits;
        WORD wCurrentX;
        WORD wCurrentY;
} GIFDECODER;


/***************************/
/****** LOOKUP TABLE ******/
/***************************/
static const WORD GIF_awMask[] = { 0x000, 0x001, 0x003, 0x007, 0x00F, 0x01F, 0x03F, 0x07F,
0x0FF, 0x1FF, 0x3FF, 0x7FF, 0xFFF };

/***************************/
/******* FUNCTIONS  *******/
/***************************/
/*******************************************************************************
Function:       void GIF_vResetData(GIFDECODER *pGifDec)

Precondition:   None

Overview:       This function resets the variables so that new GIF image
                can be decoded

Input:          GIF decoder's data structure

Output:         None
*******************************************************************************/
static void GIF_vResetData(GIFDECODER *pGifDec)
{
    pGifDec->pImageFile = NULL;
    pGifDec->wImageWidth = 0;
    pGifDec->wImageHeight = 0;
    pGifDec->wImageX = 0;
    pGifDec->wImageY = 0;
    pGifDec->wScreenWidth = 0;
    pGifDec->wScreenHeight = 0;
    pGifDec->wGlobalPaletteEntries = 0;
    pGifDec->wLocalPaletteEntries = 0;
    pGifDec->bBgColorIndex = 0;
    pGifDec->bPixelAspectRatio = 0;
    pGifDec->blGifMarkerFlag = 0;
    pGifDec->blGloabalColorTableFlag = 0;
    pGifDec->blLocalColorTableFlag = 0;
    pGifDec->blInterlacedFlag = 0;
    pGifDec->blFirstcodeFlag = 1;
    pGifDec->bInterlacePass = 0;
    pGifDec->wMaxSymbol = 0;
    pGifDec->bMaxSymbolBits = 0;
    pGifDec->wInitialSymbols = 0;
    pGifDec->bInitialSymbolBits = 0;
    pGifDec->bWorkBits = 0;
    pGifDec->bRemainingDataInBlock = 0;
```

```
    pGifDec->bRemainingBits = 0;
    pGifDec->wCurrentX = 0;
    pGifDec->wCurrentY = 0;
    pGifDec->lGlobalColorTablePos = 0;
}

/*****************************************************************************
Function:       void GIF_vPutPrevCode(GIFDECODER *pGifDec, WORD wAddress, WORD wCode)

Precondition:   None

Overview:       This function puts the data(code) to the provided address

Input:          GIF decoder's data structure, Address, Data(Code)

Output:         None
*****************************************************************************/
static void GIF_vPutPrevCode(GIFDECODER *pGifDec, WORD wAddress, WORD wCode)
{
    #if GIF_CRUSH_PREV_SYMBOL_PTR_TABLE == 0
        pGifDec->awPrevSymbolPtr[wAddress] = wCode;
    #else
        WORD wCrushedAddress = (wAddress * 3) / 4;
        if((wAddress & 0x03) == 0)
        {
                pGifDec->awPrevSymbolPtr[wCrushedAddress] &= 0xF000;
                pGifDec->awPrevSymbolPtr[wCrushedAddress] |= (wCode & 0x0FFF);
        }
        else if((wAddress & 0x03) == 1)
        {
                pGifDec->awPrevSymbolPtr[wCrushedAddress] &= 0x0FFF;
                pGifDec->awPrevSymbolPtr[wCrushedAddress + 1] &= 0xFF00;
                pGifDec->awPrevSymbolPtr[wCrushedAddress] |= (wCode << 12);
                pGifDec->awPrevSymbolPtr[wCrushedAddress + 1] |= ((wCode >> 4) & 0x00FF);
        }
        else if((wAddress & 0x03) == 2)
        {
                pGifDec->awPrevSymbolPtr[wCrushedAddress] &= 0x00FF;
                pGifDec->awPrevSymbolPtr[wCrushedAddress + 1] &= 0xFFF0;
                pGifDec->awPrevSymbolPtr[wCrushedAddress] |= (wCode << 8);
                pGifDec->awPrevSymbolPtr[wCrushedAddress + 1] |= ((wCode >> 8) & 0x000F);
        }
        else
        {
                pGifDec->awPrevSymbolPtr[wCrushedAddress] &= 0x000F;
                pGifDec->awPrevSymbolPtr[wCrushedAddress] |= (wCode << 4);
        }
    #endif
}

/*****************************************************************************
Function:       WORD GIF_wGetPrevCode(GIFDECODER *pGifDec, WORD wAddress)

Precondition:   None

Overview:       This function gets the data(code) from the provided address

Input:          GIF decoder's data structure, Address

Output:         Data(Code)
*****************************************************************************/
static WORD GIF_wGetPrevCode(GIFDECODER *pGifDec, WORD wAddress)
{
        WORD wCode;
    #if GIF_CRUSH_PREV_SYMBOL_PTR_TABLE == 0
        wCode = pGifDec->awPrevSymbolPtr[wAddress];
    #else
        WORD wCrushedAddress = (wAddress * 3) / 4;
        if((wAddress & 0x03) == 0)
        {
                wCode = (pGifDec->awPrevSymbolPtr[wCrushedAddress] & 0x0FFF);
        }
```

```
        else if((wAddress & 0x03) == 1)
        {
                wCode = (pGifDec->awPrevSymbolPtr[wCrushedAddress] >> 12);
                wCode |= ((pGifDec->awPrevSymbolPtr[wCrushedAddress + 1] & 0x00FF) << 4);
        }
        else if((wAddress & 0x03) == 2)
        {
                wCode = (pGifDec->awPrevSymbolPtr[wCrushedAddress] >> 8);
                wCode |= ((pGifDec->awPrevSymbolPtr[wCrushedAddress + 1] & 0x000F) << 8);
        }
        else
        {
                wCode = (pGifDec->awPrevSymbolPtr[wCrushedAddress] >> 4);
        }
    #endif
        return wCode;
}

/*****************************************************************************
Function:       void GIF_vInitializeTable(GIFDECODER *pGifDec)

Precondition:   pGifDec->wInitialSymbols must be set to a proper value by
                reading the Header

Overview:       This function initializes the code table to the initial number
                of symbols

Input:          GIF decoder's data structure

Output:         None
*****************************************************************************/
static void GIF_vInitializeTable(GIFDECODER *pGifDec)
{
        WORD wCounter;
        for(wCounter = 0; wCounter < pGifDec->wInitialSymbols; wCounter++)
        {
                 pGifDec->abSymbol[wCounter] = wCounter;
                 GIF_vPutPrevCode(pGifDec, wCounter, pGifDec->wInitialSymbols);
        }
}

/*****************************************************************************
Function:       void GIF_vReadColorTable(GIFDECODER *pGifDec, WORD wNumberOfEntries)

Precondition:   None

Overview:       This function initializes the code table to the initial number
                of symbols

Input:          GIF decoder's data structure

Output:         None
*****************************************************************************/
static void GIF_vReadColorTable(GIFDECODER *pGifDec, WORD wNumberOfEntries)
{
        BYTE r, g, b;
        WORD wCounter;
        for(wCounter = 0; wCounter < wNumberOfEntries; wCounter++)
        {
                 IMG_FREAD(&r, sizeof(BYTE), 1, pGifDec->pImageFile); /* R */
                 IMG_FREAD(&g, sizeof(BYTE), 1, pGifDec->pImageFile); /* G */
                 IMG_FREAD(&b, sizeof(BYTE), 1, pGifDec->pImageFile); /* B */
                 #if GIF_USE_16_BITS_PER_PIXEL == 0
                 pGifDec->aPalette[wCounter][0] = r;
                 pGifDec->aPalette[wCounter][1] = g;
                 pGifDec->aPalette[wCounter][2] = b;
                 #else
                 pGifDec->awPalette[wCounter] = RGB565CONVERT(r, g, b);
                 #endif
        }
}
```

```
/***************************************************************************
Function:       BYTE GIF_bReadHeader(GIFDECODER *pGifDec)

Precondition:   None

Overview:       This function reads the GIF file's header information

Input:          GIF decoder's data structure

Output:         Error code - '0' means no error
***************************************************************************/
static BYTE GIF_bReadHeader(GIFDECODER *pGifDec)
{
        BYTE abByte[6];
        BYTE bFlags;

        IMG_FREAD(abByte, sizeof(BYTE), 6, pGifDec->pImageFile);  /* Marker */
        if(abByte[0] == 'G' && abByte[1] == 'I' && abByte[2] == 'F' &&
           abByte[3] == '8' && (abByte[4] == '7' || abByte[4] == '9') &&
           abByte[5] == 'a')
        {
                pGifDec->blGifMarkerFlag = 1;
        }
        else
        {
            return(100);
        }

        IMG_FREAD(&pGifDec->wScreenWidth, sizeof(WORD), 1, pGifDec->pImageFile);
        IMG_FREAD(&pGifDec->wScreenHeight, sizeof(WORD), 1, pGifDec->pImageFile);
        IMG_FREAD(&bFlags, sizeof(BYTE), 1, pGifDec->pImageFile);  /* Packed fields */
        IMG_FREAD(&pGifDec->bBgColorIndex, sizeof(BYTE), 1, pGifDec->pImageFile);
        IMG_FREAD(&pGifDec->bPixelAspectRatio, sizeof(BYTE), 1, pGifDec->pImageFile);
        if(bFlags & 0x80)
        {
                pGifDec->blGloabalColorTableFlag = 1;
        }
        pGifDec->wGlobalPaletteEntries = 0x01 << ((bFlags & 0x07) + 1);

        if(pGifDec->blGloabalColorTableFlag == 1)
        {
                pGifDec->lGlobalColorTablePos = IMG_FTELL(pGifDec->pImageFile);
                GIF_vReadColorTable(pGifDec, pGifDec->wGlobalPaletteEntries);
        }
        return(0);
}

/***************************************************************************
Function:       BYTE GIF_bReadNextImageDescriptor(GIFDECODER *pGifDec)

Precondition:   File pointer must be pointing to proper location (Like start of
                extension)

Overview:       This function reads the next image descriptor

Input:          GIF decoder's data structure

Output:         Error code - '0' means no error
***************************************************************************/
static BYTE GIF_bReadNextImageDescriptor(GIFDECODER *pGifDec)
{
       BYTE bSection, bSectionDetails, bBlockSize, bTemp;
       BYTE bFlags;
       BYTE bCounter;
       do
       {
            IMG_FREAD(&bSection, sizeof(BYTE), 1, pGifDec->pImageFile);
            if(bSection == 0x21) /* Extension block */
            {
                        IMG_FREAD(&bSectionDetails, sizeof(BYTE), 1, pGifDec->pImageFile);
                        switch(bSectionDetails)
                        {
```

```
/* GRAPHICS EXTENSION */            case 0xF9: IMG_FREAD(&bBlockSize, sizeof(BYTE), 1,
pGifDec->pImageFile);
                                               IMG_FSEEK(pGifDec->pImageFile, bBlockSize, 1);
                                               break;
/* PLAIN TEXT EXTENSION */          case 0x01:
/* APPLICATION EXTENSION */         case 0xFF: IMG_FREAD(&bBlockSize, sizeof(BYTE), 1,
pGifDec->pImageFile);
                                               IMG_FSEEK(pGifDec->pImageFile, bBlockSize, 1);
/* COMMENT EXTENSION */             case 0xFE: IMG_FREAD(&bBlockSize, sizeof(BYTE), 1,
pGifDec->pImageFile);
                                               for(bCounter = 0; bCounter < bBlockSize;
bCounter++)
                                               {
                                                     IMG_FREAD(&bTemp, sizeof(BYTE), 1,
pGifDec->pImageFile);
                                               }
                                               break;

                                    default:   return(100);
                            }
/* BLOCK TERMINATOR */      IMG_FREAD(&bTemp, sizeof(BYTE), 1, pGifDec->pImageFile);
                            if(bTemp != 0)
                            {
                                    return(100);
                            }
                    }
                    else if(bSection != 0x2C)
                    {
                            return(100);
                    }
            }
            while(bSection != 0x2C);

            pGifDec->blFirstcodeFlag = 1;
            pGifDec->bInterlacePass = 0;
            pGifDec->bWorkBits = 0;
            pGifDec->bRemainingDataInBlock = 0;
            pGifDec->bRemainingBits = 0;
            IMG_FREAD(&pGifDec->wImageX, sizeof(WORD), 1, pGifDec->pImageFile);
            IMG_FREAD(&pGifDec->wImageY, sizeof(WORD), 1, pGifDec->pImageFile);
            IMG_FREAD(&pGifDec->wImageWidth, sizeof(WORD), 1, pGifDec->pImageFile);
            IMG_FREAD(&pGifDec->wImageHeight, sizeof(WORD), 1, pGifDec->pImageFile);
            IMG_FREAD(&bFlags, sizeof(BYTE), 1, pGifDec->pImageFile);  /* Packed fields */
            if(bFlags & 0x40)
            {
                    pGifDec->blInterlacedFlag = 1;
            }
            if(bFlags & 0x80)
            {
                    pGifDec->blLocalColorTableFlag = 1;
            }
            pGifDec->wLocalPaletteEntries = 0x01 << ((bFlags & 0x07) + 1);
            if(pGifDec->blLocalColorTableFlag == 1)
            {
                    GIF_vReadColorTable(pGifDec, pGifDec->wLocalPaletteEntries);
            }
            pGifDec->wCurrentX = pGifDec->wImageX;
            pGifDec->wCurrentY = pGifDec->wImageY;
            return 0;
}

/*****************************************************************************
Function:       WORD GIF_wGetNextByte(GIFDECODER *pGifDec)

Precondition:   None

Overview:       This function reads the next data byte. It also handles the
                data block intra-boundaries

Input:          GIF decoder's data structure

Output:         Data byte, 0xFFFF means error
```

```
*****************************************************************************/
static WORD GIF_wGetNextByte(GIFDECODER *pGifDec)
{
        BYTE bByte;
        if(pGifDec->bRemainingDataInBlock == 0)
        {
                IMG_FREAD(&pGifDec->bRemainingDataInBlock, sizeof(BYTE), 1,
pGifDec->pImageFile);
                if(pGifDec->bRemainingDataInBlock == 0)
                {
                        return 0xFFFF; /* End of data */
                }
        }
        IMG_FREAD(&bByte, sizeof(BYTE), 1, pGifDec->pImageFile);
        pGifDec->bRemainingDataInBlock--;
        return (WORD)bByte;
}

/*****************************************************************************
Function:       WORD GIF_wGetNextSymbol(GIFDECODER *pGifDec)

Precondition:   pGifDec->bMaxSymbolBits must be properly updated

Overview:       This function reads the next code symbol brom the data stream

Input:          GIF decoder's data structure

Output:         Next Code, 0xFFFF means error
*****************************************************************************/
static WORD GIF_wGetNextSymbol(GIFDECODER *pGifDec)
{
        WORD wDataBits = 0xFFFF;
        if(pGifDec->bRemainingBits < pGifDec->bMaxSymbolBits)
        {
                WORD wTemp1, wTemp2;
                BYTE bBitsRequired;

                bBitsRequired = pGifDec->bMaxSymbolBits - pGifDec->bRemainingBits;
                wDataBits = pGifDec->bWorkBits & GIF_awMask[pGifDec->bRemainingBits];
                wTemp1 = GIF_wGetNextByte(pGifDec);
                if(wTemp1 == 0xFFFF)
                {
                        return 0xFFFF;
                }
                pGifDec->bWorkBits = (BYTE)wTemp1;
                if(bBitsRequired > 8)
                {
                        wTemp2 = GIF_wGetNextByte(pGifDec);
                        if(wTemp2 == 0xFFFF)
                        {
                                return 0xFFFF;
                        }
                        pGifDec->bWorkBits = (BYTE)wTemp2;
                        wTemp1 |= wTemp2 << 8;
                }
                wDataBits |= wTemp1 << pGifDec->bRemainingBits;
                pGifDec->bRemainingBits = 8 - (bBitsRequired % 8);
                if(bBitsRequired == 8)
                {
                        pGifDec->bRemainingBits = 0;
                }
                pGifDec->bWorkBits >>= 8 - pGifDec->bRemainingBits;
        }
        else
        {
                wDataBits = pGifDec->bWorkBits;
                pGifDec->bRemainingBits -= pGifDec->bMaxSymbolBits;
                pGifDec->bWorkBits >>= pGifDec->bMaxSymbolBits;
        }
        wDataBits &= GIF_awMask[pGifDec->bMaxSymbolBits];
        return wDataBits;
}
```

```
/*****************************************************************************
Function:       void GIF_vPaintData(GIFDECODER *pGifDec, BYTE bData)

Precondition:   pGifDec->blInterlacedFlag must be properly set

Overview:       This function puts the actual pixel on the display. It also
                takes care of the interlaced pixel arrangement.

Input:          GIF decoder's data structure, palette index

Output:         None
*****************************************************************************/
static void GIF_vPaintData(GIFDECODER *pGifDec, BYTE bData)
{
    #if GIF_USE_16_BITS_PER_PIXEL == 0
        IMG_vSetColor(pGifDec->aPalette[bData][0], pGifDec->aPalette[bData][1],
pGifDec->aPalette[bData][2]);
    #else
        IMG_vSetColor565(pGifDec->awPalette[bData]);
    #endif
       IMG_vPutPixel(pGifDec->wCurrentX, pGifDec->wCurrentY);
       pGifDec->wCurrentX++;
       if(pGifDec->wCurrentX >= pGifDec->wImageWidth)
       {
                IMG_vLoopCallback();
                pGifDec->wCurrentX = pGifDec->wImageX;
                if(pGifDec->blInterlacedFlag == 0)
                {
                            pGifDec->wCurrentY++;
                }
                else
                {
                            switch(pGifDec->bInterlacePass)
                            {
                              case 0: pGifDec->wCurrentY += 8;
                                      break;
                              case 1: pGifDec->wCurrentY += 8;
                                      break;
                              case 2: pGifDec->wCurrentY += 4;
                                      break;
                              case 3: pGifDec->wCurrentY += 2;
                                      break;
                            }
                            if(pGifDec->wCurrentY - pGifDec->wImageY >=
pGifDec->wImageHeight)
                            {
                                      switch(pGifDec->bInterlacePass)
                                      {
                                        case 0: pGifDec->wCurrentY = pGifDec->wImageY + 4;
                                                break;
                                        case 1: pGifDec->wCurrentY = pGifDec->wImageY + 2;
                                                break;
                                        case 2: pGifDec->wCurrentY = pGifDec->wImageY + 1;
                                                break;
                                        case 3: pGifDec->wCurrentY = pGifDec->wImageY + 0;
                                                break;
                                      }
                                      pGifDec->bInterlacePass++;
                            }
                }
       }
}

/*****************************************************************************
Function:       void GIF_vDisplayCode(GIFDECODER *pGifDec, WORD wCode)

Precondition:   None

Overview:       This function puts the stream of pixels corresponding to the
                code. This uses recursion.
```

```
Input:          GIF decoder's data structure, code

Output:         None
*****************************************************************************/
static void GIF_vDisplayCode(GIFDECODER *pGifDec, WORD wCode)
{
        WORD wPrevCode = GIF_wGetPrevCode(pGifDec, wCode);
        if(wPrevCode != pGifDec->wInitialSymbols)
        {
                GIF_vDisplayCode(pGifDec, wPrevCode);
        }
        GIF_vPaintData(pGifDec, pGifDec->abSymbol[wCode]);
}

/*****************************************************************************
Function:       BYTE GIF_bTraceFirstData(GIFDECODER *pGifDec, WORD wCode)

Precondition:   None

Overview:       This function gets the first symbol of the code

Input:          GIF decoder's data structure, code

Output:         First symbol of the code
*****************************************************************************/
static BYTE GIF_bTraceFirstData(GIFDECODER *pGifDec, WORD wCode)
{
        while(GIF_wGetPrevCode(pGifDec, wCode) != pGifDec->wInitialSymbols)
        {
                wCode = GIF_wGetPrevCode(pGifDec, wCode);
        }
        return pGifDec->abSymbol[wCode];
}

/*****************************************************************************
Function:       GIF_vExecuteClearCode(GIFDECODER *pGifDec)

Precondition:   None

Overview:       This function sets the code table to the initial condition

Input:          GIF decoder's data structure

Output:         None
*****************************************************************************/
static void GIF_vExecuteClearCode(GIFDECODER *pGifDec)
{
        pGifDec->wMaxSymbol = pGifDec->wInitialSymbols;
        pGifDec->bMaxSymbolBits = pGifDec->bInitialSymbolBits;
        pGifDec->blFirstcodeFlag = 1;
}

/*****************************************************************************
Function:       BYTE GIF_bDecodeNextImage(GIFDECODER *pGifDec)

Precondition:   Header must be read and file pointer should point to the end
                of the previous image

Overview:       This function sets the code table to the initial condition

Input:          GIF decoder's data structure

Output:         Error code - '0' means no error
*****************************************************************************/
static BYTE GIF_bDecodeNextImage(GIFDECODER *pGifDec)
{
         BYTE bBlockTerminator;
         WORD wCode;
         if(GIF_bReadNextImageDescriptor(pGifDec) != 0)
         {
             return(100);
         }
```

```
        IMG_FREAD(&pGifDec->bMaxSymbolBits, sizeof(BYTE), 1, pGifDec->pImageFile);
        if(pGifDec->bMaxSymbolBits > 11)
        {
            return(100);
        }
        pGifDec->wMaxSymbol = (0x01 << pGifDec->bMaxSymbolBits) + 1;
        pGifDec->bMaxSymbolBits++;
        pGifDec->wInitialSymbols = pGifDec->wMaxSymbol;
        pGifDec->bInitialSymbolBits = pGifDec->bMaxSymbolBits;
        GIF_vInitializeTable(pGifDec);

        /* Actual decoding starts here */
        while(!IMG_FEOF(pGifDec->pImageFile))
        {
                IMG_vCheckAndAbort();
                wCode = GIF_wGetNextSymbol(pGifDec);
                if(wCode >= 4096)
                {
                        return(100);
                }
                if(wCode == pGifDec->wInitialSymbols) /* End code */
                {
                        break;
                }
                if(wCode == pGifDec->wInitialSymbols - 1) /* Clear code */
                {
                        GIF_vExecuteClearCode(pGifDec);
                        continue;
                }
                if(wCode <= pGifDec->wMaxSymbol) /* Code exists */
                {
                        GIF_vDisplayCode(pGifDec, wCode);
                        if(pGifDec->blFirstcodeFlag == 0 && pGifDec->wMaxSymbol < 4095)
                        {
                                pGifDec->wMaxSymbol++;
                                pGifDec->abSymbol[pGifDec->wMaxSymbol] =
GIF_bTraceFirstData(pGifDec, wCode);
                        }
                        if(pGifDec->wMaxSymbol < 4095)
                        {
                                GIF_vPutPrevCode(pGifDec, pGifDec->wMaxSymbol + 1,
wCode);
                        }
                        pGifDec->blFirstcodeFlag = 0;
                }
                else if(wCode == pGifDec->wMaxSymbol + 1 && wCode <= 4095)
                {
                        if(pGifDec->blFirstcodeFlag == 1)
                        {
                                return(100);
                        }
                        pGifDec->wMaxSymbol++;
                        pGifDec->abSymbol[pGifDec->wMaxSymbol] =
GIF_bTraceFirstData(pGifDec, GIF_wGetPrevCode(pGifDec, pGifDec->wMaxSymbol));
                        GIF_vDisplayCode(pGifDec, wCode);
                        if(pGifDec->wMaxSymbol < 4095)
                        {
                                GIF_vPutPrevCode(pGifDec, pGifDec->wMaxSymbol + 1,
wCode);
                        }
                }
                else
                {
                        return(100);
                }

                if((pGifDec->bMaxSymbolBits < 12) && (pGifDec->wMaxSymbol >= (0x01 <<
pGifDec->bMaxSymbolBits) - 1))
                {
                        pGifDec->bMaxSymbolBits++;
                }
```

```
        }
        if(pGifDec->blLocalColorTableFlag == 1) /* Restore Global color table */
        {
                if(pGifDec->lGlobalColorTablePos > 0)
                {
                        LONG lFilePos = IMG_FTELL(pGifDec->pImageFile);
                        IMG_FSEEK(pGifDec->pImageFile, pGifDec->lGlobalColorTablePos, 0);
                        GIF_vReadColorTable(pGifDec, pGifDec->wGlobalPaletteEntries);
                        IMG_FSEEK(pGifDec->pImageFile, lFilePos, 0);
                }
        }
        IMG_FREAD(&bBlockTerminator, sizeof(BYTE), 1, pGifDec->pImageFile);
        if(bBlockTerminator == 0)
        {
                BYTE bTemp;
                IMG_FREAD(&bTemp, sizeof(BYTE), 1, pGifDec->pImageFile);
                if(bTemp == 0x3B) /* End of GIF stream */
                {
                        return 0x3B;
                }
                IMG_FSEEK(pGifDec->pImageFile, -1, 1);
        }
        return bBlockTerminator;
}

/*****************************************************************************
Function:       BYTE GIF_bDecode(IMG_FILE *pFile)

Precondition:   None

Overview:       This function decodes and displays a GIF image

Input:          Image file

Output:         Error code - '0' means no error
*****************************************************************************/
BYTE GIF_bDecode(IMG_FILE *pFile)
{
        GIFDECODER GifDec;

        GIF_vResetData(&GifDec);
        GifDec.pImageFile = pFile;
        GIF_bReadHeader(&GifDec);
        if(GifDec.blGifMarkerFlag == 0)
        {
            return(100);
        }
        IMG_wImageWidth = GifDec.wScreenWidth;
        IMG_wImageHeight = GifDec.wScreenHeight;
        IMG_vSetboundaries();
        return GIF_bDecodeNextImage(&GifDec);
}

#endif
#undef __GIFDECODER_C__
```

12.8.4 GifDecoder.h

This is file GifDecoder.h.

Body Source

```
#ifndef __GIFDECODER_H__
 #define __GIFDECODER_H__
/*****************************************************************************
* FileName:        GifDecoder.h
* Dependencies:    Image decoding library; project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
```

```
 * Compiler:        C30 v2.01/C32 v0.00.18
 * Company:         Microchip Technology, Inc.

  * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    14-Mar-2008    First release
*******************************************************************************/

/* User configuration */
#define GIF_CRUSH_PREV_SYMBOL_PTR_TABLE     1 /* If 1, this saves 2KB of RAM but requires
more time to decode */

#ifdef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
    #define GIF_USE_16_BITS_PER_PIXEL           1 /* If this is 1, then 16 bits/pixel is
used and hence requires 256 Bytes less RAM */
#else
    #define GIF_USE_16_BITS_PER_PIXEL           0
#endif

/* User configuration */

/* Function prototype */
/* This function must be called after setting proper values in the global variables of
ImageDecoder.c */
BYTE GIF_bDecode(IMG_FILE *pFile);

#endif
```

12.8.5 ImageDecoder.c

This is file ImageDecoder.c.

Body Source

```
#define __IMAGEDECODER_C__
/*****************************************************************************

 * FileName:        ImageDecoder.c
 * Dependencies:    project requires File System library
 * Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
 * Compiler:        C30 v2.01/C32 v0.00.18
 * Company:         Microchip Technology, Inc.

  * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
*******************************************************************************/

/* This is used as a common header for all image decoders.
   This contains interfaces to sheild the image decoders
   from project specific information like reading from
   USB drive or from sd card, etc... */

#include "Image Decoders/ImageDecoder.h"

/***************************/
/**** GLOBAL VARIABLES ****/
/***************************/

IMG_FILE *IMG_pCurrentFile;

WORD IMG_wStartX;
WORD IMG_wStartY;
WORD IMG_wWidth;
WORD IMG_wHeight;
WORD IMG_wImageWidth;
WORD IMG_wImageHeight;
BYTE IMG_bDownScalingFactor;
BYTE IMG_bAlignCenter;
BYTE IMG_blAbortImageDecoding;

#ifndef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
IMG_PIXEL_OUTPUT IMG_pPixelOutput;
IMG_PIXEL_XY_RGB_888 IMG_PixelXYColor;
#endif

#ifndef IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT
IMG_FILE_SYSTEM_API *IMG_pFileAPIs;
#endif

#ifdef IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
IMG_LOOP_CALLBACK  IMG_pLoopCallbackFn;
#endif

/***************************/
/*******************************************************************************
Function:       void ImageDecoderInit(void)

Precondition:   None

Overview:       This function resets the variables and initializes the display
```

```
Input:          None

Output:         None
*****************************************************************************/
void ImageDecoderInit(void)
{
     IMG_wStartX = 0;
     IMG_wStartY = 0;
     IMG_wWidth  = 0;
     IMG_wHeight = 0;
     IMG_wImageWidth = 0;
     IMG_wImageHeight = 0;

   #ifndef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
     IMG_pPixelOutput = NULL;
     IMG_PixelXYColor.X = 0;
     IMG_PixelXYColor.Y = 0;
     IMG_PixelXYColor.R = 0;
     IMG_PixelXYColor.G = 0;
     IMG_PixelXYColor.B = 0;
   #endif

   #ifndef IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT
     IMG_pFileAPIs = NULL;
   #endif

   #ifdef IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
     IMG_pLoopCallbackFn = NULL;
   #endif
    IMG_blAbortImageDecoding = 0;

#ifdef IMG_USE_NON_BLOCKING_DECODING
    IMG_pCurrentFile = NULL;
#endif
}

/*****************************************************************************
Function:       void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pLoopCallbackFn)

Precondition:   None

Overview:       This function registers the loop callback function

Input:          Loop callback function pointer

Output:         None
*****************************************************************************/
void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pLoopCallbackFn)
{
   #ifdef IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
     IMG_pLoopCallbackFn = pLoopCallbackFn;
   #endif
}

/*****************************************************************************
Function:       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)

Precondition:   None

Overview:       This function uses the approperiate image decoding function to
                decode and display the image

Input:          File pointer, Kind of image, Image position and boundaries, If center
alignment and downscaling to fit into the display is required, File System API pointer and
function to output the decoded pixels

Output:         Error code - '0' means no error
*****************************************************************************/
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)
{
     BYTE bRetVal = 0;

     IMG_wStartX = wStartx;
     IMG_wStartY = wStarty;
     IMG_wWidth  = wWidth;
     IMG_wHeight = wHeight;
     IMG_wImageWidth = 0;
     IMG_wImageHeight = 0;

     IMG_bDownScalingFactor = (wFlags & IMG_DOWN_SCALE)? 1: 0;
     IMG_bAlignCenter = (wFlags & IMG_ALIGN_CENTER)? 1: 0;

   #ifndef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
     IMG_pPixelOutput = pPixelOutput;
   #endif

   #ifndef IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT
     IMG_pFileAPIs = pFileAPIs;
   #endif

     switch(eImgFormat)
     {
      #ifdef IMG_SUPPORT_BMP
       case IMG_BMP: bRetVal = BMP_bDecode(pImageFile);
                     break;
      #endif
      #ifdef IMG_SUPPORT_JPEG
       case IMG_JPEG: bRetVal = JPEG_bDecode(pImageFile, TRUE);
                      IMG_pCurrentFile = pImageFile;
                     break;
      #endif
      #ifdef IMG_SUPPORT_GIF
       case IMG_GIF: bRetVal = GIF_bDecode(pImageFile);
                     break;
      #endif
       default:      bRetVal = 0xFF;
     }

     return(bRetVal);
}


/*****************************************************************************
Function:       BYTE ImageDecodeTask(void)

Precondition:   ImageDecode() must have been called

Overview:       This function completes one small part of the image decode function

Input:          None

Output:         Status code - '1' means decoding is completed
                            - '0' means decoding is not yet completed, call this function
again
*****************************************************************************/
BYTE ImageDecodeTask(void)
{
    #ifdef IMG_USE_NON_BLOCKING_DECODING
    {
        if(IMG_pCurrentFile != NULL) // At present, supports only JPEG
        {
            return JPEG_bDecode(IMG_pCurrentFile, FALSE);
        }
    }
    #endif

    return 1;
}
```

```
/*****************************************************************************
Function:       BYTE IMG_vSetboundaries(void)

Precondition:   None

Overview:       This function sets the boundaries and scaling factor. THIS IS
                NOT FOR THE USER.

Input:          None

Output:         None
*****************************************************************************/
void IMG_vSetboundaries(void)
{
     if(IMG_bDownScalingFactor > 0)
     {
         WORD wXScalingFactor = IMG_wImageWidth / IMG_wWidth;
         WORD wYScalingFactor = IMG_wImageHeight / IMG_wHeight;

         if(wXScalingFactor * IMG_wWidth < IMG_wImageWidth)
         {
             wXScalingFactor++;
         }
         if(wYScalingFactor * IMG_wHeight < IMG_wImageHeight)
         {
             wYScalingFactor++;
         }

         IMG_bDownScalingFactor = (BYTE)(wXScalingFactor > wYScalingFactor)?
wXScalingFactor: wYScalingFactor;

         if(IMG_bDownScalingFactor <= 1)
         {
             IMG_bDownScalingFactor = 0;
         }
     }

     if(IMG_bAlignCenter > 0)
     {
         BYTE bDownScalingFactor = (IMG_bDownScalingFactor <= 1)? 1: IMG_bDownScalingFactor;
         if(IMG_wWidth > (IMG_wImageWidth / bDownScalingFactor))
         {
             IMG_wStartX += (IMG_wWidth - (IMG_wImageWidth / bDownScalingFactor)) / 2;
         }
         if(IMG_wHeight > (IMG_wImageHeight / bDownScalingFactor))
         {
             IMG_wStartY += (IMG_wHeight - (IMG_wImageHeight / bDownScalingFactor)) / 2;
         }
     }
}

#undef __IMAGEDECODER_C__
```

12.8.6 ImageDecoder.h

Functions

	Name	Description
	ImageDecode (see page 452)	This function decodes and displays the image on the screen
	ImageDecoderInit (see page 453)	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
	ImageDecodeTask (see page 454)	This function completes one small part of the image decode function

	ImageLoopCallbackRegister (see page 454)	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 maintainance activities such as fetching data, updating the display, etc...

Macros

Name	Description
ImageAbort (see page 456)	This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.
ImageFullScreenDecode (see page 455)	This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required

Description

This is file ImageDecoder.h.

Body Source

```
#ifndef __IMAGEDECODER_H__
#define __IMAGEDECODER_H__

/*****************************************************************************
* FileName:        ImageDecoder.h
* Dependencies:    project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
* Compiler:        C30 v2.01/C32 v0.00.18
* Company:         Microchip Technology, Inc.

 * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
*****************************************************************************/

/* This is used as a common header for all image decoders.
   This contains interfaces to sheild the image decoders
   from project specific information like reading from
   USB drive or from sd card, etc... */

#include "GenericTypeDefs.h"
#include "stddef.h"

/************* User configuration start *************/
#include "ImageDecoderConfig.h"
/************* User configuration end *************/

/*********************************************************************
* Overview: Typedefs of the used File System APIs
```

```
*****************************************************************/
typedef size_t (*FileRead)(void *ptr, size_t size, size_t n, void *stream);
typedef int    (*FileSeek)(void *stream, long offset, int whence);
typedef long   (*FileTell)(void *fo);
typedef int    (*FileFeof)(void *stream);

/*****************************************************************
* Overview: IMG_FileSystemAPI holds function pointers to the used
*           File System APIs
*****************************************************************/
typedef struct _IMG_FILE_SYSTEM_API
{
     FileRead pFread;
     FileSeek pFseek;
     FileTell pFtell;
     FileFeof pFeof;
} IMG_FILE_SYSTEM_API;

/*****************************************************************
* Overview: IMG_ImageFileFormat specifies all the supported
*           image file formats
*****************************************************************/
typedef enum _IMG_FILE_FORMAT
{
     IMG_NONE = 0,
     IMG_BMP,                /* Bitmap image */
     IMG_JPEG,               /* JPEG image */
     IMG_GIF                 /* GIF image */
} IMG_FILE_FORMAT;

/*****************************************************************
* Overview: IMG_PixelRgb holds the RGB information of a pixel in BYTES
*****************************************************************/
typedef struct _IMG_PIXEL_XY_RGB_888
{
     WORD X;
     WORD Y;
     BYTE R;
     BYTE G;
     BYTE B;
} IMG_PIXEL_XY_RGB_888;

/*****************************************************************
* Overview: IMG_PixelOutput is a callback function which receives the
*           color information of the output pixel
*****************************************************************/
typedef void (*IMG_PIXEL_OUTPUT)(IMG_PIXEL_XY_RGB_888 *pPix);

/*****************************************************************
* Overview: IMG_LoopCallback is a callback function which is called
*           in every loop of the decoding cycle so that user
*           application can do some maintainance activities
*****************************************************************/
typedef void (*IMG_LOOP_CALLBACK)(void);

/* The global variables which define the image position and size */
#ifndef __IMAGEDECODER_C__
  extern BYTE IMG_blAbortImageDecoding;
  extern WORD IMG_wStartX;
  extern WORD IMG_wStartY;
  extern WORD IMG_wWidth;
  extern WORD IMG_wHeight;
  extern WORD IMG_wImageWidth;
  extern WORD IMG_wImageHeight;
  extern BYTE IMG_bDownScalingFactor;
  extern BYTE IMG_bAlignCenter;

 #ifndef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
  extern IMG_PIXEL_OUTPUT IMG_pPixelOutput;
  extern IMG_PIXEL_XY_RGB_888 IMG_PixelXYColor;
 #endif
```

```
 #ifndef IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT
  extern IMG_FILE_SYSTEM_API *IMG_pFileAPIs;
 #endif

 #ifdef IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
  extern IMG_LOOP_CALLBACK  IMG_pLoopCallbackFn;
 #endif
#endif

#ifdef IMG_USE_ONLY_MDD_FILE_SYSTEM_FOR_INPUT

 #include <MDD File System/FSIO.h>

 #define IMG_FILE         FSFILE
 #define IMG_FREAD        FSfread
 #define IMG_FSEEK        FSfseek
 #define IMG_FTELL        FSftell
 #define IMG_FEOF         FSfeof

// added by Paolo 9/10/08 for MultiApp Demo
 #define IMG_FOPEN        FSfopen
 #define IMG_FCLOSE       FSfclose

#else

 #define IMG_FILE         void
 #define IMG_FREAD        IMG_pFileAPIs->pFread
 #define IMG_FSEEK        IMG_pFileAPIs->pFseek
 #define IMG_FTELL        IMG_pFileAPIs->pFtell
 #define IMG_FEOF         IMG_pFileAPIs->pFeof

#endif

#ifndef __IMAGEDECODER_C__

  extern IMG_FILE *IMG_pCurrentFile;

#endif

/* The individual image decoders use these defines instead of directly using those provided
in the Display driver file */
#define IMG_SCREEN_WIDTH         (GetMaxX()+1)
#define IMG_SCREEN_HEIGHT        (GetMaxY()+1)

#ifdef IMG_USE_ONLY_565_GRAPHICS_DRIVER_FOR_OUTPUT
       #include "Graphics/Graphics.h"
       #define IMG_vSetColor(r, g, b)   SetColor(RGB565CONVERT((r), (g), (b)))
       #define IMG_vSetColor565(color)  SetColor(color)
       #define _PutPixel(x, y)          PutPixel(x, y)
#else
       #define IMG_vSetColor(r, g, b)   IMG_PixelXYColor.R = r; IMG_PixelXYColor.G = g;
IMG_PixelXYColor.B = b;
       #define IMG_vSetColor565(color)  IMG_PixelXYColor.R = ((color)>>11)&0x1F;
IMG_PixelXYColor.G = ((color)>>5)&0x3F; IMG_PixelXYColor.B = (color)&0x1F;
       #define _PutPixel(x, y)          if(IMG_pPixelOutput != NULL) { IMG_PixelXYColor.X =
x; IMG_PixelXYColor.Y = y; IMG_pPixelOutput(&IMG_PixelXYColor); }
       #ifndef RGB565CONVERT
            #define RGB565CONVERT(r, g, b) (((r >> 3) << 11) | ((g >> 2) << 5) | (b >> 3))
       #endif
#endif

#define IMG_vPutPixel(x, y)      if(IMG_bDownScalingFactor <= 1)\
                                 {                                \
                                     if((x) < IMG_wImageWidth && (y) < IMG_wImageHeight) {
_PutPixel(IMG_wStartX + (x), IMG_wStartY + (y)); }\
                                 }                                \
                                 else if((x)%IMG_bDownScalingFactor == 0 &&
(y)%IMG_bDownScalingFactor == 0)      \
                                 {                                \
                                     if((x) < IMG_wImageWidth && (y) < IMG_wImageHeight) {
_PutPixel(IMG_wStartX + ((x)/IMG_bDownScalingFactor), IMG_wStartY +
((y)/IMG_bDownScalingFactor)); }\
```

```
                                    }


#ifdef IMG_SUPPORT_IMAGE_DECODER_LOOP_CALLBACK
       #define IMG_vLoopCallback() if(IMG_pLoopCallbackFn != NULL) { IMG_pLoopCallbackFn();
}
#else
       #define IMG_vLoopCallback()
#endif

#define IMG_DECODE_ABORTED 0xFF
#define IMG_vCheckAndAbort()                \
        if(IMG_blAbortImageDecoding == 1)   \
        {                                   \
              IMG_blAbortImageDecoding = 0; \
              return IMG_DECODE_ABORTED;    \
        }


#ifdef IMG_SUPPORT_BMP
 #include "BmpDecoder.h"
#endif

#ifdef IMG_SUPPORT_JPEG
 #include "JpegDecoder.h"
#endif

#ifdef IMG_SUPPORT_GIF
 #include "GifDecoder.h"
#endif

/* Function prototypes */
/* User uses only these three functions */

/*********************************************************************
* Function: void ImageDecoderInit(void)
*
* 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
*
* Input: None
*
* Output: None
*
* Example:
*   <PRE>
*   void main(void)
*   {
*       ImageInit();
*       ...
*   }
*   </PRE>
*
* Side Effects: The graphics driver will be reset
*
*********************************************************************/
void ImageDecoderInit(void);

/*********************************************************************
* Function: void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pLoopCallbackFn)
*
* 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
*           maintainance activities such as fetching data, updating
*           the display, etc...
*
* Input: pLoopCallbackFn -> Loop callback function pointer
*
* Output: None
```

```
*
* Example:
*   <PRE>
*   void Mainantance(void)
*   {
*       ...
*   }
*
*   void main(void)
*   {
*       ImageInit();
*       ImageLoopCallbackRegister(Mainantance);
*       ...
*   }
*   </PRE>
*
* Side Effects: The graphics driver will be reset
*
*********************************************************************/
void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pFn);

/*********************************************************************
* Function: 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)
*
* Overview: This function decodes and displays the image on the screen
*
* Input: pImageFile -> The image file pointer
*        eImgFormat -> Type of image to be decoded
*        wStartx  -> Starting x position for the image to be displayed
*        wStarty  -> Starting y position for the image to be displayed
*        wWidth   -> The width into which the image to be displayed/compressed
*                    (This is not the width of the image)
*        wHeight  -> The height into which the image to be displayed/compressed
*                    (This is not the height of the image)
*        wFlags   -> If bit 0 is set, the image would be center aligned into the area
*                          specified by wStartx, wStarty, wWidth and wHeight
*                 -> If bit 1 is set, the image would be downscaled if required to fit
*                          into the area specified by wStartx, wStarty, wWidth
*                          and wHeight
*        pFileAPIs     -> The pointer to a structure which has function pointers
*                          to the File System APIs
*        pPixelOutput  -> The function to output (x, y) coordinates and the color
*
* Output: Error code -> 0 means no error
*
* Example:
*   <PRE>
*   void main(void)
*   {
*       IMG_FILE pImageFile;
*       IMG_vInitialize();
*       pImageFile = IMG_FOPEN("Image.jpg", "r");
*       if(pImageFile == NULL)
*       {
*                <- Error handling ->
*       }
*       IMG_bDecode(pImageFile, IMG_JPEG, 0, 0, 320, 240, 0, NULL, NULL);
*       IMG_FCLOSE(pImageFile);
*       while(1);
*   }
*   </PRE>
*
* Side Effects: None
*
*********************************************************************/
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);
```

```
/* Flags */
#define IMG_ALIGN_CENTER 0x0001
#define IMG_DOWN_SCALE   0x0002


/*********************************************************************
* Function: BYTE ImageFullScreenDecode(IMG_FILE *pImageFile, IMG_FILE_FORMAT eImgFormat,
IMG_FILE_SYSTEM_API pFileAPIs, IMG_PIXEL_OUTPUT pPixelOutput)
*
* Overview: This function decodes and displays the image on the screen in
*           fullscreen mode with center aligned and downscaled if required
*
* Input: pImageFile -> The image file pointer
*        eImgFormat -> Type of image to be decoded
*        pFileAPIs     -> The pointer to a structure which has function pointers
*                         to the File System APIs
*        pPixelOutput -> The function to output (x, y) coordinates and the color
*
* Output: Error code -> 0 means no error
*
* Example:
*   <PRE>
*   void main(void)
*   {
*       IMG_FILE pImageFile;
*       IMG_vInitialize();
*       pImageFile = IMG_FOPEN("Image.jpg", "r");
*       if(pImageFile == NULL)
*       {
*                 <- Error handling ->
*       }
*       IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
*       IMG_FCLOSE(pImageFile);
*       while(1);
*   }
*   </PRE>
*
* Side Effects: None
*
*********************************************************************/
#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)

/*********************************************************************
* Function: BYTE ImageDecodeTask(void)
*
* Overview: This function completes one small part of the image decode function
*
* Input: None
*
* Output: Status code - '1' means decoding is completed
*                     - '0' means decoding is not yet completed, call this function again
*
* Example:
*   <PRE>
*
*       IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
*       while(!ImageDecodeTask());
*
*   </PRE>
*
* Side Effects: None
*
*********************************************************************/
BYTE ImageDecodeTask(void);

/*********************************************************************
* Function: void ImageAbort(void)
*
* Overview: This function sets the Image Decoder's Abort flag so that
*           decoding aborts in the next decoding loop.
```

```
*
* Input: None
*
* Output: None
*
* Example:
*   <PRE>
*
*   void callback(void);
*   void main(void)
*   {
*       IMG_FILE pImageFile;
*       IMG_vInitialize();
*       ImageLoopCallbackRegister(callback);
*       pImageFile = IMG_FOPEN("Image.jpg", "r");
*       if(pImageFile == NULL)
*       {
*                <- Error handling ->
*       }
*       IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
*       IMG_FCLOSE(pImageFile);
*       while(1);
*   }
*
*   void callback(void)
*   {
*       if(<- check for abort condition ->)
*       {
*           ImageAbort();
*       }
*   }
*
*   </PRE>
*
* Side Effects: None
*
*********************************************************************/
#define ImageAbort() IMG_blAbortImageDecoding = 1;

/********* This is not for the user *********/
/* This is used by the individual decoders */
void IMG_vSetboundaries(void);
/********* This is not for the user *********/
#endif
```

12.8.7 JpegDecoder.c

Structures

	Name	Description
	_JPEGDECODER (see page 513)	DATA STRUCTURES

Description

This is file JpegDecoder.c.

Body Source

```
#define __JPEGDECODER_C__
/*****************************************************************************

* FileName:        JpegDecoder.c
* Dependencies:    Image decoding library; project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
* Compiler:        C30 v2.01/C32 v0.00.18
* Company:         Microchip Technology, Inc.
```

```
 * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
                     03-Sep-2010    Changed paint block to more efficient code.
*******************************************************************************/

#include "Image Decoders/ImageDecoder.h"

#ifdef IMG_SUPPORT_JPEG

#define JPEG_SAMPLE_1x1 0
#define JPEG_SAMPLE_1x2 1
#define JPEG_SAMPLE_2x1 2
#define JPEG_SAMPLE_2x2 3

/******************************/
/**** FUNCTION PROTOTYPES ****/
/******************************/
void jpeg_idct_islow (SHORT *inbuf, WORD *quantptr);

/**************************/
/**** DATA STRUCTURES ****/
/**************************/
typedef struct _JPEGDECODER
{
        IMG_FILE *pImageFile;                       /* Image file pointer */

        /************ From APP0 ************/
        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 --------*/

        /************ From SOF0 ************/
        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 abChannelType[MAX_CHANNELS];
        BYTE abChannelHSampFactor[MAX_CHANNELS];
        BYTE abChannelVSampFactor[MAX_CHANNELS];
        BYTE abChannelQuantTableMap[MAX_CHANNELS];

        /************ From DQT ************/
        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 */
```

```
        /************ From DRI ************/
        WORD wRestartInterval;                  /* The restart interval in blocks */

        /************ From DHT ************/
        BYTE bHuffTables;
        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 :-) */
        /************ For Huffman-Decoding ************/
        WORD awHuffAcSymStart[MAX_HUFF_TABLES][16]; /* Starting symbol for each length */
        WORD awHuffDcSymStart[MAX_HUFF_TABLES][16]; /* Starting symbol for each length */

        /************ From SOS ************/
        BYTE abChannelHuffAcTableMap[MAX_CHANNELS];
        BYTE abChannelHuffDcTableMap[MAX_CHANNELS];

        BYTE bError;

        /************ Work memory ************/
        WORD wWorkBits;
        BYTE bBitsAvailable;
        BYTE bBlocksInOnePass;
        SHORT asOneBlock[MAX_BLOCKS][64];     /* Temporary storage for a 8x8 block */
        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;
} JPEGDECODER;

/***************************/
/**** CONSTANT TABLES  ****/
/***************************/
static const BYTE abZigzag[64] =
{
  0,  1,  8,  16, 9,  2,  3,  10,
  17, 24, 32, 25, 18, 11, 4,  5,
  12, 19, 26, 33, 40, 48, 41, 34,
  27, 20, 13, 6,  7,  14, 21, 28,
  35, 42, 49, 56, 57, 50, 43, 36,
  29, 22, 15, 23, 30, 37, 44, 51,
  58, 59, 52, 45, 38, 31, 39, 46,
  53, 60, 61, 54, 47, 55, 62, 63
};

/***************************/
/******* FUNCTIONS  *******/
/***************************/

/*******************************************************************************
Function:       void JPEG_vResetDecoder(JPEGDECODER *pJpegDecoder)

Precondition:   None

Overview:       This function resets the variables so that new jpeg image can be
                decoded

Input:          JPEGDECODER
```

```
Output:         None
*****************************************************************************/
static void JPEG_vResetDecoder(JPEGDECODER *pJpegDecoder)
{
        WORD wIndex;
        BYTE *pbptr = (BYTE*)pJpegDecoder;
        for(wIndex = 0; wIndex < sizeof(JPEGDECODER); wIndex++)
        {
                pbptr[wIndex] = 0;
        }
}

/*****************************************************************************
Function:       WORD JPEG_wReadWord(IMG_FILE *pfile)

Precondition:   None

Overview:       This function reads and returns a single word obtained as
                Higher byte first followed by lower byte

Input:          Image file

Output:         One word
*****************************************************************************/
static WORD JPEG_wReadWord(IMG_FILE *pfile)
{
     BYTE btemp;
     WORD wData;

     IMG_FREAD(&btemp, sizeof(btemp), 1, pfile);
     wData = btemp;
     IMG_FREAD(&btemp, sizeof(btemp), 1, pfile);
     wData = (wData << 8) | btemp;

     return wData;
}

/*****************************************************************************
Function:       BYTE JPEG_bReadByte(IMG_FILE *pfile)

Precondition:   None

Overview:       This function reads and returns a single byte from the file

Input:          Image file

Output:         One byte
*****************************************************************************/
static BYTE JPEG_bReadByte(IMG_FILE *pfile)
{
     BYTE bData;

     IMG_FREAD(&bData, sizeof(bData), 1, pfile);

     return bData;
}

/*****************************************************************************
Function:       BYTE JPEG_bReadHeader(JPEGDECODER *pJpegDecoder)

Precondition:   None

Overview:       This function reads the JPEG file header and
                fills the data structure

Input:          JPEGDECODER

Output:         Error code - '0' means no error
*****************************************************************************/
static BYTE JPEG_bReadHeader(JPEGDECODER *pJpegDecoder)
{
     BYTE blSOSOver = FALSE;
```

```
        while(!IMG_FEOF(pJpegDecoder->pImageFile))
        {
                BYTE btemp, bcount, bsection;
                WORD wSegLen, wOffset;

                if(blSOSOver == TRUE)
                {
                        if(pJpegDecoder->bChannels == 1)
                        {
                                pJpegDecoder->bBlocksInOnePass = 1;
                                pJpegDecoder->bSubSampleType = JPEG_SAMPLE_1x1;
                        }
                        else if(pJpegDecoder->bChannels == 3)
                        {
                                if((pJpegDecoder->abChannelHSampFactor[0] == 1 &&
pJpegDecoder->abChannelVSampFactor[0] == 1) &&
                                   (pJpegDecoder->abChannelHSampFactor[1] == 1 &&
pJpegDecoder->abChannelVSampFactor[1] == 1) &&
                                   (pJpegDecoder->abChannelHSampFactor[2] == 1 &&
pJpegDecoder->abChannelVSampFactor[2] == 1))
                                {
                                        pJpegDecoder->bBlocksInOnePass = 3;
                                        pJpegDecoder->abChannelMap[0] = 0;
                                        pJpegDecoder->abChannelMap[1] = 1;
                                        pJpegDecoder->abChannelMap[2] = 2;
                                        pJpegDecoder->bSubSampleType = JPEG_SAMPLE_1x1;
                                }
                                else if((pJpegDecoder->abChannelHSampFactor[0] == 1 &&
pJpegDecoder->abChannelVSampFactor[0] == 2) &&
                                        (pJpegDecoder->abChannelHSampFactor[1] == 1 &&
pJpegDecoder->abChannelVSampFactor[1] == 1) &&
                                        (pJpegDecoder->abChannelHSampFactor[2] == 1 &&
pJpegDecoder->abChannelVSampFactor[2] == 1))
                                {
                                        pJpegDecoder->bBlocksInOnePass = 4;
                                        pJpegDecoder->abChannelMap[0] = 0;
                                        pJpegDecoder->abChannelMap[1] = 0;
                                        pJpegDecoder->abChannelMap[2] = 1;
                                        pJpegDecoder->abChannelMap[3] = 2;
                                        pJpegDecoder->bSubSampleType = JPEG_SAMPLE_1x2;
                                }
                                else if((pJpegDecoder->abChannelHSampFactor[0] == 2 &&
pJpegDecoder->abChannelVSampFactor[0] == 1) &&
                                        (pJpegDecoder->abChannelHSampFactor[1] == 1 &&
pJpegDecoder->abChannelVSampFactor[1] == 1) &&
                                        (pJpegDecoder->abChannelHSampFactor[2] == 1 &&
pJpegDecoder->abChannelVSampFactor[2] == 1))
                                {
                                        pJpegDecoder->bBlocksInOnePass = 4;
                                        pJpegDecoder->abChannelMap[0] = 0;
                                        pJpegDecoder->abChannelMap[1] = 0;
                                        pJpegDecoder->abChannelMap[2] = 1;
                                        pJpegDecoder->abChannelMap[3] = 2;
                                        pJpegDecoder->bSubSampleType = JPEG_SAMPLE_2x1;
                                }
                                else if((pJpegDecoder->abChannelHSampFactor[0] == 2 &&
pJpegDecoder->abChannelVSampFactor[0] == 2) &&
                                        (pJpegDecoder->abChannelHSampFactor[1] == 1 &&
pJpegDecoder->abChannelVSampFactor[1] == 1) &&
                                        (pJpegDecoder->abChannelHSampFactor[2] == 1 &&
pJpegDecoder->abChannelVSampFactor[2] == 1))
                                {
                                        pJpegDecoder->bBlocksInOnePass = 6;
                                        pJpegDecoder->abChannelMap[0] = 0;
                                        pJpegDecoder->abChannelMap[1] = 0;
                                        pJpegDecoder->abChannelMap[2] = 0;
                                        pJpegDecoder->abChannelMap[3] = 0;
                                        pJpegDecoder->abChannelMap[4] = 1;
                                        pJpegDecoder->abChannelMap[5] = 2;
                                        pJpegDecoder->bSubSampleType = JPEG_SAMPLE_2x2;
                                }
                                else
```

```
                                {
                                        JPEG_SendError(100);
                                }
                        }
                        return 0;
                }

                IMG_FREAD(&btemp, sizeof(btemp), 1, pJpegDecoder->pImageFile);

                if(btemp != 0xFF)
                {
                        continue;
                }

                IMG_FREAD(&bsection, sizeof(bsection), 1, pJpegDecoder->pImageFile);
                switch(bsection)
                {
                  case SOI:
                  case TEM:
                  case EOI:
                  case RST0:
                  case RST1:
                  case RST2:
                  case RST3:
                  case RST4:
                  case RST5:
                  case RST6:
                  case RST7: break;

                  case SOF0: /* Start of frame */
                             wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                             if(wSegLen <= 8)
                             {
                                        JPEG_SendError(100);
                                        break;
                             }
                             pJpegDecoder->bDataBits =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                             pJpegDecoder->wHeight   =
JPEG_wReadWord(pJpegDecoder->pImageFile);
                             pJpegDecoder->wWidth    =
JPEG_wReadWord(pJpegDecoder->pImageFile);
                             pJpegDecoder->bChannels =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                             if(wSegLen < 8 + (pJpegDecoder->bChannels * 3))
                             {
                                        JPEG_SendError(100);
                                        break;
                             }
                             for(bcount = 0; bcount < pJpegDecoder->bChannels; bcount++)
                             {
                                        pJpegDecoder->abChannelType[bcount] =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                                        btemp = JPEG_bReadByte(pJpegDecoder->pImageFile);
                                        pJpegDecoder->abChannelHSampFactor[bcount] = btemp >>
4;
                                        pJpegDecoder->abChannelVSampFactor[bcount] = btemp &
0x0F;
                                        pJpegDecoder->abChannelQuantTableMap[bcount] =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                             }
                             break;

                  case APP0: /* Start of Application */
                             wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                             if(wSegLen < 16)
                             {
                                        JPEG_SendError(100);
                                        break;
                             }
                             else
                             {
```

```
                                        unsigned char buf[5];
                                        IMG_FREAD(buf, sizeof(buf[0]), 5,
pJpegDecoder->pImageFile);
                                        if(buf[0] == 'J' && buf[1] == 'F' && buf[2] == 'I' &&
buf[3] == 'F' && buf[4] == '\0')
                                        {
                                                  pJpegDecoder->blJFIF = TRUE;
                                        }
                              }
                              pJpegDecoder->bMajorRev =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                              pJpegDecoder->bMinorRev =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                              if(pJpegDecoder->bMajorRev != 1)
                              {
                                        JPEG_SendError(100);
                                        break;
                              }
                              /* Ignore other bytes in this segment */
                              break;

                    case DRI: /* Start of Quantization table */
                              wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                              if(wSegLen == 4)
                              {
                                        pJpegDecoder->wRestartInterval =
JPEG_wReadWord(pJpegDecoder->pImageFile);
                              }
                              break;

                    case DQT: /* Start of Quantization table */
                              wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                              if(wSegLen < 67)
                              {
                                        JPEG_SendError(100);
                                        break;
                              }

                              do
                              {
                                        BYTE bQTableIndex, bCounter;

                                        btemp = JPEG_bReadByte(pJpegDecoder->pImageFile);
                                        bQTableIndex = btemp & 0x0F;
                                        pJpegDecoder->blQuantUses16bits |= (btemp >> 4);

                                        for(bCounter = 0; bCounter < 64; bCounter++)
                                        {
                                                  BYTE bData1, bData2 = 0;
                                                  if(pJpegDecoder->blQuantUses16bits == 0)
                                                  {
                                                            IMG_FREAD(&bData1, sizeof(BYTE),
1, pJpegDecoder->pImageFile);
                                                            pJpegDecoder->awQuantTable[bQTableI
ndex][abZigzag[bCounter]]
= bData1;
                                                  }
                                                  else
                                                  {
                                                            IMG_FREAD(&bData1, sizeof(BYTE),
1, pJpegDecoder->pImageFile);
                                                            IMG_FREAD(&bData2, sizeof(BYTE),
1, pJpegDecoder->pImageFile);
                                                            pJpegDecoder->awQuantTable[bQTableI
ndex][abZigzag[bCounter]]
= (((WORD)bData1) << 8) + (WORD)bData2;
                                                  }
                                        }
                                        wSegLen -= (pJpegDecoder->blQuantUses16bits == 0)? 65:
129; /* Table length + information byte */
                              } while(wSegLen >= 67);
                              break;
```

```
                case DHT: /* Start of Huffmann table */
                            wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                            if(wSegLen < 19)
                            {
                                        JPEG_SendError(100);
                                        break;
                            }

                            do
                            {
                                        BYTE bHTableIndex, bCounter;
                                        WORD wNumOfSymbols;
                                        BYTE blIsAc;
                                        BYTE *pbLenTable, *pbSymTable;

                                        btemp = JPEG_bReadByte(pJpegDecoder->pImageFile);
                                        bHTableIndex = btemp & 0x0F;
                                        blIsAc = (btemp >> 4) & 0x01;

                                        if(blIsAc == 0)
                                        {
                                                    pbLenTable =
(BYTE*)&pJpegDecoder->abHuffDcSymLen[bHTableIndex][0];
                                                    pbSymTable =
(BYTE*)&pJpegDecoder->abHuffDcSymbol[bHTableIndex][0];
                                        }
                                        else
                                        {
                                                    pbLenTable =
(BYTE*)&pJpegDecoder->abHuffAcSymLen[bHTableIndex][0];
                                                    pbSymTable =
(BYTE*)&pJpegDecoder->abHuffAcSymbol[bHTableIndex][0];
                                        }

                                        IMG_FREAD(pbLenTable, sizeof(BYTE), 16,
pJpegDecoder->pImageFile);

                                        for(bCounter = 0, wNumOfSymbols = 0; bCounter < 16;
bCounter++)
                                        {
                                                    wNumOfSymbols += pbLenTable[bCounter];
                                        }

                                        wSegLen -= 17; /* This table length + information byte
*/

                                        if(wSegLen < wNumOfSymbols || (blIsAc == 1 &&
wNumOfSymbols > 256) || (blIsAc == 0 && wNumOfSymbols > 16))
                                        {
                                                    JPEG_SendError(100);
                                                    break;
                                        }

                                        IMG_FREAD(pbSymTable, sizeof(BYTE), wNumOfSymbols,
pJpegDecoder->pImageFile);
                                        wSegLen -= wNumOfSymbols; /* This table length +
information byte */
                                        pJpegDecoder->bHuffTables++;
                            } while(wSegLen >= 19);
                            break;

                case SOS: /* Start of Scan parameters */
                            wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                            if(wSegLen < 3)
                            {
                                        JPEG_SendError(100);
                                        break;
                            }

                            btemp = JPEG_bReadByte(pJpegDecoder->pImageFile);
                            wOffset = wSegLen - (3 + (btemp * 2));
```

```
                        if(pJpegDecoder->bChannels != btemp || wSegLen < 3 + (btemp * 2))
                        {
                                    JPEG_SendError(100);
                                    break;
                        }
                        else
                        {
                                    BYTE bCounter, bChannelId = 0xFF;

                                    for(bCounter = 0; bCounter < pJpegDecoder->bChannels;
bCounter++)
                                    {
                                                BYTE bindex;

                                                btemp =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                                                for(bindex = 0; bindex < MAX_CHANNELS;
bindex++)
                                                {
                                                            if(pJpegDecoder->abChannelType[bin
dex]
== btemp)

                                                            {
                                                                        bChannelId = bindex;
                                                                        break;
                                                            }
                                                }

                                                if(bChannelId < MAX_CHANNELS)
                                                {
                                                            btemp =
JPEG_bReadByte(pJpegDecoder->pImageFile);
                                                            pJpegDecoder->abChannelHuffAcTable
Map[bChannelId]
= btemp & 0x0F;

                                                            pJpegDecoder->abChannelHuffDcTable
Map[bChannelId]
= btemp >> 4;

                                                }
                                    }
                                    IMG_FSEEK(pJpegDecoder->pImageFile, wOffset, 1);
                        }
                        blSOSOver = TRUE;
                        break;

                default: /* Any other segment with length */
                        wSegLen = JPEG_wReadWord(pJpegDecoder->pImageFile);
                        if(wSegLen < 2)
                        {
                                    JPEG_SendError(100);
                                    break;
                        }
                        IMG_FSEEK(pJpegDecoder->pImageFile, wSegLen - 2, 1);
            }
      }
      return 0;
}

/*****************************************************************************
Function:       BYTE JPEG_bGenerateHuffmanTables(JPEGDECODER *pJpegDecoder)

Precondition:   None

Overview:       This function generated the table required for Huffman decoding
                from the data read from the header

Input:          JPEGDECODER

Output:         Error code - '0' means no error
*****************************************************************************/
static BYTE JPEG_bGenerateHuffmanTables(JPEGDECODER *pJpegDecoder)
```

```
{
	BYTE bLength, bTable;

	for(bTable = 0; bTable < pJpegDecoder->bHuffTables / 2; bTable++)
	{
		pJpegDecoder->awHuffAcSymStart[bTable][0] = 0;
		pJpegDecoder->awHuffDcSymStart[bTable][0] = 0;

		for(bLength = 1; bLength < 16; bLength++)
		{
			pJpegDecoder->awHuffAcSymStart[bTable][bLength] =
(pJpegDecoder->awHuffAcSymStart[bTable][bLength - 1] +
pJpegDecoder->abHuffAcSymLen[bTable][bLength - 1]) << 1;
			pJpegDecoder->awHuffDcSymStart[bTable][bLength] =
(pJpegDecoder->awHuffDcSymStart[bTable][bLength - 1] +
pJpegDecoder->abHuffDcSymLen[bTable][bLength - 1]) << 1;
		}
	}
	return 0;
}

/*****************************************************************************
Function:	BYTE JPEG_bGet1Bit(JPEGDECODER *pJpegDecoder)

Precondition:	None

Overview:	This function returns 1 bit as the lsb of the returned byte.
		It automatically fills the buffer if it becomes empty.
		It also converts 0xFF00 into 0.

Input:		JPEGDECODER

Output:		One bit
*****************************************************************************/
static BYTE JPEG_bGet1Bit(JPEGDECODER *pJpegDecoder)
{
	BYTE bBit = 0;

	if(pJpegDecoder->wBufferIndex >= pJpegDecoder->wBufferLen)
	{
		pJpegDecoder->wBufferLen = IMG_FREAD(&pJpegDecoder->abDataBuffer[0],
sizeof(BYTE), MAX_DATA_BUF_LEN, pJpegDecoder->pImageFile);
		while(pJpegDecoder->abDataBuffer[pJpegDecoder->wBufferLen - 1] == 0xFF)
		{
			pJpegDecoder->wBufferLen--;
			IMG_FSEEK(pJpegDecoder->pImageFile, -1, 1);
		}
		pJpegDecoder->wBufferIndex = 0;
	}

	while(pJpegDecoder->bBitsAvailable == 0) /* Be very careful to touch this part of
code! You must know exactly what you are doing */
	{
		pJpegDecoder->bBitsAvailable = 16;
		pJpegDecoder->wWorkBits =
pJpegDecoder->abDataBuffer[pJpegDecoder->wBufferIndex++];
		pJpegDecoder->wWorkBits = (pJpegDecoder->wWorkBits << 8) +
pJpegDecoder->abDataBuffer[pJpegDecoder->wBufferIndex++];
		if(pJpegDecoder->wBufferIndex > pJpegDecoder->wBufferLen) /* wBufferIndex need
not be even because of the below condition */
		{
			pJpegDecoder->bBitsAvailable = 8;
		}
		else if((pJpegDecoder->wWorkBits & 0x00FF) == 0x00FF)
		{
			pJpegDecoder->bBitsAvailable = 8;
			pJpegDecoder->wBufferIndex--;
		}
		else if(pJpegDecoder->wWorkBits >= 0xFF00)
		{
			if(pJpegDecoder->wWorkBits == 0xFF00)
			{
```

```
                        pJpegDecoder->bBitsAvailable = 8;
                    }
            }
    }

    bBit = (pJpegDecoder->wWorkBits & 0x8000)? 0x01: 0;

    pJpegDecoder->wWorkBits <<= 1;
    pJpegDecoder->bBitsAvailable--;

    return bBit;
}

/*****************************************************************************
Function:       WORD JPEG_wGetBits(JPEGDECODER *pJpegDecoder, BYTE bLen)

Precondition:   None

Overview:       This function returns bLen number of bits as the lsb of the
                returned word and it automatically fills the buffer
                if it becomes empty.

Input:          JPEGDECODER, Number of bits

Output:         Requested number of bits
*****************************************************************************/
static WORD JPEG_wGetBits(JPEGDECODER *pJpegDecoder, BYTE bLen)
{
    WORD wVal = 0;

    wVal = pJpegDecoder->bFirstBit = JPEG_bGet1Bit(pJpegDecoder);
    bLen--;

    while(bLen)
    {
            wVal = (wVal << 1) + JPEG_bGet1Bit(pJpegDecoder);
            bLen--;
    }

    return wVal;
}

/*****************************************************************************
Function:       WORD JPEG_wGetRestartWord(JPEGDECODER *pJpegDecoder)

Precondition:   File pointer must point to the restart word

Overview:       Returns the restart word

Input:          JPEGDECODER

Output:         Restart word
*****************************************************************************/
static WORD JPEG_wGetRestartWord(JPEGDECODER *pJpegDecoder)
{
    WORD wRestartWord;
    while((pJpegDecoder->bBitsAvailable & 0x07) != 0) /* This is to clearoff wnwanted bits
to go to fresh byte */
    {
            JPEG_bGet1Bit(pJpegDecoder);
    }
    wRestartWord = JPEG_wGetBits(pJpegDecoder, 16);
    return(wRestartWord);
}

/*****************************************************************************
Function:       SHORT JPEG_sGetBitsValue(JPEGDECODER *pJpegDecoder, BYTE bLen)

Precondition:   None

Overview:       Returns the signed value of the bLen bits of data
```

```
Input:          JPEGDECODER, Number of bits

Output:         Signed number
*****************************************************************************/
static SHORT JPEG_sGetBitsValue(JPEGDECODER *pJpegDecoder, BYTE bLen)
{
     WORD wVal = 0;

     if(bLen != 0)
     {
            wVal = JPEG_wGetBits(pJpegDecoder, bLen);
            if(pJpegDecoder->bFirstBit == 0)
            {
                   wVal = ~(wVal | (0xFFFF << bLen));
                   return ((SHORT)wVal * -1);
            }
     }
     return (SHORT)wVal;
}

/*****************************************************************************
Function:       BYTE JPEG_bGetNextHuffByte(JPEGDECODER *pJpegDecoder)

Precondition:   File pointer must point to the Huffman data

Overview:       Returns the Huffman decoded data

Input:          JPEGDECODER

Output:         Huffman decoded data
*****************************************************************************/
static BYTE JPEG_bGetNextHuffByte(JPEGDECODER *pJpegDecoder)
{
     BYTE bBits, bSymbol = 0;
     WORD wBitPattern = 0, wSymbolOffset = 0;

     for(bBits = 0; bBits < 16; bBits++)
     {
            BYTE bSymbols;
            WORD wDiff;

            wBitPattern = (wBitPattern << 1) + JPEG_bGet1Bit(pJpegDecoder);
            bSymbols = pJpegDecoder->pbCurrentHuffSymLenTable[bBits];
            if(bSymbols == 0)
            {
                   continue;
            }

            wDiff = wBitPattern - pJpegDecoder->pwCurrentHuffSymStartTable[bBits];
            if(wDiff < bSymbols)
            {
                   bSymbol = pJpegDecoder->pbCurrentHuffSymbolTable[wSymbolOffset + wDiff];
                   break;
            }
            wSymbolOffset += bSymbols;
     }
     return bSymbol;
}

#define range_limit(x) (x<0)?0:(x>0xFF)?0xFF:x
/*****************************************************************************
Function:       BYTE JPEG_bDecodeOneBlock(JPEGDECODER *pJpegDecoder)

Precondition:   File pointer must point to a new block of data

Overview:       Decodes the 8x8 pixel values of all the channels
                (A multiple of 8x8 block if subsampling is used)

Input:          JPEGDECODER

Output:         Error code - '0' means no error
*****************************************************************************/
```

```
static BYTE JPEG_bDecodeOneBlock(JPEGDECODER *pJpegDecoder)
{
	BYTE bBlock, bCounter;

	IMG_vLoopCallback();
	for(bBlock = 0; bBlock < pJpegDecoder->bBlocksInOnePass; bBlock++)
	{
		BYTE bByteCount, bHuffbyte;

		if((pJpegDecoder->wRestartInterval > 0) && (pJpegDecoder->wBlockNumber ==
pJpegDecoder->wRestartInterval * pJpegDecoder->bBlocksInOnePass))
		{
			WORD wRestartWord = JPEG_wGetRestartWord(pJpegDecoder);
			if(wRestartWord < 0xFFD0 || wRestartWord > 0xFFD7)
			{
				JPEG_SendError(100);
			}
			for(bCounter = 0; bCounter < MAX_CHANNELS; bCounter++)
			{
				pJpegDecoder->asPrevDcValue[bCounter] = 0;
			}
			pJpegDecoder->wBlockNumber = 0;
		}

		for(bCounter = 0; bCounter < 64; bCounter++)
		{
			pJpegDecoder->asOneBlock[bBlock][bCounter] = 0;
		}

		pJpegDecoder->pwCurrentQuantTable =
&pJpegDecoder->awQuantTable[pJpegDecoder->abChannelQuantTableMap[pJpegDecoder->abChannelMap[
bBlock]]][0];

		/* Decode DC value */
		bByteCount = 0;
		pJpegDecoder->pbCurrentHuffSymLenTable =
&pJpegDecoder->abHuffDcSymLen[pJpegDecoder->abChannelHuffDcTableMap[pJpegDecoder->abChannelM
ap[bBlock]]][0];
		pJpegDecoder->pbCurrentHuffSymbolTable =
&pJpegDecoder->abHuffDcSymbol[pJpegDecoder->abChannelHuffDcTableMap[pJpegDecoder->abChannelM
ap[bBlock]]][0];
		pJpegDecoder->pwCurrentHuffSymStartTable =
&pJpegDecoder->awHuffDcSymStart[pJpegDecoder->abChannelHuffDcTableMap[pJpegDecoder->abChanne
lMap[bBlock]]][0];
		bHuffbyte = JPEG_bGetNextHuffByte(pJpegDecoder);
		pJpegDecoder->asOneBlock[bBlock][0] = JPEG_sGetBitsValue(pJpegDecoder,
bHuffbyte & 0x0F) + pJpegDecoder->asPrevDcValue[pJpegDecoder->abChannelMap[bBlock]];
		pJpegDecoder->asPrevDcValue[pJpegDecoder->abChannelMap[bBlock]] =
pJpegDecoder->asOneBlock[bBlock][0];

		/* Decode AC value */
		bByteCount = 1;
		pJpegDecoder->pbCurrentHuffSymLenTable =
&pJpegDecoder->abHuffAcSymLen[pJpegDecoder->abChannelHuffAcTableMap[pJpegDecoder->abChannelM
ap[bBlock]]][0];
		pJpegDecoder->pbCurrentHuffSymbolTable =
&pJpegDecoder->abHuffAcSymbol[pJpegDecoder->abChannelHuffAcTableMap[pJpegDecoder->abChannelM
ap[bBlock]]][0];
		pJpegDecoder->pwCurrentHuffSymStartTable =
&pJpegDecoder->awHuffAcSymStart[pJpegDecoder->abChannelHuffAcTableMap[pJpegDecoder->abChanne
lMap[bBlock]]][0];
		while(bByteCount < 64)
		{
			bHuffbyte = JPEG_bGetNextHuffByte(pJpegDecoder);
			bByteCount += (bHuffbyte >> 4);
			if(bHuffbyte == 0 /* EOB */)
			{
				break;
			}
			if(bByteCount > 63)
			{
				JPEG_SendError(100);
```

```
                    }
                    pJpegDecoder->asOneBlock[bBlock][abZigzag[bByteCount++]] =
JPEG_sGetBitsValue(pJpegDecoder, bHuffbyte & 0x0F);
            }
            pJpegDecoder->wBlockNumber++;
            jpeg_idct_islow(&pJpegDecoder->asOneBlock[bBlock][0],pJpegDecoder->pwCurrentQuan
tTable);
      }

//      SetVisualPage(1);

      return 0;
}

#define JPEG_WRITE_TO_DISPLAY
/*****************************************************************************
Function:       void JPEG_vInitDisplay(JPEGDECODER *pJpegDecoder)

Precondition:   None

Overview:       Initializes the (x, y) co-ordinates to (0, 0)

Input:          JPEGDECODER

Output:         None
*****************************************************************************/
static void JPEG_vInitDisplay(JPEGDECODER *pJpegDecoder)
{
 #ifndef JPEG_WRITE_TO_DISPLAY
      printf("%4u %4u\n", pJpegDecoder->wWidth, pJpegDecoder->wHeight);
 #else
     pJpegDecoder->wPrevX = 0;
     pJpegDecoder->wPrevY = 0;
//     SetActivePage(1);
 #endif
}

/*****************************************************************************
Function:       BYTE JPEG_bPaintOneBlock(JPEGDECODER *pJpegDecoder)

Precondition:   One block - 8x8 pixel data of all channels must be decoded

Overview:       Displays one 8x8 on the screen
                (A multiple of 8x8 block if subsampling is used)

Input:          JPEGDECODER

Output:         Error code - '0' means no error
*****************************************************************************/
static BYTE JPEG_bPaintOneBlock(JPEGDECODER *pJpegDecoder)
{
      BYTE bCounter;

 #ifndef JPEG_WRITE_TO_DISPLAY

      for(bCounter = 0; bCounter < 64; bCounter++)
      {
            printf("%3u %3u %3u\n", abBlockR[bCounter]&0xF8, abBlockG[bCounter]&0xFC,
abBlockB[bCounter]&0xF8);
      }

 #else

      WORD wX, wY;
      bCounter = 0;
      BYTE r,g,b;

      if(pJpegDecoder->bSubSampleType == JPEG_SAMPLE_1x1)
      {
            SHORT *psY = &pJpegDecoder->asOneBlock[0][0], *psCb =
&pJpegDecoder->asOneBlock[1][0], *psCr = &pJpegDecoder->asOneBlock[2][0];
```

```
            for(wY = 0; wY < 8; wY++)
            {
                    for(wX = 0; wX < 8; wX++)
                    {
                            LONG s1 = ((*psY) + 128)*128, s2 = (*psCb), s3 = (*psCr);
                            r = range_limit((s1 + 180*s3)>>7);
                            g = range_limit((s1 - 44*s2 - 91*s3)>>7);
                            b = range_limit((s1 + 227*s2)>>7);
                            IMG_vSetColor(r, g, b);
                            IMG_vPutPixel(pJpegDecoder->wPrevX + wX, pJpegDecoder->wPrevY +
wY);
                            psY++; psCb++; psCr++;
                    }
             }

            pJpegDecoder->wPrevX += 8;

            if(pJpegDecoder->wPrevX >= pJpegDecoder->wWidth)
            {
                    pJpegDecoder->wPrevX = 0;
                    pJpegDecoder->wPrevY += 8;
            }
     }
     else if(pJpegDecoder->bSubSampleType == JPEG_SAMPLE_1x2)
     {
            SHORT *psY, *psCb = &pJpegDecoder->asOneBlock[2][0], *psCr =
&pJpegDecoder->asOneBlock[3][0];
            BYTE bBlock, bOffsetY[2] = {0,8};

            for(bBlock = 0; bBlock < 2; bBlock++)
            {
                    psY = &pJpegDecoder->asOneBlock[bBlock][0];
                    psCb = &pJpegDecoder->asOneBlock[2][bBlock*32];
                    psCr = &pJpegDecoder->asOneBlock[3][bBlock*32];
                    for(wY = 0; wY < 8; wY++)
                    {
                            for(wX = 0; wX < 8; wX++)
                            {
                                    LONG s1 = ((*psY) + 128)*128;
                                    LONG s2 = psCb[(wY>>1)*8+wX];
                                    LONG s3 = psCr[(wY>>1)*8+wX];
                                    r = range_limit((s1 + 180*s3)>>7);
                                    g = range_limit((s1 - 44*s2 - 91*s3)>>7);
                                    b = range_limit((s1 + 227*s2)>>7);
                                    IMG_vSetColor(r, g, b);
                                    IMG_vPutPixel(pJpegDecoder->wPrevX + wX,
                                            pJpegDecoder->wPrevY + bOffsetY[bBlock] + wY);
                                    psY++;
                            }
                    }
            }

            pJpegDecoder->wPrevX += 8;

            if(pJpegDecoder->wPrevX >= pJpegDecoder->wWidth)
            {
                    pJpegDecoder->wPrevX = 0;
                    pJpegDecoder->wPrevY += 16;
            }
     }
     else if(pJpegDecoder->bSubSampleType == JPEG_SAMPLE_2x1)
     {
            SHORT *psY, *psCb = &pJpegDecoder->asOneBlock[2][0], *psCr =
&pJpegDecoder->asOneBlock[3][0];
            BYTE bBlock, bOffsetX[2] = {0,8};

            for(bBlock = 0; bBlock < 2; bBlock++)
            {
                    psY = &pJpegDecoder->asOneBlock[bBlock][0];
                    psCb = &pJpegDecoder->asOneBlock[2][bBlock*4];
                    psCr = &pJpegDecoder->asOneBlock[3][bBlock*4];
                    for(wY = 0; wY < 8; wY++)
```

```
                        {
                                for(wX = 0; wX < 8; wX++)
                                {
                                        LONG s1 = ((*psY) + 128)*128;
                                        LONG s2 = psCb[(wY*8)+(wX>>1)];
                                        LONG s3 = psCr[(wY*8)+(wX>>1)];
                                        r = range_limit((s1 + 180*s3)>>7);
                                        g = range_limit((s1 - 44*s2 - 91*s3)>>7);
                                        b = range_limit((s1 + 227*s2)>>7);
                                        IMG_vSetColor(r, g, b);
                                        IMG_vPutPixel(pJpegDecoder->wPrevX + bOffsetX[bBlock]
+ wX,
                                                pJpegDecoder->wPrevY + wY);
                                        psY++;
                                }
                        }
                }

                pJpegDecoder->wPrevX += 16;

                if(pJpegDecoder->wPrevX >= pJpegDecoder->wWidth)
                {
                        pJpegDecoder->wPrevX = 0;
                        pJpegDecoder->wPrevY += 8;
                }
        }
        else if(pJpegDecoder->bSubSampleType == JPEG_SAMPLE_2x2)
        {
                SHORT *psY, *psCb, *psCr;
                BYTE bBlock, bOffsetX[4] = {0,8,0,8}, bOffsetY[4] = {0,0,8,8}, bCbCrOffset[4] =
{0,4,32,36};

                for(bBlock = 0; bBlock < 4; bBlock++)
                {
                    WORD wXPos = pJpegDecoder->wPrevX + bOffsetX[bBlock];
                    WORD wYPos = pJpegDecoder->wPrevY + bOffsetY[bBlock];

                    psY = &pJpegDecoder->asOneBlock[bBlock][0];
                    psCb = &pJpegDecoder->asOneBlock[4][bCbCrOffset[bBlock]];
                    psCr = &pJpegDecoder->asOneBlock[5][bCbCrOffset[bBlock]];
                    for(wY = 0; wY < 8; wY++)
                    {
                        WORD wYPos2 = wYPos + wY;

                        for(wX = 0; wX < 4; wX++)
                        {
                            WORD psCxIndex = (wY>>1)*8 + wX;
                            LONG s2 = psCb[psCxIndex];
                            LONG s3 = psCr[psCxIndex];
                            WORD wX2;

                            for(wX2 = 0; wX2 < 2; wX2++)
                            {
                                LONG s1 = ((*psY++) + 128)*128;
                                r = range_limit((s1 + 180*s3)>>7);
                                g = range_limit((s1 - 44*s2 - 91*s3)>>7);
                                b = range_limit((s1 + 227*s2)>>7);
                                IMG_vSetColor(r, g, b);
                                IMG_vPutPixel(wXPos + wX*2 + wX2, wYPos2);
                            }
                        }
                    }
                }

                pJpegDecoder->wPrevX += 16;

                if(pJpegDecoder->wPrevX >= pJpegDecoder->wWidth)
                {
                        pJpegDecoder->wPrevX = 0;
                        pJpegDecoder->wPrevY += 16;
                }
        }
```

```
 #endif
     return 0;
}

#ifndef IMG_USE_NON_BLOCKING_DECODING

/*****************************************************************************
Function:       BYTE JPEG_bDecode(IMG_FILE *pfile, BOOL bFirstTime)

Precondition:   The global variables of Image decoder must be set

Overview:       This function decodes and displays a jpeg image

Input:          Image file, ignored BOOLean

Output:         Error code - '0' means no error
*****************************************************************************/
BYTE JPEG_bDecode(IMG_FILE *pfile, BOOL bFirstTime)
{
     WORD whblocks, wvblocks;
     WORD wi, wj;
     JPEGDECODER JPEG_JpegDecoder;

     JPEG_vResetDecoder(&JPEG_JpegDecoder);
     JPEG_JpegDecoder.pImageFile = pfile;
     if(JPEG_bReadHeader(&JPEG_JpegDecoder) != 0)
     {
         return JPEG_JpegDecoder.bError;
     }

     IMG_wImageWidth = JPEG_JpegDecoder.wWidth;
     IMG_wImageHeight = JPEG_JpegDecoder.wHeight;
     IMG_vSetboundaries();

     JPEG_bGenerateHuffmanTables(&JPEG_JpegDecoder);

     whblocks = JPEG_JpegDecoder.wWidth >> 3;
     wvblocks = JPEG_JpegDecoder.wHeight >> 3;

     if(whblocks * 8 < JPEG_JpegDecoder.wWidth) /* Odd sizes */
     {
         whblocks++;
     }

     if(wvblocks * 8 < JPEG_JpegDecoder.wHeight) /* Odd sizes */
     {
         wvblocks++;
     }

     if(JPEG_JpegDecoder.bSubSampleType == JPEG_SAMPLE_1x2)
     {
         wvblocks =  (wvblocks>>1) + (wvblocks&1);
     }
     else if(JPEG_JpegDecoder.bSubSampleType == JPEG_SAMPLE_2x1)
     {
         whblocks = (whblocks>>1) + (whblocks&1);
     }
     else if(JPEG_JpegDecoder.bSubSampleType == JPEG_SAMPLE_2x2)
     {
         wvblocks =  (wvblocks>>1) + (wvblocks&1);
         whblocks = (whblocks>>1) + (whblocks&1);
     }

     JPEG_vInitDisplay(&JPEG_JpegDecoder);

     for(wi = 0; wi < whblocks; wi++)
     {
            for(wj = 0; wj < wvblocks; wj++)
            {
                   IMG_vCheckAndAbort();
                   JPEG_bDecodeOneBlock(&JPEG_JpegDecoder); /* Fills a block after
```

```
correcting the zigzag, dequantizing, IDCR and color conversion to RGB */
                    JPEG_bPaintOneBlock(&JPEG_JpegDecoder); /* Sends the block to the
Graphics unit */
            }
     }
     return JPEG_JpegDecoder.bError;
}

#else

/*****************************************************************************
Function:       BYTE JPEG_bDecode(IMG_FILE *pfile, BOOL bFirstTime)

Precondition:   The global variables of Image decoder must be set

Overview:       This function decodes and displays a jpeg image

Input:          Image file, BOOLean indicating if this is the first time calling
                the JPEG_bDecode function (needed to reset internal decoding
                state variables).  If bFirstTime is TRUE, pfile must be set.
                If bFirstTime is FALSE, pfile is ignored (uses previously
                provided file handle).

Output:         Error code - '0' means not yet completed
*****************************************************************************/
BYTE JPEG_bDecode(IMG_FILE *pfile, BOOL bFirstTime)
{
    static WORD whblocks, wvblocks;
    static WORD wi, wj;
    static JPEGDECODER JPEG_JpegDecoder;
    static enum
    {
        INITIAL,
        HEADER_DECODED,
        BLOCK_DECODE,
        DECODE_DONE
    } decodestate;


    if(bFirstTime)
        decodestate = INITIAL;

    switch(decodestate)
    {

        case INITIAL:   JPEG_vResetDecoder(&JPEG_JpegDecoder);
                        JPEG_JpegDecoder.pImageFile = pfile;
                        if(JPEG_bReadHeader(&JPEG_JpegDecoder) != 0)
                        {
                            return 1;
                        }
                        decodestate = HEADER_DECODED;
                        return 0;

        case HEADER_DECODED:
                            IMG_wImageWidth = JPEG_JpegDecoder.wWidth;
                            IMG_wImageHeight = JPEG_JpegDecoder.wHeight;
                            IMG_vSetboundaries();

                            JPEG_bGenerateHuffmanTables(&JPEG_JpegDecoder);

                            whblocks = JPEG_JpegDecoder.wWidth >> 3;
                            wvblocks = JPEG_JpegDecoder.wHeight >> 3;

                            if(whblocks * 8 < JPEG_JpegDecoder.wWidth) /* Odd sizes */
                            {
                                whblocks++;
                            }

                            if(wvblocks * 8 < JPEG_JpegDecoder.wHeight) /* Odd sizes */
                            {
                                wvblocks++;
```

```
                                }

                                if(JPEG_JpegDecoder.bSubSampleType == JPEG_SAMPLE_1x2)
                                {
                                    wvblocks =  (wvblocks>>1) + (wvblocks&1);
                                }
                                else if(JPEG_JpegDecoder.bSubSampleType == JPEG_SAMPLE_2x1)
                                {
                                    whblocks = (whblocks>>1) + (whblocks&1);
                                }
                                else if(JPEG_JpegDecoder.bSubSampleType == JPEG_SAMPLE_2x2)
                                {
                                    wvblocks =  (wvblocks>>1) + (wvblocks&1);
                                    whblocks = (whblocks>>1) + (whblocks&1);
                                }

                                JPEG_vInitDisplay(&JPEG_JpegDecoder);

                                wi = 0;
                                wj = 0;

                                decodestate = BLOCK_DECODE;
                                return 0;

        case BLOCK_DECODE:      if(wi < whblocks)
                                {
                                    JPEG_bDecodeOneBlock(&JPEG_JpegDecoder); /* Fills a block
after correcting the zigzag, dequantizing, IDCR and color conversion to RGB */
                                    JPEG_bPaintOneBlock(&JPEG_JpegDecoder); /* Sends the block
to the Graphics unit */
                                    wj++;

                                    if(wj >= wvblocks)
                                    {
                                        wj = 0;
                                        wi++;
                                    }
                                    return 0;
                                }

                                decodestate = DECODE_DONE;
                                // No break needed

        case DECODE_DONE:       return 1;

        default:                return 1;
    }
}

#endif /* #ifndef IMG_USE_NON_BLOCKING_DECODING */

#endif /* #ifdef IMG_SUPPORT_JPEG */

#undef __JPEGDECODER_C__
```

12.8.8 JpegDecoder.h

This is file JpegDecoder.h.

Body Source

```
#ifndef __JPEGDECODER_H__
 #define __JPEGDECODER_H__

/*****************************************************************************
* FileName:        JpegDecoder.h
* Dependencies:    Image decoding library; project requires File System library
* Processor:       PIC24/dsPIC30/dsPIC33/PIC32MX
```

```
 * Compiler:        C30 v2.01/C32 v0.00.18
 * Company:         Microchip Technology, Inc.

  * 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.

Author                  Date           Comments
--------------------------------------------------------------------------------
Pradeep Budagutta    03-Mar-2008    First release
*******************************************************************************/

#define JPEG_SendError(x)  pJpegDecoder->bError = x; return(x); /* Point to proper error
handler routine */

#define MAX_CHANNELS      3 /* This supports only Grayscale and YCbCr images - DONT CHANGE
THIS */
#define MAX_BLOCKS        6 /* To decode one logical block, we have to decode 1 to 6 blocks
depending on channels and subsampling - DONT REDUCE THIS */
#define MAX_HUFF_TABLES   2 /* Each causes 2 tables -> One for AC and another for DC - DONT
REDUCE THIS */
#define MAX_DATA_BUF_LEN  128 /* Increase if you have more data memory */

/* Error list */
enum Errors
{
     IMAGE_FILE_NOT_AVAILABLE
};

/* JPEG Markers list */
enum Markers
{
     SOF0 = 0xC0,
     DHT  = 0xC4,
     SOI  = 0xD8,
     EOI  = 0xD9,
     SOS  = 0xDA,
     DQT  = 0xDB,
     DRI  = 0xDD,
     APP0 = 0xE0,
     COM  = 0xFE,
/* The below markers doesn't have parameters */
     TEM  = 0x01,
     RST0 = 0xD0,
     RST1 = 0xD1,
     RST2 = 0xD2,
     RST3 = 0xD3,
     RST4 = 0xD4,
     RST5 = 0xD5,
     RST6 = 0xD6,
     RST7 = 0xD7
};
```

```
/* Function prototype */
/* This function must be called after setting proper values in the global variables of
ImageDecoder.c */
BYTE JPEG_bDecode(IMG_FILE *pfile, BOOL bFirstTime);

#endif
```

12.8.9 jidctint.c

This is file jidctint.c.

Body Source

```
/*
 * jidctint.c
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying 'IJG License.pdf' file.
 *
 * This file contains a slow-but-accurate integer implementation of the
 * inverse DCT (Discrete Cosine Transform).  In the IJG code, this routine
 * must also perform dequantization of the input coefficients.
 *
 * A 2-D IDCT can be done by 1-D IDCT on each column followed by 1-D IDCT
 * on each row (or vice versa, but it's more convenient to emit a row at
 * a time).  Direct algorithms are also available, but they are much more
 * complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on an algorithm described in
 *   C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT
 *   Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics,
 *   Speech, and Signal Processing 1989 (ICASSP '89), pp. 988-991.
 * The primary algorithm described there uses 11 multiplies and 29 adds.
 * We use their alternate method with 12 multiplies and 32 adds.
 * The advantage of this method is that no data path contains more than one
 * multiplication; this allows a very simple and accurate implementation in
 * scaled fixed-point arithmetic, with a minimal number of shifts.
 */

#include "GenericTypeDefs.h"


/*
 * This module is specialized to the case DCTSIZE = 8.
 */

/*
 * The poop on this scaling stuff is as follows:
 *
 * Each 1-D IDCT step produces outputs which are a factor of sqrt(N)
 * larger than the true IDCT outputs.  The final outputs are therefore
 * a factor of N larger than desired; since N=8 this can be cured by
 * a simple right shift at the end of the algorithm.  The advantage of
 * this arrangement is that we save two multiplications per 1-D IDCT,
 * because the y0 and y4 inputs need not be divided by sqrt(N).
 *
 * We have to do addition and subtraction of the integer inputs, which
 * is no problem, and multiplication by fractional constants, which is
 * a problem to do in integer arithmetic.  We multiply all the constants
 * by CONST_SCALE and convert them to integer constants (thus retaining
 * CONST_BITS bits of precision in the constants).  After doing a
 * multiplication we have to divide the product by CONST_SCALE, with proper
 * rounding, to produce the correct output.  This division can be done
 * cheaply as a right shift of CONST_BITS bits.  We postpone shifting
 * as long as possible so that partial sums can be added together with
 * full fractional precision.
```

```
 *
 * The outputs of the first pass are scaled up by PASS1_BITS bits so that
 * they are represented to better-than-integral precision.  These outputs
 * require BITS_IN_JSAMPLE + PASS1_BITS + 3 bits; this fits in a 16-bit word
 * with the recommended scaling.  (To scale up 12-bit sample data further, an
 * intermediate INT32 array would be needed.)
 *
 * To avoid overflow of the 32-bit intermediate results in pass 2, we must
 * have BITS_IN_JSAMPLE + CONST_BITS + PASS1_BITS <= 26.  Error analysis
 * shows that the values given below are the most effective.
 */
#define DCTSIZE         8   /* The basic DCT block is 8x8 samples */
#define DCTSIZE2        64  /* DCTSIZE squared; # of elements in a block */
#define BITS_IN_JSAMPLE  8
#define NO_ZERO_ROW_TEST
#define DCTELEM LONG

#define CONST_BITS  13
#define PASS1_BITS  2

#define INT32 LONG

/* Some C compilers fail to reduce "FIX(constant)" at compile time, thus
 * causing a lot of useless floating-point operations at run time.
 * To get around this we use the following pre-calculated constants.
 * If you change CONST_BITS you may want to add appropriate values.
 * (With a reasonable C compiler, you can just rely on the FIX() macro...)
 */

#define FIX_0_298631336  ((INT32)  2446)    /* FIX(0.298631336) */
#define FIX_0_390180644  ((INT32)  3196)    /* FIX(0.390180644) */
#define FIX_0_541196100  ((INT32)  4433)    /* FIX(0.541196100) */
#define FIX_0_765366865  ((INT32)  6270)    /* FIX(0.765366865) */
#define FIX_0_899976223  ((INT32)  7373)    /* FIX(0.899976223) */
#define FIX_1_175875602  ((INT32)  9633)    /* FIX(1.175875602) */
#define FIX_1_501321110  ((INT32)  12299)   /* FIX(1.501321110) */
#define FIX_1_847759065  ((INT32)  15137)   /* FIX(1.847759065) */
#define FIX_1_961570560  ((INT32)  16069)   /* FIX(1.961570560) */
#define FIX_2_053119869  ((INT32)  16819)   /* FIX(2.053119869) */
#define FIX_2_562915447  ((INT32)  20995)   /* FIX(2.562915447) */
#define FIX_3_072711026  ((INT32)  25172)   /* FIX(3.072711026) */


/* Multiply an INT32 variable by an INT32 constant to yield an INT32 result.
 * For 8-bit samples with the recommended scaling, all the variable
 * and constant values involved are no more than 16 bits wide, so a
 * 16x16->32 bit multiply can be used instead of a full 32x32 multiply.
 * For 12-bit samples, a full 32-bit multiplication will be needed.
 */

#define DESCALE(x,n)  ((x) + ((LONG)0x01 << ((n)-1)))>>(n)
#define MULTIPLY(var,constant)  ((LONG)(var) * (constant));
#define range_limit(x) ((x)<-128)?-128:((x)>127)?127:(x)

/* Dequantize a coefficient by multiplying it by the multiplier-table
 * entry; produce an int result.  In this module, both inputs and result
 * are 16 bits or less, so either int or short multiply will work.
 */

#define DEQUANTIZE(coef,quantval)  ((LONG)(coef) * (quantval))


/*
 * Perform dequantization and inverse DCT on one block of coefficients.
 */

void jpeg_idct_islow (SHORT *inbuf, WORD *quantptr)
{
  LONG tmp0, tmp1, tmp2, tmp3;
  LONG tmp10, tmp11, tmp12, tmp13;
  LONG z1, z2, z3, z4, z5;
```

```
    BYTE ctr;
    SHORT *inptr = inbuf, *outptr;
    DCTELEM *wsptr;
    DCTELEM workspace[DCTSIZE2];  /* buffers data between passes */

    wsptr = workspace;

    /* Pass 1: process columns from input, store into work array. */
    /* Note results are scaled up by sqrt(8) compared to a true IDCT; */
    /* furthermore, we scale the results by 2**PASS1_BITS. */

    for (ctr = DCTSIZE; ctr > 0; ctr--) {
      /* Due to quantization, we will usually find that many of the input
       * coefficients are zero, especially the AC terms.  We can exploit this
       * by short-circuiting the IDCT calculation for any column in which all
       * the AC terms are zero.  In that case each output is equal to the
       * DC coefficient (with scale factor as needed).
       * With typical images and quantization tables, half or more of the
       * column DCT calculations can be simplified this way.
       */

      if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*2] == 0 &&
      inptr[DCTSIZE*3] == 0 && inptr[DCTSIZE*4] == 0 &&
      inptr[DCTSIZE*5] == 0 && inptr[DCTSIZE*6] == 0 &&
      inptr[DCTSIZE*7] == 0) {
        /* AC terms all zero */
        LONG dcval = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]) << PASS1_BITS;

        wsptr[DCTSIZE*0] = dcval;
        wsptr[DCTSIZE*1] = dcval;
        wsptr[DCTSIZE*2] = dcval;
        wsptr[DCTSIZE*3] = dcval;
        wsptr[DCTSIZE*4] = dcval;
        wsptr[DCTSIZE*5] = dcval;
        wsptr[DCTSIZE*6] = dcval;
        wsptr[DCTSIZE*7] = dcval;

        inptr++;          /* advance pointers to next column */
        quantptr++;
        wsptr++;
        continue;
      }

      /* Even part: reverse the even part of the forward DCT. */
      /* The rotator is sqrt(2)*c(-6). */

      z2 = DEQUANTIZE(inptr[DCTSIZE*2], quantptr[DCTSIZE*2]);
      z3 = DEQUANTIZE(inptr[DCTSIZE*6], quantptr[DCTSIZE*6]);

      z1 = MULTIPLY(z2 + z3, FIX_0_541196100);
      tmp2 = z1 + MULTIPLY(z3, - FIX_1_847759065);
      tmp3 = z1 + MULTIPLY(z2, FIX_0_765366865);

      z2 = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);
      z3 = DEQUANTIZE(inptr[DCTSIZE*4], quantptr[DCTSIZE*4]);

      tmp0 = (z2 + z3) << CONST_BITS;
      tmp1 = (z2 - z3) << CONST_BITS;

      tmp10 = tmp0 + tmp3;
      tmp13 = tmp0 - tmp3;
      tmp11 = tmp1 + tmp2;
      tmp12 = tmp1 - tmp2;

      /* Odd part per figure 8; the matrix is unitary and hence its
       * transpose is its inverse.  i0..i3 are y7,y5,y3,y1 respectively.
       */

      tmp0 = DEQUANTIZE(inptr[DCTSIZE*7], quantptr[DCTSIZE*7]);
      tmp1 = DEQUANTIZE(inptr[DCTSIZE*5], quantptr[DCTSIZE*5]);
      tmp2 = DEQUANTIZE(inptr[DCTSIZE*3], quantptr[DCTSIZE*3]);
      tmp3 = DEQUANTIZE(inptr[DCTSIZE*1], quantptr[DCTSIZE*1]);
```

```
    z1 = tmp0 + tmp3;
    z2 = tmp1 + tmp2;
    z3 = tmp0 + tmp2;
    z4 = tmp1 + tmp3;
    z5 = MULTIPLY(z3 + z4, FIX_1_175875602); /* sqrt(2) * c3 */

    tmp0 = MULTIPLY(tmp0, FIX_0_298631336); /* sqrt(2) * (-c1+c3+c5-c7) */
    tmp1 = MULTIPLY(tmp1, FIX_2_053119869); /* sqrt(2) * ( c1+c3-c5+c7) */
    tmp2 = MULTIPLY(tmp2, FIX_3_072711026); /* sqrt(2) * ( c1+c3+c5-c7) */
    tmp3 = MULTIPLY(tmp3, FIX_1_501321110); /* sqrt(2) * ( c1+c3-c5-c7) */
    z1 = MULTIPLY(z1, - FIX_0_899976223); /* sqrt(2) * (c7-c3) */
    z2 = MULTIPLY(z2, - FIX_2_562915447); /* sqrt(2) * (-c1-c3) */
    z3 = MULTIPLY(z3, - FIX_1_961570560); /* sqrt(2) * (-c3-c5) */
    z4 = MULTIPLY(z4, - FIX_0_390180644); /* sqrt(2) * (c5-c3) */

    z3 += z5;
    z4 += z5;

    tmp0 += z1 + z3;
    tmp1 += z2 + z4;
    tmp2 += z2 + z3;
    tmp3 += z1 + z4;

    /* Final output stage: inputs are tmp10..tmp13, tmp0..tmp3 */

    wsptr[DCTSIZE*0] = (LONG) DESCALE((tmp10 + tmp3), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*7] = (LONG) DESCALE((tmp10 - tmp3), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*1] = (LONG) DESCALE((tmp11 + tmp2), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*6] = (LONG) DESCALE((tmp11 - tmp2), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*2] = (LONG) DESCALE((tmp12 + tmp1), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*5] = (LONG) DESCALE((tmp12 - tmp1), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*3] = (LONG) DESCALE((tmp13 + tmp0), (CONST_BITS-PASS1_BITS));
    wsptr[DCTSIZE*4] = (LONG) DESCALE((tmp13 - tmp0), (CONST_BITS-PASS1_BITS));

    inptr++;            /* advance pointers to next column */
    quantptr++;
    wsptr++;
  }

  /* Pass 2: process rows from work array, store into output array. */
  /* Note that we must descale the results by a factor of 8 == 2**3, */
  /* and also undo the PASS1_BITS scaling. */

  wsptr = workspace;
  outptr = &inbuf[0];
  for (ctr = 0; ctr < DCTSIZE; ctr++) {
    /* Rows of zeroes can be exploited in the same way as we did with columns.
     * However, the column calculation has created many nonzero AC terms, so
     * the simplification applies less often (typically 5% to 10% of the time).
     * On machines with very fast multiplication, it's possible that the
     * test takes more time than it's worth.  In that case this section
     * may be commented out.
     */

#ifndef NO_ZERO_ROW_TEST
    if (wsptr[1] == 0 && wsptr[2] == 0 && wsptr[3] == 0 && wsptr[4] == 0 &&
    wsptr[5] == 0 && wsptr[6] == 0 && wsptr[7] == 0) {
      /* AC terms all zero */
      JSAMPLE dcval = range_limit[(LONG) DESCALE((INT32) wsptr[0], PASS1_BITS+3)
                  & RANGE_MASK];

      outptr[0] = dcval;
      outptr[1] = dcval;
      outptr[2] = dcval;
      outptr[3] = dcval;
      outptr[4] = dcval;
      outptr[5] = dcval;
      outptr[6] = dcval;
      outptr[7] = dcval;

      wsptr += DCTSIZE;     /* advance pointer to next row */
```

```
        continue;
      }
  #endif

      /* Even part: reverse the even part of the forward DCT. */
      /* The rotator is sqrt(2)*c(-6). */

      z2 = (INT32) wsptr[2];
      z3 = (INT32) wsptr[6];

      z1 = MULTIPLY(z2 + z3, FIX_0_541196100);
      tmp2 = z1 + MULTIPLY(z3, - FIX_1_847759065);
      tmp3 = z1 + MULTIPLY(z2, FIX_0_765366865);

      tmp0 = ((INT32) wsptr[0] + (INT32) wsptr[4]) << CONST_BITS;
      tmp1 = ((INT32) wsptr[0] - (INT32) wsptr[4]) << CONST_BITS;

      tmp10 = tmp0 + tmp3;
      tmp13 = tmp0 - tmp3;
      tmp11 = tmp1 + tmp2;
      tmp12 = tmp1 - tmp2;

      /* Odd part per figure 8; the matrix is unitary and hence its
       * transpose is its inverse.  i0..i3 are y7,y5,y3,y1 respectively.
       */

      tmp0 = (INT32) wsptr[7];
      tmp1 = (INT32) wsptr[5];
      tmp2 = (INT32) wsptr[3];
      tmp3 = (INT32) wsptr[1];

      z1 = tmp0 + tmp3;
      z2 = tmp1 + tmp2;
      z3 = tmp0 + tmp2;
      z4 = tmp1 + tmp3;
      z5 = MULTIPLY(z3 + z4, FIX_1_175875602); /* sqrt(2) * c3 */

      tmp0 = MULTIPLY(tmp0, FIX_0_298631336); /* sqrt(2) * (-c1+c3+c5-c7) */
      tmp1 = MULTIPLY(tmp1, FIX_2_053119869); /* sqrt(2) * ( c1+c3-c5+c7) */
      tmp2 = MULTIPLY(tmp2, FIX_3_072711026); /* sqrt(2) * ( c1+c3+c5-c7) */
      tmp3 = MULTIPLY(tmp3, FIX_1_501321110); /* sqrt(2) * ( c1+c3-c5-c7) */
      z1 = MULTIPLY(z1, - FIX_0_899976223); /* sqrt(2) * (c7-c3) */
      z2 = MULTIPLY(z2, - FIX_2_562915447); /* sqrt(2) * (-c1-c3) */
      z3 = MULTIPLY(z3, - FIX_1_961570560); /* sqrt(2) * (-c3-c5) */
      z4 = MULTIPLY(z4, - FIX_0_390180644); /* sqrt(2) * (c5-c3) */

      z3 += z5;
      z4 += z5;

      tmp0 += z1 + z3;
      tmp1 += z2 + z4;
      tmp2 += z2 + z3;
      tmp3 += z1 + z4;

      /* Final output stage: inputs are tmp10..tmp13, tmp0..tmp3 */

      outptr[0] = (SHORT)range_limit((LONG) DESCALE(tmp10 + tmp3, CONST_BITS+PASS1_BITS+3));
      outptr[7] = (SHORT)range_limit((LONG) DESCALE(tmp10 - tmp3, CONST_BITS+PASS1_BITS+3));
      outptr[1] = (SHORT)range_limit((LONG) DESCALE(tmp11 + tmp2, CONST_BITS+PASS1_BITS+3));
      outptr[6] = (SHORT)range_limit((LONG) DESCALE(tmp11 - tmp2, CONST_BITS+PASS1_BITS+3));
      outptr[2] = (SHORT)range_limit((LONG) DESCALE(tmp12 + tmp1, CONST_BITS+PASS1_BITS+3));
      outptr[5] = (SHORT)range_limit((LONG) DESCALE(tmp12 - tmp1, CONST_BITS+PASS1_BITS+3));
      outptr[3] = (SHORT)range_limit((LONG) DESCALE(tmp13 + tmp0, CONST_BITS+PASS1_BITS+3));
      outptr[4] = (SHORT)range_limit((LONG) DESCALE(tmp13 - tmp0, CONST_BITS+PASS1_BITS+3));

      outptr += DCTSIZE;   /* advance pointer to next row */
      wsptr += DCTSIZE;       /* advance pointer to next row */
    }
  }
```

12.9 _BMPDECODER Structure

File

BmpDecoder.c (see page 457)

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

Module

Image Decoders (see page 449)

Description

DATA STRUCTURES

12.10 _GIFDECODER Structure

File

GifDecoder.c (see page 464)

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 blGloabalColorTableFlag : 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

Module

Image Decoders (see page 449)

Description

DATA STRUCTURES

12.11 _JPEGDECODER Structure

File

JpegDecoder.c (see page 487)

C

```
struct _JPEGDECODER {
  IMG_FILE * pImageFile;
  BYTE blJFIF;
  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 * pImageFile;	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 (see page 448) in pixels
WORD wHeight;	Height (see page 447) 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

Module

Image Decoders (see page 449)

Description

DATA STRUCTURES

13 Miscellaneous Topics

This section contains common procedures to start using the library or to modify the library.

13.1 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: Creating the directory structure - In order to easily maintain the library components the following directory structure is recommended.

```
Microchip Solutions
    |
    MyApplication
    Microchip
        |
        Help
        Graphics
        Include
            |
            Graphics
```

- MyApplication directory can be any directory that you specify.
- 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 C files.
- Under Microchip directory, the Include directory will have another Graphics directory that will contain all the header files with the exception of one. The **"GenericTypeDefs.h"** file will be located in *Microchip->Include* directory.

Step 2: Adding library files - Before creating the MPLAB Project follow the following steps add library files to your created directory tree.

1. Place the following C files in the *Microchip->Graphics* directory.

Button.c (see page 644)	**ListBox.c (see page 782)**
CheckBox.c (see page 725)	**Meter.c (see page 804)**
EditBox.c (see page 746)	**Picture.c (see page 826)**
GOL.c (see page 531)	**Primitive.c (see page 941)**
GOLFontDefault.c (see page 579)	**ProgressBar.c (see page 834)**
GroupBox.c (see page 773)	**RadioButton.c (see page 858)**
RoundDial.c (see page 844)	**Slider.c (see page 872)**
LGDP4531_R61505_S6D0129_S6D0139_SPFD5408.c	**StaticText.c (see page 895)**
Window.c (see page 932)	

2. Place the **GenericTypeDefs.h** in the *Microchip->Include* directory.
3. Place the following header files in the *Microchip->Include->Graphics* directory.

Button.h (see page 655)	**ListBox.h (see page 794)**
CheckBox.h (see page 732)	**Meter.h (see page 817)**
EditBox.h (see page 753)	**Picture.h (see page 830)**
GOL.h (see page 552)	**Primitive.h (see page 1008)**
Graphics.h (see page 620)	**ProgressBar.h (see page 840)**
GroupBox.h (see page 778)	**RadioButton.h (see page 866)**
RoundDial.h (see page 852)	**Slider.h (see page 886)**
LGDP4531_R61505_S6D0129_S6D0139_SPFD5408.h	**StaticText.h (see page 900)**
Window.h (see page 937)	

Step 3: 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 (see page 530)** and click **Add Files (see page 530)**. Browse to *MyApplication->Microchip->Graphics* directory. Select all the c files and click **Open**.
6. Right click on the **Header Files (see page 530)** and click **Add Files (see page 530)**. Browse to *MyApplication->Microchip->Include* directory. Select **GenericTypeDefs.h** file and click **Open**.
7. Right click on the **Header Files (see page 530)** and click **Add Files (see page 530)**. Browse to *MyApplication->Microchip->Include->Graphics* directory. Select all the h files and click **Open**.

Step 4: 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. ..\..\GraphicsProject1 (you already did this in previous step)
2. ..\Microchip\Include
3. ..\Microchip\Include\Graphics
4. ..\Include

5. ..\Include\Graphics
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 (see page 530)\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.

4.9. Click **Apply** and then **OK**.

Step 5: Create your application files.

1. In MPLAB Project, select **File** tab and click **New**. The file editor will open.

2. Create your **GraphicsConfig.h (see page 928)** 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 **MyApplication** 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 **MyApplication** directory.

6. After saving the two header files, add these files to your project. To add just perform the same sequence in **Step 3.6**.

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,                    // object's ID
              20, 105, 300, 150,    // object's dimension
              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 **MyApplication** directory.

9. After saving the application file, add the file to your project. To add just perform the same sequence in **Step 3.5**.

Step 6: 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 (see page 237) and Button (see page 65).

Step 7: 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 (see page 28) sub-section in the Getting Started (see page 21) section to download your application to the Explorer 16 board using your generated hex file.

13.2 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 GOLFontDefault (see page 332):

1. Open the generated font file (generated by the "Graphics Resource Converter" utility) and change the font structure name to **GOLFontDefault (see page 332)** (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;}GOLFontDefault =
{0,L1191258975};
const char L1191258975[] __attribute__((aligned(2))) = {
```

2. Remove the file **GOLFONTDefault.c** in the project and replace it with your own font file that contains the **GOLFontDefault (see page 332)**.
3. Build and test your new font.

Replacing the GOLFontDefault (see page 332) with any user defined fonts

1. Open the **GraphicsConfig.h (see page 928)** 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 (see page 552) and GOLFontDefault.c (see page 579) files checks if FONTDEFAULT (see page 332) is defined. If it is, it skips the declaration of the GOLFontDefault (see page 332) and uses the user defined font.

13.3 Using Primitive Rendering Functions in Blocking and Non-Blocking Modes

Basic rendering functions such as Line(), Rectangle (see page 353)(), Circle (see page 355)() 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 (see page 41)".

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 (see page 928) 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 (see page 353)() and Bar (see page 352)() 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 (see page 42) 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 (see page 42) 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 (see page 353) and Bar (see page 352) were drawn successfully.

13.4 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 x 240) x (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 x 240) x (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 (see page 394), GREEN (see

page 392), 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 (see page 433) _GOL_Palette_Demo;" in the application. See MainDemo.c in "Graphics Object Layer Palette Demo". Set the palette using the APIs SetPaletteBpp (see page 432)() and SetPalette (see page 396)() and then enable the palette using the API EnablePalette (see page 429)() 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 Graphis 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.

14 Files

14.1 GOL Layer

Files

Name	Description
GOL.c (see page 531)	This is file GOL.c.
GOL.h (see page 552)	This is file GOL.h.
GOLFontDefault.c (see page 579)	This is file GOLFontDefault.c.
Graphics.h (see page 620)	This is file Graphics.h.
ScanCodes.h (see page 622)	This is file ScanCodes.h.
AnalogClock.c (see page 625)	This is file AnalogClock.c.
AnalogClock.h (see page 637)	This is file AnalogClock.h.
Button.c (see page 644)	This is file Button.c.
Button.h (see page 655)	This is file Button.h.
Chart.c (see page 663)	This is file Chart.c.
Chart.h (see page 708)	This is file Chart.h.
CheckBox.c (see page 725)	This is file CheckBox.c.
CheckBox.h (see page 732)	This is file CheckBox.h.
DigitalMeter.c (see page 736)	This is file DigitalMeter.c.
DigitalMeter.h (see page 742)	This is file DigitalMeter.h.
EditBox.c (see page 746)	This is file EditBox.c.
EditBox.h (see page 753)	This is file EditBox.h.
Grid.c (see page 758)	This is file Grid.c.
Grid.h (see page 766)	This is file Grid.h.
GroupBox.c (see page 773)	This is file GroupBox.c.
GroupBox.h (see page 778)	This is file GroupBox.h.
ListBox.c (see page 782)	This is file ListBox.c.
ListBox.h (see page 794)	This is file ListBox.h.
Meter.c (see page 804)	This is file Meter.c.
Meter.h (see page 817)	This is file Meter.h.
Picture.c (see page 826)	This is file Picture.c.
Picture.h (see page 830)	This is file Picture.h.
ProgressBar.c (see page 834)	This is file ProgressBar.c.
ProgressBar.h (see page 840)	This is file ProgressBar.h.
RoundDial.c (see page 844)	This is file RoundDial.c.
RoundDial.h (see page 852)	This is file RoundDial.h.
RadioButton.c (see page 858)	This is file RadioButton.c.
RadioButton.h (see page 866)	This is file RadioButton.h.
Slider.c (see page 872)	This is file Slider.c.
Slider.h (see page 886)	This is file Slider.h.
StaticText.c (see page 895)	This is file StaticText.c.
StaticText.h (see page 900)	This is file StaticText.h.

TextEntry.c (see page 904)	This is file TextEntry.c.
TextEntry.h (see page 919)	This is file TextEntry.h.
GraphicsConfig.h (see page 928)	This is file GraphicsConfig.h.
Window.c (see page 932)	This is file Window.c.
Window.h (see page 937)	This is file Window.h.

Description

Lists all files describing the Graphic Object Layer.

14.1.1 GOL.c

This is file GOL.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *****************************************************************************
 * FileName:        GOL.c
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07               Version 1.0 release
 * 06/29/09              Added multi-line support for Buttons.
 * 03/12/09              Added Double Buffering Support
 * 04/29/10              Fixed GOLGetFocusNext() issue.
 * 06/07/10              To save drawing time, add check on panel to skip
 *                       drawing the panel face and frame if the bitmap
 *                       used is greater than area of the panel.
 * 09/08/10              Removed redundant code in GOLRedrawRec().
 *                       Added EditBox in GOLCanBeFocused().
 * 03/30/11              In 1 BPP mode: Removed all references to WHITE and
 *                       BLACK color. Replaced them to use the emboss light
 *                       and dark color instead.
 *****************************************************************************/
#include "Graphics/Graphics.h"
```

```
#ifdef USE_GOL

#include <string.h>

#ifdef USE_TRANSITION_EFFECTS

    #include "Graphics/Transitions.h"

    static BYTE TransitionPendingStatus;
#endif

// Pointer to the current linked list of objects displayed and receiving messages
OBJ_HEADER  *_pGolObjects = NULL;

// Pointer to the default GOL scheme (created in GOLInit())
GOL_SCHEME  *_pDefaultGolScheme = NULL;

// Pointer to the object receiving keyboard input
OBJ_HEADER  *_pObjectFocused = NULL;

#ifdef USE_GRADIENT
GFX_GRADIENT_STYLE _gradientScheme;
#endif

    #ifdef USE_FOCUS

/*********************************************************************
* Function: OBJ_HEADER *GOLGetFocusPrev()
*
* Overview: This function returns the pointer to the previous object
*           in the active linked list which is able to receive
*           keyboard input.
*
* PreCondition: none
*
* Input: none
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
OBJ_HEADER *GOLGetFocusPrev(void)
{
    OBJ_HEADER  *pPrevObj;
    OBJ_HEADER  *pCurObj;
    OBJ_HEADER  *pFocusedObj;

    if(_pGolObjects == NULL)
        return (NULL);

    if(_pObjectFocused == NULL)
    {
        pFocusedObj = _pGolObjects;
    }
    else
    {
        pFocusedObj = _pObjectFocused;
    }

    pPrevObj = NULL;
    pCurObj = pFocusedObj;

    while(1)
    {
        if(GOLCanBeFocused(pCurObj))
            pPrevObj = pCurObj;

        if(pCurObj->pNxtObj == NULL)
```

```
            if(_pGolObjects == pFocusedObj)
                return (pPrevObj);

        if(pCurObj->pNxtObj == pFocusedObj)
            return (pPrevObj);

        pCurObj = (OBJ_HEADER *)pCurObj->pNxtObj;

        if(pCurObj == NULL)
            pCurObj = _pGolObjects;
    }
}

/*********************************************************************
* Function: OBJ_HEADER *GOLGetFocusNext()
*
* Overview: This function returns the pointer to the next object
*           in the active linked list which is able to receive
*           keyboard input.
*
* PreCondition: none
*
* Input: none
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
OBJ_HEADER *GOLGetFocusNext(void)
{
    OBJ_HEADER  *pNextObj, *pObjStart;

    if(_pGolObjects == NULL)
        return (NULL);

    if(_pObjectFocused == NULL)
    {
        pNextObj = _pGolObjects;
    }
    else
    {
        pNextObj = _pObjectFocused;
    }

    pObjStart = pNextObj;

    do
    {
        pNextObj = (OBJ_HEADER *)pNextObj->pNxtObj;

        if(pNextObj == NULL)
            pNextObj = _pGolObjects;

        if(GOLCanBeFocused(pNextObj))
            break;
    } while(pNextObj != pObjStart);

    // if we reached the starting point and the starting point cannot
    // be focused, then all objects cannot be focused. return NULL
    if(!GOLCanBeFocused(pNextObj))
        return NULL;

    return (pNextObj);
}

/*********************************************************************
* Function: void GOLSetFocus(OBJ_HEADER* object)
*
```

```
* PreCondition: none
*
* Input: pointer to the object to be focused
*
* Output:
*
* Side Effects: none
*
* Overview: moves keyboard focus to the object
*
* Note: none
*
*********************************************************************/
void GOLSetFocus(OBJ_HEADER *object)
{
    if(!GOLCanBeFocused(object))
        return;

    if(_pObjectFocused != NULL)
    {
        ClrState(_pObjectFocused, FOCUSED);
        SetState(_pObjectFocused, DRAW_FOCUS);
    }

    SetState(object, DRAW_FOCUS | FOCUSED);

    _pObjectFocused = object;
}

/*********************************************************************
* Function: WORD GOLCanBeFocused(OBJ_HEADER* object)
*
* PreCondition: none
*
* Input: pointer to the object
*
* Output: non-zero if the object supports keyboard focus, zero if not
*
* Side Effects: none
*
* Overview: checks if object support keyboard focus
*
* Note: none
*
*********************************************************************/
WORD GOLCanBeFocused(OBJ_HEADER *object)
{
    if
    (
        (object->type == OBJ_BUTTON) ||
        (object->type == OBJ_CHECKBOX) ||
        (object->type == OBJ_RADIOBUTTON) ||
        (object->type == OBJ_LISTBOX) ||
        (object->type == OBJ_SLIDER) ||
        (object->type == OBJ_EDITBOX)
    )
    {
        if(!GetState(object, DISABLED))
            return (1);
    }

    return (0);
}

    #endif

/*********************************************************************
* Function: GOL_SCHEME *GOLCreateScheme(void)
*
* PreCondition: none
*
* Input: none
```

```
 *
 * Output: pointer to scheme object
 *
 * Side Effects: none
 *
 * Overview: creates a color scheme object and assign default colors
 *
 * Note: none
 *
 ********************************************************************/
GOL_SCHEME *GOLCreateScheme(void)
{
    GOL_SCHEME  *pTemp;

    pTemp = (GOL_SCHEME *)GFX_malloc(sizeof(GOL_SCHEME));

    if(pTemp != NULL)
    {
        memcpy(pTemp, &GFX_SCHEMEDEFAULT, sizeof(GOL_SCHEME));
    }

    return (pTemp);
}

/********************************************************************
* Function: void GOLInit()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: initializes GOL
*
* Note: none
*
********************************************************************/
void GOLInit(void)
{

    // Initialize display
    InitGraph();

    // Initialize the default GOL scheme
    _pDefaultGolScheme = GOLCreateScheme();

    #ifdef USE_TRANSITION_EFFECTS

        TransitionPendingStatus = 0;

    #endif
}

/********************************************************************
* Function: void GOLFree()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: frees memory of all objects in the current linked list
*           and starts a new linked list. This function must be
*           called only inside the GOLDrawCallback()function when
*           using GOLDraw() and GOLMsg() to manage rendering and
*           message processing.
```

```
 *
 * Note: drawing and messaging must be suspended
 *
 ********************************************************************/
void GOLFree(void)
{
    OBJ_HEADER  *pNextObj;
    OBJ_HEADER  *pCurrentObj;

    pCurrentObj = _pGolObjects;
    while(pCurrentObj != NULL)
    {
        pNextObj = (OBJ_HEADER *)pCurrentObj->pNxtObj;

        // check if there are additional items to free
        if(pCurrentObj->FreeObj)
            pCurrentObj->FreeObj(pCurrentObj);

        GFX_free(pCurrentObj);
        pCurrentObj = pNextObj;
    }

    GOLNewList();

    #ifdef USE_DOUBLE_BUFFERING

        InvalidateAll();

    #endif
}

/********************************************************************
 * Function: BOOL GOLDeleteObject(OBJ_HEADER * object)
 *
 * PreCondition: none
 *
 * Input: pointer to the object
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: Deletes an object to the linked list objects for the current screen.
 *
 * Note: none
 *
 ********************************************************************/
BOOL GOLDeleteObject(OBJ_HEADER *object)
{
    if(!_pGolObjects)
        return (FALSE);

    if(object == _pGolObjects)
    {
        _pGolObjects = (OBJ_HEADER *)object->pNxtObj;
    }
    else
    {
        OBJ_HEADER  *curr;
        OBJ_HEADER  *prev;

        curr = (OBJ_HEADER *)_pGolObjects->pNxtObj;
        prev = (OBJ_HEADER *)_pGolObjects;

        while(curr)
        {
            if(curr == object)
                break;

            prev = (OBJ_HEADER *)curr;
            curr = (OBJ_HEADER *)curr->pNxtObj;
        }
```

```
        if(!curr)
            return (FALSE);

        prev->pNxtObj = curr->pNxtObj;
    }

    if(object->FreeObj)
        object->FreeObj(object);

    GFX_free(object);

    return (TRUE);
}

/*********************************************************************
* Function: BOOL GOLDeleteObject(OBJ_HEADER * object)
*
* PreCondition: none
*
* Input: pointer to the object
*
* Output: none
*
* Side Effects: none
*
* Overview: Deletes an object to the linked list objects for the current screen using
*           the given ID to search for the object.
*
* Note: none
*
*********************************************************************/
BOOL GOLDeleteObjectByID(WORD ID)
{
    OBJ_HEADER  *object;

    object = GOLFindObject(ID);

    if(!object)
        return (FALSE);

    return (GOLDeleteObject(object));
}

/*********************************************************************
* Function: OBJ_HEADER* GOLFindObject(WORD ID)
*
* PreCondition: none
*
* Input: object ID
*
* Output: pointer to the object
*
* Side Effects: none
*
* Overview: searches for the object by its ID in the current objects linked list,
*           returns NULL if the object is not found
*
* Note: none
*
*********************************************************************/
OBJ_HEADER *GOLFindObject(WORD ID)
{
    OBJ_HEADER  *pNextObj;

    pNextObj = _pGolObjects;
    while(pNextObj != NULL)
    {
        if(pNextObj->ID == ID)
        {
            return (pNextObj);
        }
```

```
        pNextObj = (OBJ_HEADER *)pNextObj->pNxtObj;
    }

    return (NULL);
}

/*****************************************************************
* Function: void GOLAddObject(OBJ_HEADER * object)
*
* PreCondition: none
*
* Input: pointer to the object
*
* Output: none
*
* Side Effects: none
*
* Overview: adds an object to the linked list objects for the current screen.
*           Object will be drawn and will receive messages.
*
* Note: none
*
*****************************************************************/
void GOLAddObject(OBJ_HEADER *object)
{
    OBJ_HEADER  *pNextObj;

    if(_pGolObjects == NULL)
    {
        _pGolObjects = object;
    }
    else
    {
        pNextObj = _pGolObjects;
        while(pNextObj->pNxtObj != NULL)
        {
            pNextObj = (OBJ_HEADER *)pNextObj->pNxtObj;
        }

        pNextObj->pNxtObj = (void *)object;
    }

    object->pNxtObj = NULL;
}

/*****************************************************************
* Function: WORD GOLDraw()
*
* PreCondition: none
*
* Input: none
*
* Output: non-zero if drawing is complete
*
* Side Effects: none
*
* Overview: redraws objects in the current linked list
*
* Note: none
*
*****************************************************************/
WORD GOLDraw(void)
{
    static OBJ_HEADER   *pCurrentObj = NULL;
    SHORT               done;

    #ifdef USE_DOUBLE_BUFFERING
    static BYTE         DisplayUpdated = 0;
    if(IsDisplayUpdatePending())
    {
        return 0;
```

```
    }
    #endif // USE_DOUBLE_BUFFERING

    if(pCurrentObj == NULL)
    {
        if(GOLDrawCallback())
        {
            #ifdef USE_TRANSITION_EFFECTS
                if(TransitionPendingStatus)
                {
                    #ifdef PIC24FJ256DA210_DEV_BOARD
                    TransitionPendingStatus = 0;
                    while(IsDisplayUpdatePending());
                    GFXExecutePendingTransition(_drawbuffer, (_drawbuffer == GFX_BUFFER1)?
GFX_BUFFER2: GFX_BUFFER1);
                    #endif
                }
            #endif

            // It's last object jump to head
            pCurrentObj = _pGolObjects;

            #ifdef USE_DOUBLE_BUFFERING
            if(DisplayUpdated)
            {
                RequestDisplayUpdate();
                DisplayUpdated = 0;
                return(0);
            }
            #endif //USE_DOUBLE_BUFFERING
        }
        else
        {
            return (0);     // drawing is not done
        }
    }

    done = 0;
    while(pCurrentObj != NULL)
    {
        if(IsObjUpdated(pCurrentObj))
        {
            done = pCurrentObj->DrawObj(pCurrentObj);

            if(done)
            {
                GOLDrawComplete(pCurrentObj);

            #ifdef USE_DOUBLE_BUFFERING

                InvalidateRectangle(pCurrentObj->left, pCurrentObj->top,
                                    pCurrentObj->right, pCurrentObj->bottom);
                DisplayUpdated = 1;

            #endif //USE_DOUBLE_BUFFERING

            }
            else
            {
                return (0); // drawing is not done
            }
        }

        pCurrentObj = (OBJ_HEADER *)pCurrentObj->pNxtObj;
    }

    #if defined(USE_TRANSITION_EFFECTS) && defined(USE_DOUBLE_BUFFERING)

        TransitionPendingStatus = GFXIsTransitionPending();
        InvalidateAll(); //If needed, invalidate only the transition rectangle instead

    #endif
```

```
    return (1);             // drawing is completed
}

/*********************************************************************
* Function: void GOLRedrawRec(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* PreCondition: none
*
* Input: left,top,right,bottom - rectangle borders
*
* Output: none
*
* Side Effects: none
*
* Overview: marks objects with parts in the rectangle to be redrawn
*
* Note: none
*
*********************************************************************/
void GOLRedrawRec(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    OBJ_HEADER  *pCurrentObj;

    pCurrentObj = _pGolObjects;

    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 = (OBJ_HEADER *)pCurrentObj->pNxtObj;
    }   //end of while
}

/*********************************************************************
* Function: void GOLMsg(GOL_MSG *pMsg)
*
* PreCondition: none
*
* Input: pointer to the message
*
* Output: none
*
* Side Effects: none
*
* Overview: processes message for all objects in the liked list
*
* Note: none
*
*********************************************************************/
void GOLMsg(GOL_MSG *pMsg)
{
    OBJ_HEADER  *pCurrentObj;
    WORD        translatedMsg;

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

    pCurrentObj = _pGolObjects;

    while(pCurrentObj != NULL)
    {
        if(pCurrentObj->MsgObj)
```

```
        {
            translatedMsg = pCurrentObj->MsgObj(pCurrentObj, pMsg);

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

        pCurrentObj = (OBJ_HEADER *)pCurrentObj->pNxtObj;
    }
}

/*********************************************************************
* Variables for rounded panel drawing. Used by GOLRndPanelDraw and GOLRndPanelDrawTsk
*********************************************************************/
SHORT   _rpnlX1,        // Center x position of upper left corner
_rpnlY1,                // Center y position of upper left corner
_rpnlX2,                // Center x position of lower right corner
_rpnlY2,                // Center y position of lower right corner
_rpnlR;                 // radius
WORD    _rpnlFaceColor, // face color
_rpnlEmbossLtColor,     // emboss light color
_rpnlEmbossDkColor,     // emboss dark color
_rpnlEmbossSize;        // emboss size
void    *_pRpnlBitmap = NULL;

/*********************************************************************
* Function: WORD GOLPanelDrawTsk(void)
*
* PreCondition: parameters must be set with
*               GOLRndPanelDraw(x,y,radius,width,height,faceClr,embossLtClr,
*                               embossDkClr,pBitmap,embossSize)
*
* Input: None
*
* Output: Output: non-zero if drawing is completed
*
* Overview: draws a rounded panel on screen. Must be called repeatedly. Drawing is done
*           when it returns non-zero.
*
* Note: none
*
*********************************************************************/
WORD GOLPanelDrawTsk(void)
{

        #ifndef USE_NONBLOCKING_CONFIG

    WORD    counter;


    // check if we need to draw the panels and emboss sides
    if  (
            (_pRpnlBitmap == NULL) ||
            (
                ((_rpnlX2 - _rpnlX1 + (_rpnlR<<1)) > GetImageWidth((void *)_pRpnlBitmap))
&&
                ((_rpnlY2 - _rpnlY1 + (_rpnlR<<1)) > GetImageHeight((void *)_pRpnlBitmap))
&&
                (_pRpnlBitmap != NULL)
            )
        )
    {
        if(_rpnlR)
        {

            // draw upper left portion of the embossed area
            SetColor(_rpnlEmbossLtColor);
            Arc(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize, _rpnlR, 0xE1);
```

```
            // draw lower right portion of the embossed area
            SetColor(_rpnlEmbossDkColor);
            Arc(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize, _rpnlR, 0x1E);
        }
        else
        {

            // object is rectangular panel draw the embossed areas
            counter = 1;
            SetColor(_rpnlEmbossLtColor);
            while(counter < _rpnlEmbossSize)
            {
                Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX2 - counter, _rpnlY1 +
counter);    // draw top
                Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX1 + counter, _rpnlY2 -
counter);    // draw left
                counter++;
            }

            counter = 1;
            SetColor(_rpnlEmbossDkColor);
            while(counter < _rpnlEmbossSize)
            {
                Bar(_rpnlX1 + counter, _rpnlY2 - counter, _rpnlX2 - counter, _rpnlY2 -
counter);    // draw bottom
                Bar(_rpnlX2 - counter, _rpnlY1 + counter, _rpnlX2 - counter, _rpnlY2 -
counter);    // draw right
                counter++;
            }
        }

        // draw the face color
        SetColor(_rpnlFaceColor);
        if(_rpnlR)
            FillBevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize);
        else
            Bar(_rpnlX1 + _rpnlEmbossSize, _rpnlY1 + _rpnlEmbossSize, _rpnlX2 -
_rpnlEmbossSize, _rpnlY2 - _rpnlEmbossSize);

                #if (COLOR_DEPTH == 1)
        if(_rpnlFaceColor == _rpnlEmbossDkColor)
        {
            SetColor(_rpnlEmbossLtColor);
            if(_rpnlR)
                Bevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - (_rpnlEmbossSize - 1));
            else
                Bevel
                (
                    _rpnlX1 + (_rpnlEmbossSize - 1),
                    _rpnlY1 + (_rpnlEmbossSize - 1),
                    _rpnlX2 - (_rpnlEmbossSize - 1),
                    _rpnlY2 - (_rpnlEmbossSize - 1),
                    0
                );
        }

                #endif
    } // end of check if we need to draw the panels and emboss sides

    // draw bitmap
    if(_pRpnlBitmap != NULL)
    {
        PutImage
        (
            (((_rpnlX2 + _rpnlX1) - (GetImageWidth((void *)_pRpnlBitmap))) >> 1) + 1,
            (((_rpnlY2 + _rpnlY1) - (GetImageHeight((void *)_pRpnlBitmap))) >> 1) + 1,
            _pRpnlBitmap,
            IMAGE_NORMAL
        );
    }
```

```
    // check if we need to draw the frame
    if
    (
        (_pRpnlBitmap == NULL) ||
        (
            ((_rpnlX2 - _rpnlX1 + _rpnlR) >= GetImageWidth((void *)_pRpnlBitmap)) &&
            ((_rpnlY2 - _rpnlY1 + _rpnlR) >= GetImageHeight((void *)_pRpnlBitmap))
        )
    )
    {

        // draw the outline
        SetColor(_rpnlEmbossDkColor);
        Bevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR);
    }

    return (1);

        #else

    typedef enum
    {
        BEGIN,
        ARC1,
        DRAW_EMBOSS1,
        DRAW_EMBOSS2,
        DRAW_EMBOSS3,
        DRAW_EMBOSS4,
        DRAW_FACECOLOR,
                #if (COLOR_DEPTH == 1)
        DRAW_INNERFRAME,
                #endif
        DRAW_FRAME,
        DRAW_IMAGE,
    } ROUND_PANEL_DRAW_STATES;

    static ROUND_PANEL_DRAW_STATES state = BEGIN;
    static WORD counter;

    while(1)
    {
        if(IsDeviceBusy())
            return (0);
        switch(state)
        {
            case BEGIN:

                // check if we need to draw the face and emboss
                if (_pRpnlBitmap != NULL)
                {
                    if  (
                        ((_rpnlX2 - _rpnlX1 + (_rpnlR<<1)) <= GetImageWidth((void
*)_pRpnlBitmap)) &&
                        ((_rpnlY2 - _rpnlY1 + (_rpnlR<<1)) <= GetImageHeight((void
*)_pRpnlBitmap))
                        )
                    {
                        state = DRAW_IMAGE;
                        goto rnd_panel_draw_image;
                    }
                }

                if(_rpnlR)
                {

                    // draw upper left portion of the embossed area
                    SetColor(_rpnlEmbossLtColor);
                    if(!Arc(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize,
_rpnlR, 0xE1))
                        return (0);
                    state = ARC1;
                }
```

```
                else
                {
                    state = DRAW_EMBOSS1;
                    counter = 1;
                    goto rnd_panel_draw_emboss;
                }

            case ARC1:

                // draw upper left portion of the embossed area
                SetColor(_rpnlEmbossDkColor);
                if(!Arc(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize,
_rpnlR, 0x1E))
                    return (0);
                state = DRAW_FACECOLOR;
                goto rnd_panel_draw_facecolor;

            // now draw the upper portion of the embossed area
            case DRAW_EMBOSS1:
                rnd_panel_draw_emboss : SetColor(_rpnlEmbossLtColor);
                while(counter < _rpnlEmbossSize)
                {

                    // draw top
                    if(!Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX2 - counter,
_rpnlY1 + counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                counter = 1;
                state = DRAW_EMBOSS2;
                break;

            case DRAW_EMBOSS2:
                while(counter < _rpnlEmbossSize)
                {

                    // draw left
                    if(!Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX1 + counter,
_rpnlY2 - counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                counter = 1;
                state = DRAW_EMBOSS3;
                break;

            // now draw the lower portion of the embossed area
            case DRAW_EMBOSS3:
                SetColor(_rpnlEmbossDkColor);
                while(counter < _rpnlEmbossSize)
                {

                    // draw bottom
                    if(!Bar(_rpnlX1 + counter, _rpnlY2 - counter, _rpnlX2 - counter,
_rpnlY2 - counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                counter = 1;
```

```
                state = DRAW_EMBOSS4;
                break;

            case DRAW_EMBOSS4:
                while(counter < _rpnlEmbossSize)
                {

                    // draw right
                    if(!Bar(_rpnlX2 - counter, _rpnlY1 + counter, _rpnlX2 - counter,
_rpnlY2 - counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                state = DRAW_FACECOLOR;
                break;

            // draw the face color
            case DRAW_FACECOLOR:
                rnd_panel_draw_facecolor : SetColor(_rpnlFaceColor);
                if(_rpnlR)
                {
                    #ifdef USE_GRADIENT
                    if(_gradientScheme.gradientType != GRAD_NONE)
                    {
                       BevelGradient(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR -
_rpnlEmbossSize,
                                      _gradientScheme.gradientStartColor,_gradientScheme.grad
ientEndColor,
                                     _gradientScheme.gradientLength,_gradientScheme.gradientT
ype);
                    }
                   else
                   #endif



                    if(!FillBevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR -
_rpnlEmbossSize))
                        return (0);
                }
                else
                {
                    if
                    (
                        !Bar
                            (
                                _rpnlX1 + _rpnlEmbossSize,
                                _rpnlY1 + _rpnlEmbossSize,
                                _rpnlX2 - _rpnlEmbossSize,
                                _rpnlY2 - _rpnlEmbossSize
                            )
                    )
                    {
                        return (0);
                    }
                }

                state = DRAW_IMAGE;
                break;

            case DRAW_IMAGE:
                rnd_panel_draw_image :
                if(_pRpnlBitmap != NULL)
                {
                    if
                    (
                        !PutImage
                            (
```

```
                                ((_rpnlX2 + _rpnlX1 - GetImageWidth((void *)_pRpnlBitmap))
>> 1) + 1,
                                ((_rpnlY2 + _rpnlY1 - GetImageHeight((void *)_pRpnlBitmap))
>> 1) + 1,
                                _pRpnlBitmap,
                                IMAGE_NORMAL
                            )
                        )
                        {
                            return (0);
                        }
                    }

                            #if (COLOR_DEPTH == 1)
                    state = DRAW_INNERFRAME;
                    break;
                            #else
                    state = DRAW_FRAME;
                            #endif
                    break;

                            #if (COLOR_DEPTH == 1)

                case DRAW_INNERFRAME:
                    if(_rpnlFaceColor == _rpnlEmbossDkColor)
                    {
                        SetColor(_rpnlEmbossLtColor);
                        if(_rpnlR)
                        {
                            if(!Bevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR -
(_rpnlEmbossSize - 1)))
                            {
                                return (0);
                            }
                        }
                        else
                        {
                            if(!Bevel(  _rpnlX1 + (_rpnlEmbossSize - 1),
                                        _rpnlY1 + (_rpnlEmbossSize - 1),
                                        _rpnlX2 - (_rpnlEmbossSize - 1),
                                        _rpnlY2 - (_rpnlEmbossSize - 1),
                                        0 ))
                            {
                                return (0);
                            }
                        }
                    }

                    state = DRAW_FRAME;
                    break;
                            #endif

                case DRAW_FRAME:

                    // check if we need to draw the frame
                    if
                    (
                        (_pRpnlBitmap == NULL) ||
                        (
                            ((_rpnlX2 - _rpnlX1 + _rpnlR) >= GetImageWidth((void
*)_pRpnlBitmap)) &&
                            ((_rpnlY2 - _rpnlY1 + _rpnlR) >= GetImageHeight((void
*)_pRpnlBitmap))
                        )
                    )
                    {

                        // draw the outline frame
                                #if (COLOR_DEPTH == 1)
                        // When in 1BPP mode, the outline should always be the light color
                        // Ideally WHITE.
                        if (_rpnlEmbossLtColor > _rpnlEmbossDkColor)
```

```
                        SetColor(_rpnlEmbossLtColor);
                    else
                        SetColor(_rpnlEmbossDkColor);
                            #else
                    SetColor(_rpnlEmbossDkColor);
                            #endif
                    if(!Bevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR))
                    {
                        return (0);
                    }
                }

                state = BEGIN;
                return (1);
        }   // end of switch
    }       // end of while
        #endif //#ifndef USE_NONBLOCKING_CONFIG
}


/*****************************************************************
* Function: WORD GOLTwoTonePanelDrawTsk(void)
*
* PreCondition: parameters must be set with
*               GOLRndPanelDraw(x,y,radius,width,height,faceClr,embossLtClr,
*                               embossDkClr,pBitmap,embossSize)
*
* Input: None
*
* Output: Output: non-zero if drawing is completed
*
* Overview: draws a rounded panel on screen. Must be called repeatedly. Drawing is done
*           when it returns non-zero.
*
* Note: none
*
*****************************************************************/
WORD GOLTwoTonePanelDrawTsk(void)
{
// In this panel draw task the emboss light and dark colors are used as
// the panel face colors and the panel face color is used as an outline color

        #ifndef USE_NONBLOCKING_CONFIG

    WORD    counter;

    SetColor(_rpnlFaceColor);
    if(_rpnlR)
    {
        // draw the outline
        Arc(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize, _rpnlR, 0xFF);
    }
    else
    {
        // object is rectangular panel draw the outline embossed areas
        counter = 1;
        while(counter < _rpnlEmbossSize)
        {
            Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX2 - counter, _rpnlY1 +
counter);    // draw top
            Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX1 + counter, _rpnlY2 -
counter);    // draw left
            Bar(_rpnlX1 + counter, _rpnlY2 - counter, _rpnlX2 - counter, _rpnlY2 -
counter);    // draw bottom
            Bar(_rpnlX2 - counter, _rpnlY1 + counter, _rpnlX2 - counter, _rpnlY2 -
counter);    // draw right
            counter++;
        }
    }

    // draw the top half of the face
    SetColor(_rpnlEmbossLtColor);
```

```
    if(_rpnlR)
    {
        SetBevelDrawType(DRAWTOPBEVEL);
        FillBevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize);
    }
    else
    {
        Bar(_rpnlX1 + _rpnlEmbossSize, _rpnlY1 + _rpnlEmbossSize,
            _rpnlX2 - _rpnlEmbossSize, (_rpnlY1 + ((_rpnlY2 - _rpnlY1) >> 1)));
    }

    // draw the bottom half of the face
    SetColor(_rpnlEmbossDkColor);
    if(_rpnlR)
    {
        SetBevelDrawType(DRAWBOTTOMBEVEL);
        FillBevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize);
    }
    else
    {
        Bar(_rpnlX1 + _rpnlEmbossSize, (_rpnlY1 + ((_rpnlY2 - _rpnlY1) >> 1)) + 1,
            _rpnlX2 - _rpnlEmbossSize, _rpnlY2 - _rpnlEmbossSize);
    }

    SetBevelDrawType(DRAWFULLBEVEL);


            #if (COLOR_DEPTH == 1)
    if(_rpnlFaceColor == _rpnlEmbossDkColor)
    {
        SetColor(_rpnlEmbossLtColor);
        if(_rpnlR)
            Bevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - (_rpnlEmbossSize - 1));
        else
            Bevel
            (
                _rpnlX1 + (_rpnlEmbossSize - 1),
                _rpnlY1 + (_rpnlEmbossSize - 1),
                _rpnlX2 - (_rpnlEmbossSize - 1),
                _rpnlY2 - (_rpnlEmbossSize - 1),
                0
            );
    }

            #endif

    // draw bitmap
    if(_pRpnlBitmap != NULL)
    {
        PutImage
        (
            (((_rpnlX2 + _rpnlX1) - (GetImageWidth((void *)_pRpnlBitmap))) >> 1) + 1,
            (((_rpnlY2 + _rpnlY1) - (GetImageHeight((void *)_pRpnlBitmap))) >> 1) + 1,
            _pRpnlBitmap,
            IMAGE_NORMAL
        );
    }

    return (1);

        #else

    typedef enum
    {
        BEGIN,
        DRAW_EMBOSS1,
        DRAW_EMBOSS2,
        DRAW_EMBOSS3,
        DRAW_EMBOSS4,
        DRAW_FACECOLOR1,
        DRAW_FACECOLOR2,
                #if (COLOR_DEPTH == 1)
```

```
        DRAW_INNERFRAME,
                #endif
        DRAW_IMAGE,
    } ROUND_PANEL_DRAW_STATES;

    static ROUND_PANEL_DRAW_STATES state = BEGIN;
    static WORD counter;

    while(1)
    {
        if(IsDeviceBusy())
            return (0);
        switch(state)
        {
            case BEGIN:
                if(_rpnlR)
                {

                    // draw the outline
                    SetColor(_rpnlFaceColor);
                    if(!Arc(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR - _rpnlEmbossSize,
_rpnlR, 0xFF))
                        return (0);
                    state = DRAW_FACECOLOR1;
                }
                else
                {
                    state = DRAW_EMBOSS1;
                    counter = 1;
                    goto rnd_panel_draw_emboss;
                }

            // now draw the upper portion of the embossed area
            case DRAW_EMBOSS1:
                rnd_panel_draw_emboss : SetColor(_rpnlFaceColor);
                while(counter < _rpnlEmbossSize)
                {

                    // draw top
                    if(!Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX2 - counter,
_rpnlY1 + counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                counter = 1;
                state = DRAW_EMBOSS2;
                break;

            case DRAW_EMBOSS2:
                while(counter < _rpnlEmbossSize)
                {

                    // draw left
                    if(!Bar(_rpnlX1 + counter, _rpnlY1 + counter, _rpnlX1 + counter,
_rpnlY2 - counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                counter = 1;
                state = DRAW_EMBOSS3;
                break;

            // now draw the lower portion of the embossed area
            case DRAW_EMBOSS3:
```

```
                //SetColor(_rpnlEmbossDkColor);
                while(counter < _rpnlEmbossSize)
                {

                    // draw bottom
                    if(!Bar(_rpnlX1 + counter, _rpnlY2 - counter, _rpnlX2 - counter,
_rpnlY2 - counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                counter = 1;
                state = DRAW_EMBOSS4;
                break;

            case DRAW_EMBOSS4:
                while(counter < _rpnlEmbossSize)
                {

                    // draw right
                    if(!Bar(_rpnlX2 - counter, _rpnlY1 + counter, _rpnlX2 - counter,
_rpnlY2 - counter))
                    {
                        return (0);
                    }

                    counter++;
                }

                state = DRAW_FACECOLOR1;
                break;

            // draw the top half of the face
            case DRAW_FACECOLOR1:
                SetColor(_rpnlEmbossLtColor);
                if(_rpnlR)
                {
                    SetBevelDrawType(DRAWTOPBEVEL);
                    if(!FillBevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR -
_rpnlEmbossSize))
                        return (0);
                }
                else
                {
                    if
                    (
                        !Bar
                            (
                                _rpnlX1 + _rpnlEmbossSize,
                                _rpnlY1 + _rpnlEmbossSize,
                                _rpnlX2 - _rpnlEmbossSize,
                                (_rpnlY1 + ((_rpnlY2 - _rpnlY1) >>
1))
                            )
                    )
                    {
                        return (0);
                    }
                }

                state = DRAW_FACECOLOR2;
                break;

            // draw the bottom half of the face
            case DRAW_FACECOLOR2:
                SetColor(_rpnlEmbossDkColor);
                if(_rpnlR)
                {
                    SetBevelDrawType(DRAWBOTTOMBEVEL);
```

```
                    if(!FillBevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR -
_rpnlEmbossSize))
                        return (0);
                }
                else
                {
                    if
                    (
                        !Bar
                            (
                                _rpnlX1 + _rpnlEmbossSize,
                                (_rpnlY1 + ((_rpnlY2 - _rpnlY1) >> 1)) + 1,
                                _rpnlX2 - _rpnlEmbossSize,
                                _rpnlY2 - _rpnlEmbossSize
                            )
                    )
                    {
                        return (0);
                    }
                }
                SetBevelDrawType(DRAWFULLBEVEL);
                state = DRAW_IMAGE;
                break;

            case DRAW_IMAGE:
                if(_pRpnlBitmap != NULL)
                {
                    if
                    (
                        !PutImage
                            (
                                ((_rpnlX2 + _rpnlX1 - GetImageWidth((void *)_pRpnlBitmap))
>> 1) + 1,
                                ((_rpnlY2 + _rpnlY1 - GetImageHeight((void *)_pRpnlBitmap))
>> 1) + 1,
                                _pRpnlBitmap,
                                IMAGE_NORMAL
                            )
                    )
                    {
                        return (0);
                    }
                }

                        #if (COLOR_DEPTH == 1)
                state = DRAW_INNERFRAME;
                break;
                        #else
                state = BEGIN;
                return (1);
                        #endif
                break;

                        #if (COLOR_DEPTH == 1)

            case DRAW_INNERFRAME:
                if(_rpnlFaceColor == _rpnlEmbossDkColor)
                {
                    SetColor(_rpnlEmbossLtColor);
                    if(_rpnlR)
                    {
                        if(!Bevel(_rpnlX1, _rpnlY1, _rpnlX2, _rpnlY2, _rpnlR -
(_rpnlEmbossSize - 1)))
                        {
                            return (0);
                        }
                    }
                    else
                    {
                        if(!Bevel(  _rpnlX1 + (_rpnlEmbossSize - 1),
                                    _rpnlY1 + (_rpnlEmbossSize - 1),
                                    _rpnlX2 - (_rpnlEmbossSize - 1),
```

```
                                        _rpnlY2 - (_rpnlEmbossSize - 1),
                                        0 ))
                                {
                                    return (0);
                                }
                            }
                        }

                        state = BEGIN;
                        return (1);
                                #endif

                }   // end of switch
            }       // end of while
                #endif //#ifndef USE_NONBLOCKING_CONFIG
        }

        #endif // USE_GOL
```

14.1.2 GOL.h

Enumerations

Name	Description
GOL_OBJ_TYPE (see page 51)	This structure defines the Object types used in the library.
INPUT_DEVICE_EVENT (see page 319)	This structure defines the types of GOL (see page 48) message events used in the library.
INPUT_DEVICE_TYPE (see page 319)	This structure defines the types of input devices used in the library.
TRANS_MSG (see page 317)	This structure defines the list of translated messages for GOL (see page 48) Objects used in the library.

Functions

	Name	Description
	GOLAddObject (see page 292)	This function adds an object to the tail of the active list pointed to by _pGolObjects (see page 334). The new list tail is set to point to NULL.
	GOLCanBeFocused (see page 307)	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.
	GOLCreateScheme (see page 328)	This function creates a new style scheme object and initializes the parameters to default values.
	GOLDeleteObject (see page 301)	deletes an object to the linked list objects for the current screen.
	GOLDeleteObjectByID (see page 301)	Deletes an object in the current active linked list of objects using the ID parameter of the object.
	GOLDraw (see page 296)	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 (see page 297)() function is called by GOLDraw() when drawing of objects in the active list is completed.
	GOLDrawCallback (see page 297)	GOLDrawCallback() function MUST BE implemented by the user. This is called inside the GOLDraw (see page 296)() 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 (see page 48) if this function returns a zero. To pass drawing control to GOL (see page 48) this function must return a non-zero value. If GOL (see page 48) messaging is not using the active link list, it is safe to modify the list here.
	GOLFindObject (see page 293)	This function finds an object in the active list pointed to by _pGolObjects (see page 334) using the given object ID.

	GOLFree (see page 298)	This function frees all the memory used by objects in the active list and initializes _pGolObjects (see page 334) pointer to NULL to start a new empty list. This function must be called only inside the GOLDrawCallback (see page 297)()function when using GOLDraw (see page 296)() and GOLMsg (see page 314)() functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.
	GOLGetFocusNext (see page 307)	This function returns the pointer to the next object in the active linked list which is able to receive keyboard input.
	GOLGetFocusPrev (see page 308)	This function returns the pointer to the previous object in the active linked list which is able to receive keyboard input.
	GOLInit (see page 306)	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 (see page 48) functions can be used. It is not necessary to call GraphInit() function if this function is used.
	GOLMsg (see page 314)	This function receives a GOL (see page 48) 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 (see page 314)() 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 (see page 48) Messages section for deatils. This function should be called when GOL (see page 48) drawing is completed. It can be done... more (see page 314)
	GOLMsgCallback (see page 314)	The user MUST implement this function. GOLMsg (see page 314)() 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 (see page 48) will not perform any action.
	GOLPanelDrawTsk (see page 310)	This function draws a panel on the screen with parameters set by GOLPanelDraw (see page 308)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.
	GOLRedrawRec (see page 295)	This function marks all objects in the active list intersected by the given rectangular area to be redrawn.
	GOLSetFocus (see page 304)	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 (see page 287) state for the object was in focus previously, set FOCUSED (see page 287) state for the required object and marks the objects to be redrawn.
	GOLTwoTonePanelDrawTsk (see page 310)	This function draws a two tone panel on the screen with parameters set by GOLPanelDraw (see page 308)() macro. This function must be called repeatedly (depending on the return value) for a successful rendering of the panel.

Macros

Name	Description
ClrState (see page 289)	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.
DISABLED (see page 287)	Disabled state bit.
DRAW (see page 287)	Object redraw state bit. The whole Object must be redrawn.
DRAW_FOCUS (see page 288)	Focus redraw state bit. The focus rectangle must be redrawn.
DRAW_UPDATE (see page 288)	Partial Object redraw state bit. A part or parts of of the Object must be redrawn to show updated state.
FOCUSED (see page 287)	Focus state bit.
GetObjID (see page 299)	This macro returns the object ID.
GetObjNext (see page 300)	This macro returns the next object after the specified object.
GetObjType (see page 299)	This macro returns the object type.

GetState (see page 288)	This macro retrieves the current value of the state bits of an object. It is possible to get several state bits.
GOL_EMBOSS_SIZE (see page 332)	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 (see page 928) file then the default value is used.
GOLDrawComplete (see page 296)	This macro resets the drawing states of the object (6 MSBits of the object's state).
GOLGetFocus (see page 306)	This macro returns the pointer to the current object receiving keyboard input.
GOLGetList (see page 303)	This macro gets the current active list.
GOLGetScheme (see page 330)	This macro gets the GOL (see page 48) scheme used by the given object.
GOLGetSchemeDefault (see page 330)	This macro returns the default GOL (see page 48) scheme pointer.
GOLNewList (see page 302)	This macro starts a new linked list of objects and resets the keyboard focus to none. This macro assigns the current active list _pGolObjects (see page 334) and current receiving keyboard input _pObjectFocused (see page 335) 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 (see page 298)() function.
GOLPanelDraw (see page 308)	This macro sets the parameters to draw a panel. Panel is not an object. It will not be added to the active list. Use GOLPanelDrawTsk (see page 310)() to display the panel on the screen. The following relationships of the parameters determines the general shape of the button: 1. Panel width is determined by right - left. 2. Panel height is determined by top - bottom. 3. Panel radius - specifies if the panel will have a rounded edge. If zero then the panel will have sharp (cornered) edge. 4. If 2*radius = height = width, the panel is circular.
GOLRedraw (see page 294)	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.
GOLSetList (see page 303)	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 (see page 335) pointer to NULL. If the new active list has an object's state set to focus, the _pObjectFocused (see page 335) 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.
GOLSetScheme (see page 329)	This macro sets the GOL (see page 48) scheme to be used for the object.
HIDE (see page 287)	Object hide state bit. Object will be hidden from the screen by drawing over it the common background color.
IsObjUpdated (see page 305)	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.
SetState (see page 289)	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.

Structures

Name	Description
GOL_MSG (see page 316)	This structure defines the GOL (see page 48) message used in the library. • The types must be one of the INPUT_DEVICE_TYPE (see page 319): • TYPE_UNKNOWN • TYPE_KEYBOARD • TYPE_TOUCHSCREEN • TYPE_MOUSE • uiEvent must be one of the INPUT_DEVICE_EVENT (see page 319). • 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
GOL_SCHEME (see page 331)	GOL (see page 48) scheme defines the style scheme to be used by an object.
OBJ_HEADER (see page 52)	This structure defines the first nine fields of the Objects structure. This allows generic operations on library Objects.

Types

Name	Description
DRAW_FUNC (see page 52)	object draw function pointer typedef
FREE_FUNC (see page 53)	object free function pointer typedef
MSG_DEFAULT_FUNC (see page 53)	object default message function pointer typedef
MSG_FUNC (see page 53)	object message function pointer typedef

Variables

Name	Description
_pDefaultGolScheme (see page 334)	Pointer to the GOL (see page 48) default scheme (GOL_SCHEME (see page 331)). This scheme is created in GOLInit (see page 306)() function. GOL (see page 48) scheme defines the style scheme to be used by an object. Use GOLGetSchemeDefault (see page 330)() to get this pointer.
_pGolObjects (see page 334)	Pointer to the current linked list of objects displayed and receiving messages. GOLDraw (see page 296)() and GOLMsg (see page 314)() process objects referenced by this pointer.

_pObjectFocused (see page 335)	Pointer to the object receiving keyboad input. This pointer is used or modified by the following APIs: • GOLSetFocus (see page 304)() • GOLGetFocus (see page 306)() • GOLGetFocusNext (see page 307)() • GOLGetFocusPrev (see page 308)() • GOLCanBeFocused (see page 307)()
FONTDEFAULT (see page 332)	Default GOL (see page 48) font.

Description

This is file GOL.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *****************************************************************************
 * FileName:        GOL.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2010 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 06/29/09     Added BTN_MSG_STILLPRESSED message for
 *              button object and EVENT_STILLPRESS for
 *              INPUT_DEVICE_EVENT list.
 * 10/04/10     FONTDEFAULT can now be modified by the user without
 *              modifying the library files. To replace default font
 *              (GOLFontDefault found in the library supplied GOLFontDefault.c
 *              file) user must add this line in the GraphicsConfig.h file:
 *                  #define FONTDEFAULT userFont where "userFont" is the
 *              user supplied font.
 * 02/24/11     Replace color data type to GFX_COLOR
 *****************************************************************************/
#ifndef _GOL_H
    #define _GOL_H

/*********************************************************************
```

```
 * Section: Includes
 *****************************************************************/
#include "Graphics/Primitive.h"
#include "GenericTypeDefs.h"
#include "Graphics/gfxcolors.h"
#include "GraphicsConfig.h"

/*****************************************************************
* Overview: GOL scheme defines the style scheme to be used by an object.
*
*****************************************************************/
typedef struct
{
    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.

    #ifdef USE_ALPHABLEND
    BYTE    AlphaValue;             // Alpha value used for alpha blending.
    #endif

    #ifdef USE_GRADIENT
    GFX_GRADIENT_STYLE gradientScheme;   //Gradient Scheme for widgets
    #endif
} GOL_SCHEME;

#ifdef USE_GRADIENT
extern GFX_GRADIENT_STYLE _gradientScheme;
#endif


/*****************************************************************
* 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.
*
*****************************************************************/
extern GOL_SCHEME   *_pDefaultGolScheme;

/*****************************************************************
* Overview: This structure defines the Object types used in the library.
*
*****************************************************************/
typedef enum
{
    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.
} GOL_OBJ_TYPE;

/*********************************************************************
* 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
*
*********************************************************************/
typedef struct
{
    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.
} GOL_MSG;

/*********************************************************************
* Overview: This structure defines the types of input devices
*           used in the library.
*
*********************************************************************/
typedef enum
{
    TYPE_UNKNOWN    = 0,            // Unknown device.
    TYPE_KEYBOARD,                  // Keyboard.
    TYPE_TOUCHSCREEN,               // Touchscreen.
    TYPE_MOUSE,                     // Mouse.
    TYPE_TIMER,                     // Timer.
    TYPE_SYSTEM                     // System Messages.
} INPUT_DEVICE_TYPE;

/*********************************************************************
* Overview: This structure defines the types of GOL message events
*           used in the library.
*
*********************************************************************/
typedef enum
{
    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.
} INPUT_DEVICE_EVENT;

/*********************************************************************
* Overview: This structure defines the list of translated messages for
*           GOL Objects used in the library.
*
*********************************************************************/
typedef enum
{
    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.
} TRANS_MSG;

typedef WORD(*DRAW_FUNC)(void *);                           // object draw function pointer
typedef
typedef void(*FREE_FUNC)(void *);                           // object free function pointer
typedef
typedef WORD(*MSG_FUNC)(void *, GOL_MSG *);                 // object message function
pointer typedef
typedef void(*MSG_DEFAULT_FUNC)(WORD, void *, GOL_MSG *);   // object default message
function pointer typedef

/*********************************************************************
* Overview: This structure defines the first nine fields of the
*           Objects structure. This allows generic operations on
*           library Objects.
*
*********************************************************************/
typedef struct
```

```
{
    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
} OBJ_HEADER;

/*********************************************************************
* Overview: The following are the common Objcet States.
*
*********************************************************************/

// Focus state bit.
    #define FOCUSED 0x0001

// Disabled state bit.
    #define DISABLED    0x0002

// Object hide state bit. Object will be hidden from the
// screen by drawing over it the common background color.
    #define HIDE    0x8000

// Object redraw state bit. The whole Object must be redrawn.
    #define DRAW    0x7C00

// Focus redraw state bit. The focus rectangle must be redrawn.
    #define DRAW_FOCUS  0x2000

// Partial Object redraw state bit. A part or parts of
// of the Object must be redrawn to show updated state.
    #define DRAW_UPDATE 0x3C00

/*********************************************************************
* Overview: The following are the global variables of GOL.
*
*********************************************************************/

/*********************************************************************
* Overview: Pointer to the current linked list of objects displayed
*           and receiving messages. GOLDraw() and GOLMsg() process
*           objects referenced by this pointer.
*
*********************************************************************/
extern OBJ_HEADER   *_pGolObjects;

/*********************************************************************
* Overview: Pointer to the object receiving keyboad input. This pointer
*           is used or modified by the following APIs:
*           - GOLSetFocus()
*           - GOLGetFocus()
*           - GOLGetFocusNext()
*           - GOLGetFocusPrev()
*           - GOLCanBeFocused()
*
*********************************************************************/
extern OBJ_HEADER   *_pObjectFocused;
```

```
// Line type used for focus mark.
    #define FOCUS_LINE  2

/*********************************************************************
* 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.
*
*********************************************************************/
    #ifndef GOL_EMBOSS_SIZE
        #define GOL_EMBOSS_SIZE 3
    #endif

/*********************************************************************
* Overview: The default font GOLFontDefault is declared in
*           GOLFontDefault.c file included in the Graphics Library.
*           To replace this font, add this declaration in the
*           GraphicsConfig.h:
*               #define FONTDEFAULT yourFont
*           Then in the project the "yourFont" must be added.
*           The Graphics Library will then use the font that the user
*           supplied.
*********************************************************************/
    #ifndef FONTDEFAULT
        // Use the default GOL font in GOLFontDefault.c
        #define FONTDEFAULT GOLFontDefault
    #endif

    // Default GOL font.
    extern const FONT_FLASH FONTDEFAULT;

/*********************************************************************
* Overview: The following are the style scheme default settings.
*
*********************************************************************/
#ifndef GFX_SCHEMEDEFAULT
#define GFX_SCHEMEDEFAULT GOLSchemeDefault
#endif

    extern const GOL_SCHEME GFX_SCHEMEDEFAULT;


/*********************************************************************
* Function: WORD GOLCanBeFocused(OBJ_HEADER* object)
*
* 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.
*
* PreCondition: none
*
* Input: object - Pointer to the object of interest.
*
* Output: This returns a non-zero if the object can be focused
*         and zero if not.
*
* Side Effects: none
*
*********************************************************************/
WORD        GOLCanBeFocused(OBJ_HEADER *object);

/*********************************************************************
* Function: OBJ_HEADER *GOLGetFocusNext()
*
* Overview: This function returns the pointer to the previous object
*           in the active linked list which is able to receive
*           keyboard input.
*
* PreCondition: none
```

```
 *
 * Input: none
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 ********************************************************************/
OBJ_HEADER  *GOLGetFocusPrev(void);

/********************************************************************
 * Function: OBJ_HEADER *GOLGetFocusNext()
 *
 * Overview: This function returns the pointer to the next object
 *           in the active linked list which is able to receive
 *           keyboard input.
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 ********************************************************************/
OBJ_HEADER  *GOLGetFocusNext(void);

/********************************************************************
 * Function: void GOLSetFocus(OBJ_HEADER* object)
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: object - Pointer to the object of interest.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
void        GOLSetFocus(OBJ_HEADER *object);

/********************************************************************
 * Macros:  GOLGetFocus()
 *
 * Overview: This macro returns the pointer to the current object
 *           receiving keyboard input.
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: Returns the pointer to the object receiving keyboard input.
 *         If there is no object in focus NULL is returned.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define GOLGetFocus()   _pObjectFocused
```

```
/*********************************************************************
* Macros: GetObjType(pObj)
*
* Overview: This macro returns the object type.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object of interest.
*
* Output: Returns the OBJ_TYPE of the object.
*
* Side Effects: none
*
*********************************************************************/
    #define GetObjType(pObj)    ((OBJ_HEADER *)pObj)->type

/*********************************************************************
* Macros: GetObjID(pObj)
*
* Overview: This macro returns the object ID.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object of interest.
*
* Output: Returns the ID of the object.
*
* Example:
*   <CODE>
*   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
*           }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GetObjID(pObj)  ((OBJ_HEADER *)pObj)->ID

/*********************************************************************
* Macros: GetObjNext(pObj)
*
* Overview: This macro returns the next object after the specified
*           object.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object of interest.
*
* Output: Returns the pointer of the next object.
*
* Example:
*   <CODE>
*   // 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
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GetObjNext(pObj)    ((OBJ_HEADER *)pObj)->pNxtObj

/*********************************************************************
* Macros: IsObjUpdated(pObj)
*
* 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.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object of interest.
*
* Output: Returns a nonzero value if the object needs
*         to be redrawn. Zero if not.
*
* Example:
*   <CODE>
*   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;
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define IsObjUpdated(pObj)  (((OBJ_HEADER *)pObj)->state & 0xfc00)

/*********************************************************************
* Macros: GOLDrawComplete(pObj)
*
* Overview: This macro resets the drawing states of the object
*           (6 MSBits of the object's state).
*
* PreCondition: none
*
```

```
* Input: pObj - Pointer to the object of interest.
*
* Output: none
*
* Example:
*   <CODE>
*   // 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;
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GOLDrawComplete(pObj)   ((OBJ_HEADER *)pObj)->state &= 0x03ff

/*********************************************************************
* Macros:  SetState(pObj, stateBits)
*
* 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.
*
* PreCondition: none
*
* Input: 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
*
* Output: none
*
* Example:
*   <CODE>
*   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;
*        }
*    }
*    </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define SetState(pObj, stateBits)   ((OBJ_HEADER *)pObj)->state |= (stateBits)

/*********************************************************************
* Macros:  ClrState(pObj, stateBits)
*
* 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.
*
* PreCondition: none
*
* Input: 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
*
* Output: none
*
* Example:
*   See example for SetState().
*
* Side Effects: none
*
*********************************************************************/
    #define ClrState(pObj, stateBits)   ((OBJ_HEADER *)pObj)->state &= (~(stateBits))

/*********************************************************************
* Macros:  GetState(pObj, stateBits)
*
* Overview: This macro retrieves the current value of
*           the state bits of an object. It is possible
*           to get several state bits.
*
* PreCondition: none
*
* Input: 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
*
* Output: Macro returns a non-zero if any bit in the
*         stateBits mask is set.
*
* Example:
*   <CODE>
*   #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);
*
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GetState(pObj, stateBits)   (((OBJ_HEADER *)pObj)->state & (stateBits))
```

```
/*********************************************************************
* Function: GOL_SCHEME *GOLCreateScheme()
*
* Overview: This function creates a new style scheme object
*           and initializes the parameters to default values.
*
* PreCondition: none
*
* Input: none
*
* Output: Pointer to the new GOL_SCHEME created.
*
* Example:
*   <CODE>
*   extern const char Font22[] __attribute__((aligned(2)));
*   extern const char Font16[] __attribute__((aligned(2)));
*
*   GOLSCHEME *pScheme1, *pScheme2;
*   pScheme1 = GOLCreateScheme();
*   pScheme2 = GOLCreateScheme();
*
*   pScheme1->pFont = (BYTE*)Font22;
*   pScheme2->pFont = (BYTE*)Font16;
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
GOL_SCHEME  *GOLCreateScheme(void);

/*********************************************************************
* Macros:  GOLSetScheme(pObj, pScheme)
*
* Overview: This macro sets the GOL scheme to be used for the object.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object of interest.
*        pScheme - Pointer to the style scheme to be used.
*
* Output: none
*
* Example:
*   <CODE>
*   extern FONT_FLASH Gentium12;
*   GOLSCHEME *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);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GOLSetScheme(pObj, pScheme) ((OBJ_HEADER *)pObj)->pGolScheme = pScheme

/*********************************************************************
* Macros:  GOLGetScheme(pObj)
*
* Overview: This macro gets the GOL scheme used by the given object.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object of interest.
*
* Output: Returns the style scheme used by the given object.
```

```
 *
 * Example:
 *   <CODE>
 *   GOLSCHEME *pScheme2;
 *   BUTTON *pButton;
 *
 *   // assume button is created and initialized
 *   // get the scheme assigned to pButton
 *   pScheme2 = GOLGetScheme(pButton);
 *   </CODE>
 *
 * Side Effects: none
 *
 ***************************************************************************/
    #define GOLGetScheme(pObj)  ((OBJ_HEADER *)pObj)->pGolScheme

/***************************************************************************
 * Macros:  GOLGetSchemeDefault()
 *
 * Overview: This macro returns the default GOL scheme pointer.
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: Returns the pointer to the default style scheme.
 *
 * Side Effects: none
 *
 ***************************************************************************/
    #define GOLGetSchemeDefault()   _pDefaultGolScheme

/***************************************************************************
 * Function: void GOLInit()
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: none
 *
 * 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.
 *
 ***************************************************************************/
void GOLInit();

/***************************************************************************
 * Function: void GOLAddObject(OBJ_HEADER* object)
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: pObj - Pointer to the object to be added on the current
 *               active list.
 *
 * Output: none
 *
```

```
* Example:
*   <CODE>
*   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
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
void    GOLAddObject(OBJ_HEADER *object);

/*********************************************************************
* Macros: void GOLNewList()
*
* 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 GOLFree() function.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Example:
*   <CODE>
*   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);
*   </CODE>
*
* Side Effects: This macro sets the focused object pointer
*               (_pObjectFocused) to NULL.
*
*********************************************************************/
    #define GOLNewList() \
    _pGolObjects = NULL; \
    _pObjectFocused = NULL

/*********************************************************************
* Macros: GOLGetList()
*
* Overview: This macro gets the current active list.
*
* PreCondition: none
*
* Input: none
*
```

```
* Output: Returns the pointer to the current active list.
*
* Example:
*   See GOLNewList() example.
*
* Side Effects: none
*
*********************************************************************/
    #define GOLGetList()    _pGolObjects

/*********************************************************************
* Macros: GOLSetList(objsList)
*
* 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.
*
* PreCondition: none
*
* Input: objsList - The pointer to the new active list.
*
* Output: none
*
* Example:
*   <CODE>
*   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);
*   </CODE>
*
* 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 GOLFree() function to free the memory
*               used by the objects before calling GOLSetList().
*
*********************************************************************/
    #define GOLSetList(objsList) \
    _pGolObjects = objsList;     \
    _pObjectFocused = NULL

/*********************************************************************
* Function: OBJ_HEADER* GOLFindObject(WORD ID)
*
* Overview: This function finds an object in the active list
*           pointed to by _pGolObjects using the given object ID.
*
* PreCondition: none
*
* Input: ID - User assigned value set during the creation of the object.
*
* Output: Pointer to the object with the given ID.
*
* Example:
*   <CODE>
```

```
*   void CopyObject(OBJ_HEADER *pSrcList, OBJ_HEADER *pDstList, WORD ID)
*   {
*       OBJ_HEADER *pTemp;
*
*       pTemp = GOLFindObject(ID);              // find the object
*       if (pTemp != NULL) {
*           GOLSetList(pDstList);               // destination as active list
*           GOLAddObject(pObj);                 // add object to active list
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
OBJ_HEADER * GOLFindObject(WORD ID);

/*********************************************************************
* Function: void GOLFree()
*
* 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.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Example:
*   <CODE>
*   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
*       GolFree();                          // pPage objects are deleted
*
*       GOLSetList(pTemp);                  // restore the active list
*   }
*   </CODE>
*
* Side Effects: All objects in the active list are deleted from memory.
*
*********************************************************************/
void    GOLFree(void);

/*********************************************************************
* Function: BOOL GOLDeleteObject(OBJ_HEADER * object)
*
* PreCondition: none
*
* Input: pointer to the object
*
* Output: none
*
* Side Effects: none
*
* Overview: deletes an object to the linked list objects for the current screen.
*
* Note: none
*
*********************************************************************/
BOOL    GOLDeleteObject(OBJ_HEADER *object);

/*********************************************************************
```

```
* Function: BOOL GOLDeleteObjectByID(WORD ID)
*
* PreCondition: none
*
* Input: ID of the object.
*
* Output: none
*
* Side Effects: none
*
* Overview: Deletes an object in the current active linked list of objects using the ID
parameter
*           of the object.
*
* Note: none
*
*********************************************************************/
BOOL    GOLDeleteObjectByID(WORD ID);

/*********************************************************************
* Function: WORD GOLDraw()
*
* 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() function is called by GOLDraw()
*           when drawing of objects in the active list is completed.
*
* PreCondition: none
*
* Input: none
*
* Output: Non-zero if the active link list drawing is completed.
*
* Example:
*   <CODE>
*   // 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
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD    GOLDraw(void);

/*********************************************************************
* Function: void GOLRedrawRec(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* Overview: This function marks all objects in the active
*           list intersected by the given rectangular area
*           to be redrawn.
*
* PreCondition: none
*
* Input: left - Defines the left most border of the rectangle area.
*        top - Defines the top most border of the rectangle area.
*        right - Defines the right most border of the rectangle area.
*        bottom - Defines the bottom most border of the rectangle area.
*
* Output: none
```

```
*
* Example:
*   <CODE>
*   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
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
void    GOLRedrawRec(SHORT left, SHORT top, SHORT right, SHORT bottom);

/*********************************************************************
* Macros: GOLRedraw(pObj)
*
* 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.
*
* PreCondition: none
*
* Input: pObj - Pointer to the object to be redrawn.
*
* Output: none
*
* Example:
*   <CODE>
*   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
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GOLRedraw(pObj) ((OBJ_HEADER *)pObj)->state |= 0x7c00;

/*********************************************************************
* Function: void GOLMsg(GOL_MSG *pMsg)
*
* 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 deatils.
*
*           This function should be called when GOL drawing
*           is completed. It can be done when GOLDraw() returns
*           non-zero value or inside GOLDrawCallback() function.
*
* PreCondition: none
*
* Input: pMsg - Pointer to the GOL message from user.
*
* Output: none
*
* Example:
*   <CODE>
*   // 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);
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
void            GOLMsg(GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg)
*
* 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.
*
* PreCondition: none
*
* Input: objMsg - Translated message for the object or the action ID response from the
object.
*        pObj - Pointer to the object that processed the message.
*        pMsg - Pointer to the GOL message from user.
*
* Output: 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.
*
* Example:
*   <CODE>
*   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;
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD            GOLMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD GOLDrawCallback()
*
* 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.
*
* PreCondition: none
*
* Input: none
*
* Output: Return a one if GOLDraw() will have drawing control
*         on the active list. Return a zero if user wants to
*         keep the drawing control.
*
* Example:
*   <CODE>
*   #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;
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD            GOLDrawCallback(void);

/*********************************************************************
* Variables for rounded panel drawing. Used by GOLRndPanelDraw and GOLRndPanelDrawTsk
*********************************************************************/
extern SHORT    _rpnlX1;            // Panel center x position of upper left corner
extern SHORT    _rpnlY1;            // Panel center y position of upper left corner
extern SHORT    _rpnlX2;            // Panel center x position of lower right corner
extern SHORT    _rpnlY2;            // Panel center y position of lower right corner
extern SHORT    _rpnlR;             // radius (defines the rounded corner size)
extern WORD     _rpnlFaceColor;     // Panel face color
extern WORD     _rpnlEmbossLtColor; // Panel emboss light color
extern WORD     _rpnlEmbossDkColor; // Panel emboss dark color
extern WORD     _rpnlEmbossSize;    // Size of panel emboss (Usually uses GOL_EMBOSS_SIZE
but maybe different

// for some objects)emboss size
extern void     *_pRpnlBitmap;      // Bitmap used in the panel

/*********************************************************************
* Macros: GOLPanelDraw( left, top, right, bottom,
*                       radius,
*                       faceClr,
*                       embossLtClr, embossDkClr,
*                       pBitmap,
*                       embossSize)
*
* Overview: This macro sets the parameters to draw a panel.
*           Panel is not an object. It will not be added to
*           the active list. Use GOLPanelDrawTsk() to display
*           the panel on the screen.
*           The following relationships of the parameters determines
*           the general shape of the button:
*           1. Panel width is determined by right - left.
*           2. Panel height is determined by top - bottom.
*           3. Panel radius - specifies if the panel will have a rounded
*                       edge. If zero then the panel will have
*                       sharp (cornered) edge.
*           4. If 2*radius = height = width, the panel is circular.
*
* PreCondition: none
*
* Input: left - Panel's left most position.
*        top - Panel's top most position.
*        right - Panel's right most position.
```

```
*        bottom - Panel's bottom most position.
*        radius - The radius of the rounded corner of the panel.
*        faceClr - Defines the face color of the panel.
*        embossLtClr - Defines the emboss light color.
*        embossDkClr - Defines the emboss dark color.
*        pBitmap - Defines the bitmap used in the face of the panel.
*        embossSize - Defines the emboss size of the panel.
*
* Output: none
*
* Example:
*   <CODE>
*   // Dimensions for signature box
*   #define SIG_PANEL_LEFT      10
*   #define SIG_PANEL_RIGHT    310
*   #define SIG_PANEL_TOP       50
*   #define SIG_PANEL_BOTTOM   170
*
*   #define SIG_STATE_SET   0
*   #define SIG_STATE_DRAW  1
*
*   GOL_SCHEME *altScheme;           // style scheme
*
*   // Draws box for signature
*   WORD PanelSignature() {
*       static BYTE state = SIG_STATE_SET;
*
*       if(state == SIG_STATE_SET){
*           // set data for panel drawing (radius = 0)
*           // assume altScheme is defined
*           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; // change state
*       }
*
*       if(!GOLPanelDrawTsk())
*           return 0; // drawing is not completed
*       } else {
*           state = SIG_STATE_SET; // set state to initial
*           return 1; // drawing is done
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define GOLPanelDraw(left, top, right, bottom, radius, faceClr, embossLtClr,
embossDkClr, pBitmap, embossSize) \
    _rpnlX1 =
left;                                                                    \
    _rpnlY1 =
top;                                                                     \
    _rpnlX2 =
right;                                                                   \
    _rpnlY2 =
bottom;                                                                  \
    _rpnlR =
radius;                                                                   \
    _rpnlFaceColor =
faceClr;                                                            \
    _rpnlEmbossLtColor =
embossLtClr;                                                     \
    _rpnlEmbossDkColor =
embossDkClr;                                                     \
    _pRpnlBitmap =
pBitmap;                                                              \
```

```
    _rpnlEmbossSize = embossSize;


     #ifdef USE_GRADIENT

     #define
GOLGradientPanelDraw(pGolScheme)                                                    \
         _gradientScheme.gradientStartColor =
pGolScheme->gradientScheme.gradientStartColor;       \
         _gradientScheme.gradientEndColor =
pGolScheme->gradientScheme.gradientEndColor;           \
         _gradientScheme.gradientType  =
pGolScheme->gradientScheme.gradientType;                  \
         _gradientScheme.gradientLength =
pGolScheme->gradientScheme.gradientLength;

     #endif

/*********************************************************************
* Function: WORD GOLPanelDrawTsk()
*
* 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.
*
* PreCondition: Parameters of the panel must be set by GOLPanelDraw() macro.
*
* Input: none
*
* Output: Returns the status of the panel rendering
*   <CODE>
*          0 – Rendering of the panel is not yet finished.
*          1 – Rendering of the panel is finished.
*   </CODE>
*
* Example:
*   See GOLPanelDraw() example.
*
* Side Effects: none
*
*********************************************************************/
WORD    GOLPanelDrawTsk(void);

/*********************************************************************
* Function: WORD GOLTwoTonePanelDrawTsk()
*
* 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.
*
* PreCondition: Parameters of the panel must be set by GOLPanelDraw() macro.
*
* Input: none
*
* Output: Returns the status of the panel rendering
*   <CODE>
*          0 – Rendering of the panel is not yet finished.
*          1 – Rendering of the panel is finished.
*   </CODE>
*
* Example:
*   Usage is similar to GOLPanelDraw() example.
*
* Side Effects: none
*
*********************************************************************/
WORD    GOLTwoTonePanelDrawTsk(void);

#endif // _GOL_H
```

14.1.3 GOLFontDefault.c

Variables

Name	Description
GOLFontDefault (see page 332)	This is variable GOLFontDefault.

Description

This is file GOLFontDefault.c.

Body Source

```
/*****************************************************************************
 * FileName:        GOLFontDefault.c
 * Processor:       PIC24F, PIC24H, dsPIC
 * Compiler:        MPLAB C30 v3.23
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright © 2010 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.
 *
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * AUTO-GENERATED CODE Graphics Resource Converter version: 3.0.0
 *****************************************************************************/

/*******************************************************************************
* Created from Gentium font.
* Gentium font is Copyright (c) 2003-2005, SIL International (http://scripts.sil.org/).
* All Rights Reserved.
* Gentium font is licensed under the SIL Open Font License, Version 1.0.
* http://scripts.sil.org/OFL
*******************************************************************************/

/*****************************************************************************
 * SECTION:  Includes
 *****************************************************************************/
#include "GraphicsConfig.h"
#ifndef FONTDEFAULT
#include <Graphics/Primitive.h>

#ifdef USE_GOL
/*****************************************************************************
 * Converted Resources
 * -------------------
```

```
 *
 *
 * Fonts
 * -----
 * GOLFontDefault - Heigth: 27 pixels, range: ' ' to '~'
 ****************************************************************************/

/****************************************************************************
 * SECTION:  Fonts
 ****************************************************************************/

/*********************************
 * TYPE_FONT_FILE Structure
 * Label: GOLFontDefault
 * Description:  Heigth: 27 pixels, range: ' ' to '~'
 ************************************/

extern const char __GOLFontDefault[] __attribute__((aligned(2)));

const FONT_FLASH GOLFontDefault =
{
    (FLASH | COMP_NONE),
    __GOLFontDefault
};

const char __GOLFontDefault[] __attribute__((aligned(2))) =
{
/****************************************
 * Font header
 ****************************************/
    0x00,           // Information
    0x00,           // ID
    0x1C, 0x00,     // First Character
    0x7E, 0x00,     // Last Character
    0x1B,           // Height
    0x00,           // Reserved
/****************************************
 * Font Glyph Table
 ****************************************/
    0x10, 0x94, 0x01, 0x00,           // width, MSB Offset, LSB offset
    0x10, 0xCA, 0x01, 0x00,           // width, MSB Offset, LSB offset
    0x11, 0x00, 0x02, 0x00,           // width, MSB Offset, LSB offset
    0x11, 0x51, 0x02, 0x00,           // width, MSB Offset, LSB offset
    0x05, 0xA2, 0x02, 0x00,           // width, MSB Offset, LSB offset
    0x06, 0xBD, 0x02, 0x00,           // width, MSB Offset, LSB offset
    0x0A, 0xD8, 0x02, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x0E, 0x03, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x44, 0x03, 0x00,           // width, MSB Offset, LSB offset
    0x10, 0x7A, 0x03, 0x00,           // width, MSB Offset, LSB offset
    0x10, 0xB0, 0x03, 0x00,           // width, MSB Offset, LSB offset
    0x06, 0xE6, 0x03, 0x00,           // width, MSB Offset, LSB offset
    0x07, 0x01, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x07, 0x1C, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x0A, 0x37, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x09, 0x6D, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x05, 0xA3, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x08, 0xBE, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x05, 0xD9, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0xF4, 0x04, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x2A, 0x05, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x60, 0x05, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x96, 0x05, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0xCC, 0x05, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x02, 0x06, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x38, 0x06, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x6E, 0x06, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0xA4, 0x06, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0xDA, 0x06, 0x00,           // width, MSB Offset, LSB offset
    0x0B, 0x10, 0x07, 0x00,           // width, MSB Offset, LSB offset
    0x05, 0x46, 0x07, 0x00,           // width, MSB Offset, LSB offset
    0x05, 0x61, 0x07, 0x00,           // width, MSB Offset, LSB offset
    0x0A, 0x7C, 0x07, 0x00,           // width, MSB Offset, LSB offset
```

```
    0x0A, 0xB2, 0x07, 0x00,          // width, MSB Offset, LSB offset
    0x0A, 0xE8, 0x07, 0x00,          // width, MSB Offset, LSB offset
    0x0A, 0x1E, 0x08, 0x00,          // width, MSB Offset, LSB offset
    0x13, 0x54, 0x08, 0x00,          // width, MSB Offset, LSB offset
    0x0E, 0xA5, 0x08, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xDB, 0x08, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0x11, 0x09, 0x00,          // width, MSB Offset, LSB offset
    0x0E, 0x47, 0x09, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x7D, 0x09, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0xB3, 0x09, 0x00,          // width, MSB Offset, LSB offset
    0x0D, 0xE9, 0x09, 0x00,          // width, MSB Offset, LSB offset
    0x0F, 0x1F, 0x0A, 0x00,          // width, MSB Offset, LSB offset
    0x07, 0x55, 0x0A, 0x00,          // width, MSB Offset, LSB offset
    0x07, 0x70, 0x0A, 0x00,          // width, MSB Offset, LSB offset
    0x0D, 0x8B, 0x0A, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0xC1, 0x0A, 0x00,          // width, MSB Offset, LSB offset
    0x13, 0xF7, 0x0A, 0x00,          // width, MSB Offset, LSB offset
    0x0F, 0x48, 0x0B, 0x00,          // width, MSB Offset, LSB offset
    0x0E, 0x7E, 0x0B, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xB4, 0x0B, 0x00,          // width, MSB Offset, LSB offset
    0x0E, 0xEA, 0x0B, 0x00,          // width, MSB Offset, LSB offset
    0x0D, 0x20, 0x0C, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x56, 0x0C, 0x00,          // width, MSB Offset, LSB offset
    0x0D, 0x8C, 0x0C, 0x00,          // width, MSB Offset, LSB offset
    0x0F, 0xC2, 0x0C, 0x00,          // width, MSB Offset, LSB offset
    0x0F, 0xF8, 0x0C, 0x00,          // width, MSB Offset, LSB offset
    0x14, 0x2E, 0x0D, 0x00,          // width, MSB Offset, LSB offset
    0x0E, 0x7F, 0x0D, 0x00,          // width, MSB Offset, LSB offset
    0x0E, 0xB5, 0x0D, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xEB, 0x0D, 0x00,          // width, MSB Offset, LSB offset
    0x07, 0x21, 0x0E, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x3C, 0x0E, 0x00,          // width, MSB Offset, LSB offset
    0x08, 0x72, 0x0E, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x8D, 0x0E, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0xC3, 0x0E, 0x00,          // width, MSB Offset, LSB offset
    0x07, 0xF9, 0x0E, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x14, 0x0F, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0x4A, 0x0F, 0x00,          // width, MSB Offset, LSB offset
    0x0A, 0x80, 0x0F, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xB6, 0x0F, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0xEC, 0x0F, 0x00,          // width, MSB Offset, LSB offset
    0x07, 0x22, 0x10, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0x3D, 0x10, 0x00,          // width, MSB Offset, LSB offset
    0x0D, 0x73, 0x10, 0x00,          // width, MSB Offset, LSB offset
    0x06, 0xA9, 0x10, 0x00,          // width, MSB Offset, LSB offset
    0x06, 0xC4, 0x10, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xDF, 0x10, 0x00,          // width, MSB Offset, LSB offset
    0x06, 0x15, 0x11, 0x00,          // width, MSB Offset, LSB offset
    0x14, 0x30, 0x11, 0x00,          // width, MSB Offset, LSB offset
    0x0D, 0x81, 0x11, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xB7, 0x11, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xED, 0x11, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0x23, 0x12, 0x00,          // width, MSB Offset, LSB offset
    0x09, 0x59, 0x12, 0x00,          // width, MSB Offset, LSB offset
    0x09, 0x8F, 0x12, 0x00,          // width, MSB Offset, LSB offset
    0x08, 0xC5, 0x12, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0xE0, 0x12, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x16, 0x13, 0x00,          // width, MSB Offset, LSB offset
    0x10, 0x4C, 0x13, 0x00,          // width, MSB Offset, LSB offset
    0x0C, 0x82, 0x13, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0xB8, 0x13, 0x00,          // width, MSB Offset, LSB offset
    0x0A, 0xEE, 0x13, 0x00,          // width, MSB Offset, LSB offset
    0x08, 0x24, 0x14, 0x00,          // width, MSB Offset, LSB offset
    0x05, 0x3F, 0x14, 0x00,          // width, MSB Offset, LSB offset
    0x08, 0x5A, 0x14, 0x00,          // width, MSB Offset, LSB offset
    0x0B, 0x75, 0x14, 0x00,          // width, MSB Offset, LSB offset
/************************************
 * Font Characters
 ************************************/
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
```

```
0x00, 0x30,          //              **
0x00, 0x38,          //             ***
0x00, 0x3C,          //            ****
0x00, 0x3E,          //           *****
0x00, 0x3F,          //          ******
0x80, 0x3F,          //         *******
0xC0, 0x3F,          //        ********
0xE0, 0x3F,          //       *********
0xF0, 0x3F,          //      **********
0xF8, 0x3F,          //     ***********
0xFC, 0x3F,          //    ************
0xF8, 0x3F,          //     ***********
0xF0, 0x3F,          //      **********
0xE0, 0x3F,          //       *********
0xC0, 0x3F,          //        ********
0x80, 0x3F,          //         *******
0x00, 0x3F,          //          ******
0x00, 0x3E,          //           *****
0x00, 0x3C,          //            ****
0x00, 0x38,          //             ***
0x00, 0x30,          //              **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x0C, 0x00,          //    **
0x1C, 0x00,          //    ***
0x3C, 0x00,          //    ****
0x7C, 0x00,          //    *****
0xFC, 0x00,          //    ******
0xFC, 0x01,          //    *******
0xFC, 0x03,          //    ********
0xFC, 0x07,          //    *********
0xFC, 0x0F,          //    **********
0xFC, 0x1F,          //    ***********
0xFC, 0x3F,          //    ************
0xFC, 0x1F,          //    ***********
0xFC, 0x0F,          //    **********
0xFC, 0x07,          //    *********
0xFC, 0x03,          //    ********
0xFC, 0x01,          //    *******
0xFC, 0x00,          //    ******
0x7C, 0x00,          //    *****
0x3C, 0x00,          //    ****
0x1C, 0x00,          //    ***
0x0C, 0x00,          //    **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //
0x00, 0x01, 0x00,          //          *
0x80, 0x03, 0x00,          //         ***
0x80, 0x03, 0x00,          //         ***
0x80, 0x03, 0x00,          //         ***
0xC0, 0x07, 0x00,          //        *****
0xC0, 0x07, 0x00,          //        *****
0xE0, 0x0F, 0x00,          //       *******
0xE0, 0x0F, 0x00,          //       *******
0xE0, 0x0F, 0x00,          //       *******
0xF0, 0x1F, 0x00,          //      *********
0xF0, 0x1F, 0x00,          //      *********
0xF0, 0x1F, 0x00,          //      *********
0xF8, 0x3F, 0x00,          //     ***********
0xF8, 0x3F, 0x00,          //     ***********
0xF8, 0x3F, 0x00,          //     ***********
0xFC, 0x7F, 0x00,          //    *************
```

```
0xFC, 0x7F, 0x00,          //   *************
0xFC, 0x7F, 0x00,          //   *************
0xFE, 0xFF, 0x00,          //  ***************
0xFE, 0xFF, 0x00,          //  ***************
0xFE, 0xFF, 0x00,          //  ***************
0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //

0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //
0xFE, 0xFF, 0x00,          //  ***************
0xFE, 0xFF, 0x00,          //  ***************
0xFE, 0xFF, 0x00,          //  ***************
0xFC, 0x7F, 0x00,          //   *************
0xFC, 0x7F, 0x00,          //   *************
0xFC, 0x7F, 0x00,          //   *************
0xF8, 0x3F, 0x00,          //    ***********
0xF8, 0x3F, 0x00,          //    ***********
0xF8, 0x3F, 0x00,          //    ***********
0xF0, 0x1F, 0x00,          //     *********
0xF0, 0x1F, 0x00,          //     *********
0xF0, 0x1F, 0x00,          //     *********
0xE0, 0x0F, 0x00,          //      *******
0xE0, 0x0F, 0x00,          //      *******
0xE0, 0x0F, 0x00,          //      *******
0xC0, 0x07, 0x00,          //       *****
0xC0, 0x07, 0x00,          //       *****
0x80, 0x03, 0x00,          //        ***
0x80, 0x03, 0x00,          //        ***
0x80, 0x03, 0x00,          //        ***
0x00, 0x01, 0x00,          //         *
0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //
0x00, 0x00, 0x00,          //

0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x08,          //     *
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
```

```
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x04,          //   *
0x00,          //
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0xC4, 0x00,          //   *   **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x60, 0x02,          //       **  *
0x20, 0x03,          //       *  **
0x20, 0x01,          //       *  *
0x20, 0x01,          //       *  *
0xFC, 0x07,          //    *********
0x90, 0x01,          //      *  **
0x90, 0x00,          //      *  *
0x98, 0x00,          //     **  *
0xFE, 0x03,          //   *********
0x48, 0x00,          //     *  *
0x4C, 0x00,          //    **  *
0x44, 0x00,          //    *   *
0x64, 0x00,          //    *  **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x60, 0x00,          //       **
0x60, 0x00,          //       **
0xF8, 0x01,          //     ******
0x6C, 0x03,          //    ** ** **
0x66, 0x01,          //   **  ** *
0x66, 0x00,          //   **  **
0x6E, 0x00,          //   *** **
0x7C, 0x00,          //    *****
0xF8, 0x01,          //     ******
0xE0, 0x01,          //       ****
0xE0, 0x03,          //       *****
0x60, 0x03,          //       ** **
0x62, 0x03,          //   *   ** **
0x62, 0x03,          //   *   ** **
0x66, 0x01,          //   **  ** *
0xFC, 0x00,          //    ******
0x60, 0x00,          //       **
0x60, 0x00,          //       **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x38, 0x30,          //     ***      **
0x24, 0x18,          //    *  *     **
0x66, 0x0C,          //   **  **   **
0x66, 0x04,          //   **  **   *
0x66, 0x06,          //   **  **  **
0x66, 0x03,          //   **  ** **
0xA4, 0x01,          //    *  * **
0x9C, 0x38,          //    ***  *   ***
0xC0, 0x64,          //        **  *  **
0x60, 0x66,          //       **  **  **
0x20, 0x66,          //       *   **  **
0x10, 0x66,          //      *    **  **
0x18, 0x24,          //     **     *  *
0x0C, 0x1C,          //    **      ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xE0, 0x01,          //       ****
0x30, 0x03,          //      **  **
0x18, 0x03,          //     **   **
0x18, 0x03,          //     **   **
0x18, 0x01,          //     **   *
0x98, 0x01,          //     **  **
0x78, 0x00,          //     ****
```

```
0x78, 0x7E,          //     ****  ******
0x7C, 0x1C,          //   *****   ***
0xCC, 0x18,          //   **  **   **
0xC6, 0x19,          //  **   ***  **
0x86, 0x1B,          //  **    *** **
0x06, 0x0F,          //  **     ****
0x06, 0x0E,          //  **      ***
0x0C, 0x1A,          //   **     * **
0xF8, 0x71,          //    ******   ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,         //
0x00,         //
0x00,         //
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x04,         //   *
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //

0x00,         //
0x00,         //
0x60,         //      **
0x30,         //     **
0x10,         //     *
0x08,         //    *
0x0C,         //   **
0x0C,         //   **
0x04,         //   *
0x06,         //  **
0x06,         //  **
0x06,         //  **
0x06,         //  **
0x06,         //  **
0x06,         //  **
0x06,         //  **
0x06,         //  **
0x04,         //   *
0x0C,         //   **
0x0C,         //   **
0x08,         //    *
0x18,         //    **
0x30,         //     **
0x60,         //      **
0x00,         //
0x00,         //
0x00,         //
```

```
0x00,          //
0x00,          //
0x03,          // **
0x06,          //  **
0x0C,          //   **
0x08,          //    *
0x18,          //    **
0x18,          //    **
0x10,          //     *
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x10,          //     *
0x18,          //    **
0x18,          //    **
0x08,          //    *
0x04,          //   *
0x06,          //  **
0x03,          // **
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x22, 0x01,          //  *   *  *
0xAE, 0x03,          //  *** * ***
0x78, 0x00,          //    ****
0x78, 0x00,          //    ****
0xAE, 0x01,          //  *** * **
0x22, 0x01,          //  *   *  *
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0xFE, 0x01,          //  ********
0x30, 0x00,          //     **
```

```
0x30, 0x00,          //      **
0x30, 0x00,          //      **
0x30, 0x00,          //      **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x0E,          //  ***
0x0F,          // ****
0x0C,          //   **
0x0C,          //   **
0x04,          //   *
0x02,          //  *
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x7E,          //  ******
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00,          //
0x00,          //
0x00,          //
```

```
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x02,          //           *
0x00, 0x03,          //          **
0x00, 0x01,          //          *
0x80, 0x01,          //         **
0x80, 0x01,          //         **
0x80, 0x00,          //         *
0xC0, 0x00,          //        **
0xC0, 0x00,          //        **
0x60, 0x00,          //       **
0x60, 0x00,          //       **
0x20, 0x00,          //       *
0x30, 0x00,          //      **
0x30, 0x00,          //      **
0x10, 0x00,          //      *
0x18, 0x00,          //     **
0x18, 0x00,          //     **
0x0C, 0x00,          //    **
0x0C, 0x00,          //    **
0x04, 0x00,          //    *
0x06, 0x00,          //   **
0x06, 0x00,          //   **
0x02, 0x00,          //   *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF0, 0x00,          //      ****
0xC8, 0x01,          //     *  ***
0x8C, 0x01,          //    **   **
0x84, 0x03,          //    *    ***
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x01,          //   **     *
0x0C, 0x01,          //    **    *
0x8C, 0x00,          //    **   *
```

```
    0x70, 0x00,         //     ***
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //

    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x60, 0x00,         //       **
    0x78, 0x00,         //     ****
    0x64, 0x00,         //    *  **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0x60, 0x00,         //       **
    0xFC, 0x03,         //    ********
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //

    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x78, 0x00,         //     ****
    0xC4, 0x01,         //    *   ***
    0x86, 0x01,         //   **     **
    0x82, 0x01,         //   *      **
    0x80, 0x01,         //          **
    0xC0, 0x00,         //        **
    0xC0, 0x00,         //        **
    0x60, 0x00,         //       **
    0x30, 0x00,         //      **
    0x30, 0x00,         //      **
    0x18, 0x00,         //     **
    0x0C, 0x01,         //    **     *
    0x06, 0x01,         //   **      *
    0xFE, 0x01,         //   ********
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //

    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0xF8, 0x00,         //     *****
```

```
0xCC, 0x01,          //   **  ***
0x86, 0x01,          //  **    **
0x80, 0x01,          //        **
0x80, 0x01,          //        **
0x40, 0x00,          //       *
0xF0, 0x00,          //     ****
0x80, 0x01,          //        **
0x00, 0x03,          //         **
0x00, 0x03,          //         **
0x00, 0x03,          //         **
0x00, 0x03,          //         **
0x86, 0x01,          //  **    **
0x78, 0x00,          //    ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xC0, 0x00,          //       **
0xE0, 0x00,          //      ***
0xE0, 0x00,          //      ***
0xD0, 0x00,          //     * **
0xD8, 0x00,          //    ** **
0xC8, 0x00,          //    *  **
0xC4, 0x00,          //   *   **
0xC6, 0x00,          //  **   **
0xC2, 0x00,          //  *    **
0xFF, 0x03,          // **********
0xC0, 0x00,          //       **
0xC0, 0x00,          //       **
0xC0, 0x00,          //       **
0xF0, 0x03,          //     ******
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x01,          //          *
0xFC, 0x00,          //   ******
0x04, 0x00,          //   *
0x04, 0x00,          //   *
0x06, 0x00,          //  **
0x7E, 0x00,          //  ******
0xC2, 0x00,          //  *    **
0x80, 0x01,          //        **
0x80, 0x01,          //        **
0x80, 0x01,          //        **
0x80, 0x01,          //        **
0x80, 0x00,          //        *
0xC3, 0x00,          // **    **
0x3C, 0x00,          //   ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x80, 0x01,          //         **
0x60, 0x00,          //       **
0x18, 0x00,          //     **
0x08, 0x00,          //     *
0x0C, 0x00,          //    **
0xF6, 0x00,          //   ** ****
0x8E, 0x01,          //   ***   **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x06, 0x03,          //   **     **
0x0C, 0x01,          //    **    *
0x8C, 0x01,          //    **   **
0xF0, 0x00,          //      ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFC, 0x03,          //    ********
0x02, 0x01,          //   *      *
0x02, 0x01,          //   *      *
0x80, 0x01,          //         **
0x80, 0x00,          //         *
0xC0, 0x00,          //        **
0x40, 0x00,          //        *
0x60, 0x00,          //       **
0x20, 0x00,          //       *
0x30, 0x00,          //      **
0x30, 0x00,          //      **
0x18, 0x00,          //     **
0x08, 0x00,          //     *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF8, 0x00,          //     *****
0xCC, 0x01,          //    **  ***
0x86, 0x01,          //   **    **
0x86, 0x01,          //   **    **
0x8E, 0x01,          //   ***   **
```

```
0xDC, 0x00,          //   *** **
0xF0, 0x00,          //     ****
0xC8, 0x01,          //    *  ***
0x8C, 0x03,          //   **   ***
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x8C, 0x01,          //   **   **
0x78, 0x00,          //    ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF0, 0x00,          //     ****
0x8C, 0x01,          //   **   **
0x84, 0x01,          //   *    **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x8C, 0x03,          //   **   ***
0x78, 0x03,          //    **** **
0x00, 0x01,          //         *
0x80, 0x01,          //        **
0xC0, 0x00,          //       **
0x20, 0x00,          //      *
0x1C, 0x00,          //   ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
```

```
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x0E,          //  ***
0x0F,          // ****
0x0C,          //    **
0x0C,          //    **
0x04,          //    *
0x02,          //  *
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x01,          //          *
0xE0, 0x01,          //       ****
0x3C, 0x00,          //   ****
0x06, 0x00,          //  **
0x1C, 0x00,          //   ***
0xF0, 0x00,          //      ****
0x80, 0x01,          //         **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFE, 0x01,          //  ********
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0xFE, 0x01,          //  ********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x06, 0x00,          //  **
0x3C, 0x00,          //   ****
0xE0, 0x01,          //      ****
0x80, 0x01,          //        **
0xF0, 0x00,          //     ****
0x1E, 0x00,          //  ****
0x02, 0x00,          //  *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF8, 0x00,          //    *****
0xCC, 0x01,          //   **  ***
0x86, 0x01,          //  **    **
0x86, 0x01,          //  **    **
0x80, 0x01,          //        **
0x80, 0x01,          //        **
0xC0, 0x00,          //       **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x10, 0x00,          //     *
0x00, 0x00,          //
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x30, 0x00,          //     **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
```

```
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x3F, 0x00,         //         ******
0xC0, 0x70, 0x00,         //       **    ***
0x30, 0xC0, 0x00,         //     **        **
0x18, 0x80, 0x01,         //    **          **
0x08, 0x80, 0x01,         //    *           **
0x0C, 0x97, 0x03,         //   **    *** *  ***
0x84, 0x18, 0x03,         //   *    *   **   **
0x46, 0x18, 0x03,         //  **   *    **   **
0x66, 0x18, 0x03,         //  **  **    **   **
0x66, 0x18, 0x03,         //  **  **    **   **
0x66, 0x18, 0x03,         //  **  **    **   **
0x66, 0x18, 0x01,         //  **  **    **   *
0x6E, 0x98, 0x01,         //  *** **    **  **
0xCC, 0x9C, 0x00,         //   **  **  ***  *
0x8C, 0x73, 0x00,         //   **   ***  ***
0x18, 0x00, 0x00,         //    **
0x38, 0x80, 0x00,         //    ***         *
0xE0, 0xE0, 0x01,         //      ***     ****
0x80, 0x1F, 0x00,         //        ******
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x40, 0x00,         //       *
0xC0, 0x00,         //       **
0xA0, 0x00,         //      * *
0xA0, 0x01,         //      * **
0xA0, 0x01,         //      * **
0x10, 0x01,         //     *   *
0x10, 0x03,         //     *   **
0x10, 0x03,         //     *   **
0xF8, 0x03,         //    *******
0x08, 0x06,         //    *     **
0x08, 0x06,         //    *     **
0x04, 0x0C,         //   *       **
0x04, 0x0C,         //   *       **
0x1F, 0x1E,         // *****    ****
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0xFE, 0x00,         //  *******
0x8D, 0x03,         // * **   ***
0x0C, 0x03,         //   **    **
0x0C, 0x03,         //   **    **
0x0C, 0x03,         //   **    **
0x8C, 0x01,         //   **   **
0xFC, 0x01,         //   *******
0x8C, 0x03,         //   **   ***
0x0C, 0x07,         //   **    ***
0x0C, 0x06,         //   **     **
0x0C, 0x06,         //   **     **
0x0C, 0x06,         //   **     **
0x0C, 0x03,         //   **    **
```

```
0xFF, 0x01,          // *********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xC0, 0x07,          //        *****
0x30, 0x0C,          //      **    **
0x08, 0x00,          //     *
0x0C, 0x00,          //    **
0x04, 0x00,          //    *
0x06, 0x00,          //   **
0x06, 0x00,          //   **
0x06, 0x00,          //   **
0x06, 0x00,          //   **
0x06, 0x00,          //   **
0x0C, 0x00,          //    **
0x0C, 0x08,          //    **        *
0x38, 0x04,          //     ***     *
0xE0, 0x03,          //       *****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFE, 0x01,          //   ********
0x0D, 0x07,          // * **    ***
0x0C, 0x0E,          //   **      ***
0x0C, 0x0C,          //   **       **
0x0C, 0x18,          //   **        **
0x0C, 0x18,          //   **        **
0x0C, 0x18,          //   **        **
0x0C, 0x18,          //   **        **
0x0C, 0x18,          //   **        **
0x0C, 0x18,          //   **        **
0x0C, 0x0C,          //   **       **
0x0C, 0x0E,          //   **      ***
0x0C, 0x07,          //   **    ***
0xFF, 0x01,          // *********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFF, 0x03,          // **********
```

```
0x0C, 0x02,          //   **     *
0x0C, 0x02,          //   **     *
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0xFC, 0x01,          //   *******
0x8C, 0x00,          //   **   *
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x04,          //   **      *
0x0C, 0x06,          //   **     **
0xFF, 0x03,          // **********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFF, 0x03,          // **********
0x0C, 0x02,          //   **     *
0x0C, 0x02,          //   **     *
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0xFC, 0x01,          //   *******
0x8C, 0x00,          //   **   *
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x3F, 0x00,          // ******
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xE0, 0x03,          //      *****
0x10, 0x06,          //     *    **
0x08, 0x00,          //    *
0x0C, 0x00,          //   **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x86, 0x0F,          //  **    *****
0x06, 0x06,          //  **      **
0x06, 0x06,          //  **      **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x18, 0x06,          //    **    **
0xF0, 0x01,          //     *****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x3F, 0x7E,         // ******   ******
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0xFC, 0x1F,         //   ***********
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x0C, 0x18,         //   **       **
0x3F, 0x7E,         // ******   ******
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //

0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x3F,         // ******
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x3F,         // ******
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //

0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x7E,         //  ******
0x18,         //    **
0x18,         //    **
0x18,         //    **
0x18,         //    **
```

```
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x18,         //    **
    0x08,         //    *
    0x04,         //   *
    0x03,         // **
    0x00,         //
    0x00,         //
    0x00,         //

    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x3F, 0x0F,         // ******  ****
    0x0C, 0x02,         //   **     *
    0x0C, 0x01,         //   **    *
    0x8C, 0x00,         //   **   *
    0x4C, 0x00,         //   **  *
    0x6C, 0x00,         //   ** **
    0x3C, 0x00,         //   ****
    0x6C, 0x00,         //   ** **
    0xEC, 0x00,         //   ** ***
    0xCC, 0x00,         //   **  **
    0x8C, 0x01,         //   **   **
    0x0C, 0x03,         //   **    **
    0x0C, 0x06,         //   **     **
    0x3F, 0x1C,         // ******    ***
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //

    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x3F, 0x00,         // ******
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x00,         //   **
    0x0C, 0x04,         //   **      *
    0x0C, 0x02,         //   **     *
    0xFF, 0x03,         // **********
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
```

```
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x1E, 0xC0, 0x01,         //  ****         ***
0x1C, 0xE0, 0x00,         //   ***        ***
0x1C, 0xE0, 0x00,         //   ***        ***
0x3C, 0xF0, 0x00,         //   ****      ****
0x2C, 0xD0, 0x00,         //   ** *      * **
0x6C, 0xD0, 0x00,         //   ** **     * **
0x6C, 0xC8, 0x00,         //   ** **    *  **
0xCC, 0xC8, 0x00,         //   **  **   *  **
0xCC, 0xCC, 0x00,         //   **  **  **  **
0x8C, 0xC5, 0x00,         //   **   ** *   **
0x8C, 0xC7, 0x00,         //   **   ****   **
0x0C, 0xC3, 0x00,         //   **    **    **
0x0C, 0xC3, 0x00,         //   **    **    **
0x1E, 0xE1, 0x03,         //  ****   *    *****
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x0F, 0x7E,         // ****     ******
0x0C, 0x18,         //   **       **
0x1C, 0x18,         //   ***      **
0x3C, 0x18,         //   ****     **
0x2C, 0x18,         //   ** *     **
0x4C, 0x18,         //   **  *    **
0xCC, 0x18,         //   **  **   **
0x8C, 0x19,         //   **   **  **
0x0C, 0x19,         //   **    *  **
0x0C, 0x1B,         //   **    ** **
0x0C, 0x1E,         //   **     ****
0x0C, 0x1C,         //   **      ***
0x0C, 0x1C,         //   **      ***
0x3F, 0x18,         // ******     **
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0xE0, 0x03,         //      *****
0x18, 0x06,         //    **    **
0x08, 0x0C,         //    *      **
0x0C, 0x0C,         //   **      **
0x06, 0x18,         //  **        **
0x06, 0x18,         //  **        **
0x06, 0x18,         //  **        **
0x06, 0x18,         //  **        **
0x06, 0x18,         //  **        **
```

```
0x06, 0x18,          //  **        **
0x0C, 0x0C,          //   **      **
0x0C, 0x04,          //   **      *
0x18, 0x02,          //    **    *
0xF0, 0x01,          //     *****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFE, 0x01,          //  ********
0x0D, 0x03,          // * **    **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x03,          //   **    **
0xFC, 0x00,          //   ******
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x3F, 0x00,          // ******
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xE0, 0x03,          //      *****
0x18, 0x06,          //    **    **
0x08, 0x0C,          //    *      **
0x0C, 0x0C,          //   **      **
0x06, 0x18,          //  **        **
0x06, 0x18,          //  **        **
0x06, 0x18,          //  **        **
0x06, 0x18,          //  **        **
0x06, 0x18,          //  **        **
0x06, 0x18,          //  **        **
0x0C, 0x0C,          //   **      **
0x0C, 0x0C,          //   **      **
0x18, 0x06,          //    **    **
0xF0, 0x01,          //     *****
0x00, 0x03,          //         **
0x00, 0x06,          //           **
0x00, 0x38,          //             ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFE, 0x01,          //  ********
0x0D, 0x03,          // * **    **
0x0C, 0x06,          //   **      **
0x0C, 0x06,          //   **      **
0x0C, 0x06,          //   **      **
0x0C, 0x03,          //   **    **
0xFC, 0x00,          //   ******
0xCC, 0x00,          //   **  **
0xCC, 0x00,          //   **  **
0x8C, 0x01,          //   **   **
0x0C, 0x03,          //   **    **
0x0C, 0x03,          //   **    **
0x0C, 0x06,          //   **      **
0x3F, 0x1C,          // ******    ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF8, 0x00,          //    *****
0x8C, 0x01,          //   **   **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x0E, 0x00,          //  ***
0x3C, 0x00,          //   ****
0xF8, 0x00,          //    *****
0xE0, 0x01,          //      ****
0x80, 0x03,          //         ***
0x00, 0x03,          //          **
0x02, 0x03,          //  *       **
0x02, 0x01,          //  *       *
0x86, 0x01,          //  **    **
0x7C, 0x00,          //   *****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFF, 0x0F,          // ************
0x61, 0x08,          // *    **    *
0x61, 0x08,          // *    **    *
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x60, 0x00,          //      **
```

```
0xF8, 0x01,          //    ******
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x3F, 0x7E,          // ******   ******
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x0C, 0x18,          //   **       **
0x18, 0x0C,          //    **     **
0x38, 0x06,          //    ***   **
0xE0, 0x03,          //      *****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x1F, 0x3C,          // *****     ****
0x0C, 0x18,          //   **       **
0x0C, 0x08,          //   **       *
0x18, 0x08,          //    **      *
0x18, 0x0C,          //    **     **
0x18, 0x04,          //    **     *
0x30, 0x06,          //     **   **
0x30, 0x02,          //     **   *
0x20, 0x02,          //      *   *
0x60, 0x03,          //      ** **
0x60, 0x01,          //      ** *
0xC0, 0x01,          //       ***
0xC0, 0x01,          //       ***
0x80, 0x00,          //        *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x1F, 0xC2, 0x07,         // *****    *    *****
```

```
0x0C, 0x06, 0x01,         //   **      **      *
0x0C, 0x06, 0x01,         //   **      **      *
0x0C, 0x07, 0x01,         //   **     ***      *
0x0C, 0x8D, 0x01,         //   **     * **    **
0x08, 0x8D, 0x00,         //    *     * **    *
0x98, 0x89, 0x00,         //    **   **  *    *
0x98, 0x98, 0x00,         //    **   *    **   *
0x98, 0x98, 0x00,         //    **   *    **   *
0xD0, 0xD0, 0x00,         //     * **     * **
0x70, 0x70, 0x00,         //     ***      ***
0x70, 0x70, 0x00,         //     ***      ***
0x70, 0x60, 0x00,         //     ***       **
0x20, 0x60, 0x00,         //      *        **
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x3F, 0x1F,         // ******  *****
0x0C, 0x0C,         //   **      **
0x18, 0x06,         //    **    **
0x38, 0x03,         //    ***  **
0x30, 0x01,         //     **  *
0xE0, 0x01,         //      ****
0xC0, 0x00,         //       **
0xC0, 0x01,         //       ***
0xA0, 0x01,         //      * **
0x30, 0x03,         //     **  **
0x18, 0x06,         //    **    **
0x08, 0x0E,         //    *     ***
0x0C, 0x0C,         //   **      **
0x1F, 0x3E,         // *****    *****
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x07, 0x3E,         // ***      *****
0x0C, 0x0C,         //   **      **
0x18, 0x04,         //    **     *
0x18, 0x02,         //    **    *
0x30, 0x03,         //     **  **
0x60, 0x01,         //      ** *
0xE0, 0x01,         //      ****
0xC0, 0x00,         //       **
0xC0, 0x00,         //       **
0xC0, 0x00,         //       **
0xC0, 0x00,         //       **
0xC0, 0x00,         //       **
0xC0, 0x00,         //       **
0xF0, 0x03,         //     ******
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFC, 0x07,          //   *********
0x04, 0x03,          //   *      **
0x02, 0x03,          //  *       **
0x80, 0x01,          //         **
0xC0, 0x00,          //        **
0xC0, 0x00,          //        **
0x60, 0x00,          //       **
0x70, 0x00,          //      ***
0x30, 0x00,          //      **
0x18, 0x00,          //    **
0x18, 0x00,          //    **
0x0C, 0x04,          //   **       *
0x06, 0x04,          //  **        *
0xFE, 0x07,          //  **********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,          //
0x00,          //
0x3E,          //  *****
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x06,          //  **
0x3E,          //  *****
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x02, 0x00,          //  *
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x04, 0x00,          //   *
0x0C, 0x00,          //   **
0x08, 0x00,          //    *
0x18, 0x00,          //    **
0x18, 0x00,          //    **
0x10, 0x00,          //     *
```

```
0x30, 0x00,          //      **
0x30, 0x00,          //      **
0x20, 0x00,          //       *
0x60, 0x00,          //       **
0x40, 0x00,          //        *
0x40, 0x00,          //        *
0xC0, 0x00,          //        **
0x80, 0x00,          //         *
0x80, 0x01,          //         **
0x80, 0x01,          //         **
0x00, 0x01,          //          *
0x00, 0x03,          //          **
0x00, 0x02,          //           *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,          //
0x00,          //
0x3E,          //  *****
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x30,          //     **
0x3E,          //  *****
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x20, 0x00,          //       *
0x60, 0x00,          //       **
0x70, 0x00,          //      ***
0xD0, 0x00,          //      * **
0xD8, 0x00,          //     ** **
0x88, 0x00,          //     *   *
0x88, 0x01,          //     *   **
0x8C, 0x01,          //    **   **
0x04, 0x03,          //    *      **
0x04, 0x03,          //    *      **
0x02, 0x03,          //   *       **
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFE, 0x03,          //   *********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,          //
0x00,          //
0x00,          //
0x03,          // **
0x06,          //  **
0x0C,          //   **
0x08,          //    *
0x10,          //     *
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF0, 0x00,          //      ****
0x8C, 0x01,          //   **    **
0x86, 0x01,          //  **     **
0x80, 0x01,          //         **
0xF0, 0x01,          //      *****
```

```
0x8C, 0x01,          //   **   **
0x86, 0x01,          //  **    **
0x86, 0x01,          //  **    **
0xC6, 0x01,          //  **   ***
0xBC, 0x07,          //   **** ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x0C, 0x00,          //   **
0x0E, 0x00,          //  ***
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0xCC, 0x01,          //   **  ***
0x3C, 0x03,          //   ****  **
0x1C, 0x07,          //   ***   ***
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x02,          //   **     *
0x1C, 0x01,          //   ***   *
0xF0, 0x00,          //     ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xE0, 0x01,          //      ****
0x18, 0x03,          //    **   **
0x0C, 0x00,          //   **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x0C, 0x00,          //   **
0x1C, 0x03,          //   ***   **
0xF0, 0x00,          //     ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x03,          //          **
0xC0, 0x03,          //        ****
0x00, 0x03,          //          **
0x00, 0x03,          //          **
0x00, 0x03,          //          **
0x00, 0x03,          //          **
0x00, 0x03,          //          **
0xF0, 0x03,          //      ******
0x88, 0x03,          //     *   ***
0x04, 0x03,          //    *     **
0x06, 0x03,          //  **      **
0x06, 0x03,          //  **      **
0x06, 0x03,          //  **      **
0x06, 0x03,          //  **      **
0x0E, 0x03,          //  ***     **
0xCC, 0x0B,          //   **  **** *
0x38, 0x07,          //    ***  ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF0, 0x00,          //      ****
0x08, 0x03,          //     *    **
0x04, 0x03,          //    *     **
0xFE, 0x03,          //  *********
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x06, 0x00,          //  **
0x0C, 0x00,          //   **
0x1C, 0x03,          //   ***   **
0xF0, 0x00,          //      ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,         //
0x00,         //
0x00,         //
0x60,         //      **
0x50,         //     * *
0x18,         //    **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x7E,         //  ******
0x0D,         // * **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
```

```
    0x3F,          // ******
    0x00,          //
    0x00,          //
    0x00,          //
    0x00,          //
    0x00,          //
    0x00,          //
    0x00,          //

    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0xF8, 0x0C,          //     *****  **
    0x8C, 0x03,          //    **   ***
    0x06, 0x03,          //   **     **
    0x06, 0x03,          //   **     **
    0x06, 0x03,          //   **     **
    0x8C, 0x01,          //    **   **
    0xF8, 0x00,          //     *****
    0x18, 0x00,          //     **
    0x1C, 0x00,          //    ***
    0xF8, 0x03,          //     *******
    0x06, 0x0E,          //   **      ***
    0x03, 0x0C,          // **         **
    0x03, 0x0C,          // **         **
    0x06, 0x06,          //   **      **
    0xFC, 0x01,          //    *******
    0x00, 0x00,          //
    0x00, 0x00,          //

    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x0C, 0x00,          //    **
    0x0F, 0x00,          // ****
    0x0C, 0x00,          //    **
    0x0C, 0x00,          //    **
    0x0C, 0x00,          //    **
    0x0C, 0x00,          //    **
    0x0C, 0x00,          //    **
    0xCC, 0x03,          //    **  ****
    0x2C, 0x06,          //    ** *   **
    0x1C, 0x06,          //    ***    **
    0x0C, 0x06,          //    **     **
    0x0C, 0x06,          //    **     **
    0x0C, 0x06,          //    **     **
    0x0C, 0x06,          //    **     **
    0x0C, 0x06,          //    **     **
    0x0C, 0x06,          //    **     **
    0xBF, 0x1F,          // ****** ******
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //
    0x00, 0x00,          //

    0x00,          //
    0x00,          //
    0x00,          //
    0x00,          //
    0x00,          //
    0x0C,          //    **
    0x0C,          //    **
```

```
0x00,          //
0x00,          //
0x00,          //
0x0C,          //   **
0x0F,          // ****
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x3F,          // ******
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x0C,          //   **
0x0C,          //   **
0x00,          //
0x00,          //
0x00,          //
0x0C,          //   **
0x0F,          // ****
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x0C,          //   **
0x04,          //   *
0x02,          //  *
0x01,          // *
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x0C, 0x00,          //   **
0x0F, 0x00,          // ****
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x8C, 0x07,          //   **   ****
0x8C, 0x01,          //   **   **
0xCC, 0x00,          //   **  **
0x2C, 0x00,          //   ** *
0x3C, 0x00,          //   ****
0x6C, 0x00,          //   ** **
0xCC, 0x00,          //   **  **
0x8C, 0x01,          //   **   **
0x0C, 0x03,          //   **    **
0x1F, 0x0F,          // *****   ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //

0x00,         //
0x00,         //
0x00,         //
0x0C,         //   **
0x0F,         // ****
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x0C,         //   **
0x3F,         // ******
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //
0x00,         //

0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x8E, 0xE3, 0x01,         //  ***   ***   ****
0x6D, 0x16, 0x03,         // * ** **  ** *   **
0x1C, 0x0E, 0x03,         //   ***    ***    **
0x0C, 0x06, 0x03,         //   **     **     **
0x0C, 0x06, 0x03,         //   **     **     **
0x0C, 0x06, 0x03,         //   **     **     **
0x0C, 0x06, 0x03,         //   **     **     **
0x0C, 0x06, 0x03,         //   **     **     **
0x0C, 0x06, 0x03,         //   **     **     **
0xBF, 0xDF, 0x0F,         // ****** ****** ******
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //
0x00, 0x00, 0x00,         //

0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x00, 0x00,         //
0x8E, 0x03,         //  ***   ***
```

```
0x6D, 0x06,          // * ** **  **
0x1C, 0x06,          //   ***    **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0xBF, 0x1F,          // ****** ******
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF0, 0x00,          //     ****
0x88, 0x03,          //    *   ***
0x04, 0x03,          //   *     **
0x06, 0x06,          //  **      **
0x06, 0x06,          //  **      **
0x06, 0x06,          //  **      **
0x06, 0x06,          //  **      **
0x0C, 0x02,          //   **     *
0x1C, 0x01,          //   ***   *
0xF0, 0x00,          //     ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xCE, 0x01,          //  ***  ***
0x3D, 0x03,          // * ****  **
0x1C, 0x07,          //   ***   ***
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x06,          //   **     **
0x0C, 0x02,          //   **     *
0x1C, 0x03,          //   ***   **
0xEC, 0x00,          //   ** ***
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x3F, 0x00,          // ******
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xF0, 0x02,          //      **** *
0x88, 0x03,          //     *   ***
0x0C, 0x03,          //   **    **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x06, 0x03,          //  **     **
0x0E, 0x03,          //  ***    **
0x8C, 0x03,          //   **   ***
0x78, 0x03,          //    **** **
0x00, 0x03,          //         **
0x00, 0x03,          //         **
0x00, 0x03,          //         **
0x00, 0x03,          //         **
0xC0, 0x0F,          //       ******
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xCC, 0x01,          //   **  ***
0xAF, 0x01,          // **** * **
0x9C, 0x00,          //   ***  *
0x1C, 0x00,          //   ***
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x0C, 0x00,          //   **
0x3F, 0x00,          // ******
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x7C, 0x00,          //   *****
0x66, 0x00,          //  **  **
0x06, 0x00,          //  **
0x0E, 0x00,          //  ***
0x3C, 0x00,          //   ****
```

```
0xF0, 0x00,          //      ****
0xC0, 0x00,          //        **
0xC2, 0x00,          //  *     **
0xC2, 0x00,          //  *     **
0x3E, 0x00,          //  *****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x08,          //     *
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0xFE,          //  *******
0x0D,          // * **
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0x0C,          //    **
0x8C,          //    **    *
0x78,          //     ****
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //
0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x0C, 0x03,          //    **     **
0xCE, 0x03,          //  ***   ****
0x0C, 0x03,          //    **     **
0x0C, 0x03,          //    **     **
0x0C, 0x03,          //    **     **
0x0C, 0x03,          //    **     **
0x0C, 0x03,          //    **     **
0x0C, 0x03,          //    **     **
0xCC, 0x0B,          //    **   **** *
0x38, 0x07,          //     ***   ***
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
```

```
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x9F, 0x07,          // *****  ****
0x04, 0x02,          //   *      *
0x0C, 0x01,          //   **     *
0x0C, 0x01,          //   **     *
0x98, 0x01,          //    **  **
0x98, 0x00,          //    **  *
0x90, 0x00,          //     *  *
0x70, 0x00,          //     ***
0x60, 0x00,          //      **
0x20, 0x00,          //      *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x1F, 0xF1,          // *****   *   ****
0x04, 0x41,          //   *     *     *
0x84, 0x23,          //   *    ***   *
0x8C, 0x23,          //   **   ***   *
0x88, 0x26,          //    *   * **  *
0x48, 0x36,          //    *  *  ** **
0x58, 0x14,          //    ** *   * *
0x30, 0x1C,          //     **    ***
0x30, 0x1C,          //     **    ***
0x30, 0x08,          //     **     *
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x9F, 0x07,          // *****  ****
0x0C, 0x01,          //   **    *
0x98, 0x00,          //    **  *
0xF0, 0x00,          //     ****
0x60, 0x00,          //      **
0x70, 0x00,          //     ***
0xD0, 0x00,          //     * **
0x88, 0x01,          //    *   **
0x04, 0x03,          //   *     **
```

```
0x8F, 0x07,          // ****   ****
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x9F, 0x07,          // *****  ****
0x04, 0x02,          //   *      *
0x0C, 0x01,          //   **    *
0x0C, 0x01,          //   **    *
0x98, 0x01,          //    **  **
0x98, 0x00,          //    **  *
0x90, 0x00,          //     *  *
0x70, 0x00,          //     ***
0x60, 0x00,          //      **
0x60, 0x00,          //      **
0x20, 0x00,          //      *
0x30, 0x00,          //     **
0x10, 0x00,          //     *
0x0F, 0x00,          // ****
0x07, 0x00,          // ***
0x00, 0x00,          //
0x00, 0x00,          //

0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0xFE, 0x01,          //  ********
0x82, 0x01,          //  *      **
0xC2, 0x00,          //  *    **
0x60, 0x00,          //      **
0x20, 0x00,          //      *
0x30, 0x00,          //     **
0x18, 0x00,          //    **
0x0C, 0x01,          //   **    *
0x04, 0x01,          //   *     *
0xFE, 0x01,          //  ********
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //
0x00, 0x00,          //

0x00,         //
0x00,         //
0x40,         //       *
0x30,         //     **
0x10,         //     *
0x18,         //    **
0x18,         //    **
```

```
    0x18,           //     **
    0x38,           //    ***
    0x30,           //     **
    0x30,           //     **
    0x10,           //     *
    0x08,           //    *
    0x1E,           //  ****
    0x30,           //     **
    0x30,           //     **
    0x30,           //     **
    0x30,           //     **
    0x18,           //    **
    0x18,           //    **
    0x18,           //    **
    0x18,           //    **
    0x30,           //     **
    0x40,           //       *
    0x00,           //
    0x00,           //
    0x00,           //

    0x00,           //
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x00,           //
    0x00,           //

    0x00,           //
    0x00,           //
    0x02,           //  *
    0x0C,           //   **
    0x18,           //    **
    0x18,           //    **
    0x18,           //    **
    0x18,           //    **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x0C,           //   **
    0x78,           //    ****
    0x10,           //     *
    0x08,           //    *
    0x0C,           //   **
    0x0C,           //   **
    0x1C,           //   ***
    0x18,           //    **
    0x18,           //    **
    0x18,           //    **
    0x08,           //    *
    0x0C,           //   **
```

```
    0x02,         //  *
    0x00,         //
    0x00,         //
    0x00,         //

    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x18, 0x04,         //    **      *
    0x7C, 0x02,         //   *****   *
    0xF2, 0x01,         //  *   *****
    0xC2, 0x01,         //  *     ***
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //
    0x00, 0x00,         //

};

#endif // USE_GOL
#endif // #ifndef FONTDEFAULT
```

14.1.4 Graphics.h

This is file Graphics.h.

Body Source

```
/*****************************************************************
 * Module for Microchip Graphics Library
 * The header file joins all header files used in the graphics library
 * and contains compile options and defaults.
 *****************************************************************
 * FileName:        Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Company:         Microchip Technology, Inc.
 *
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 02/24/11     - Removed GRAPHICS CONTROLLER and DISPLAY PANEL Codes
 *              - Changed USE_CUSTOM to USE_CUSTOM_WIDGET. When custom widgets
 *                are created and users do not want to modify files in the
 *                graphics library.
 *              - Removed the dependency of this file from HardwareProfile.h
 *              - Removed macros that defines supported controllers and
 *                display panels. Refer to Graphics Library Help file
 *                for complete list.
 *              - added #include "gfxcolors.h"
 ********************************************************************/
#ifndef _GRAPHICS_H
    #define _GRAPHICS_H

///////////////////////// GRAPHICS_LIBRARY_VERSION /////////////////////
// MSB is version, LSB is subversion
    #define GRAPHICS_LIBRARY_VERSION    0x0300

//////////////////////////////// INCLUDES ////////////////////////////////
    #include <stdlib.h>             // needed because of malloc()
    #include "GenericTypeDefs.h"
    #include "GraphicsConfig.h"
    #include "DisplayDriver.h"      // Display Driver layer
    #include "Primitive.h"          // Graphic Primitives layer
    #include "GOL.h"                // Graphics Object layer

    #include "ScanCodes.h"          // Scan codes for AT keyboard
    #include "gfxcolors.h"          // default color definitions, can be overidden by
application side
                                    // see gfxcolor.h for details

    #include "Graphics/Palette.h"

    #if defined(USE_BUTTON) || defined(USE_BUTTON_MULTI_LINE)
        #include "Button.h"
    #endif
    #ifdef USE_WINDOW
        #include "Window.h"
    #endif
    #ifdef USE_GROUPBOX
        #include "GroupBox.h"
    #endif
    #ifdef USE_STATICTEXT
        #include "StaticText.h"
    #endif
    #ifdef USE_SLIDER
        #include "Slider.h"
    #endif
    #ifdef USE_CHECKBOX
        #include "CheckBox.h"
    #endif
    #ifdef USE_RADIOBUTTON
        #include "RadioButton.h"
    #endif
    #ifdef USE_PICTURE
        #include "Picture.h"
    #endif
    #ifdef USE_PROGRESSBAR
        #include "ProgressBar.h"
```

```
    #endif
    #ifdef USE_EDITBOX
        #include "EditBox.h"
    #endif
    #ifdef USE_LISTBOX
        #include "ListBox.h"
    #endif
    #ifdef USE_ROUNDDIAL
        #include "RoundDial.h"
    #endif
    #ifdef USE_METER
        #include "Meter.h"
    #endif
    #ifdef USE_GRID
        #include "Grid.h"
    #endif
    #ifdef USE_CHART
        #include "Chart.h"
    #endif
    #ifdef USE_TEXTENTRY
        #include "TextEntry.h"
    #endif
    #ifdef USE_DIGITALMETER
        #include "DigitalMeter.h"
    #endif
    #ifdef USE_ANALOGCLOCK
        #include "AnalogClock.h"
    #endif
    #ifdef USE_CUSTOM_WIDGET
        // Use this in case users wants to create a custom widget and do not want to
        // modify the Graphics Library files.
        #include "CustomControlWidget.h"
    #endif
#endif
```

14.1.5 ScanCodes.h

Macros

Name	Description
SCAN_BS_PRESSED (see page 321)	Back space key pressed.
SCAN_BS_RELEASED (see page 321)	Back space key released.
SCAN_CR_PRESSED (see page 321)	Carriage return pressed.
SCAN_CR_RELEASED (see page 321)	Carriage return released.
SCAN_DEL_PRESSED (see page 322)	Delete key pressed.
SCAN_DEL_RELEASED (see page 322)	Delete key released.
SCAN_DOWN_PRESSED (see page 322)	Down key pressed.
SCAN_DOWN_RELEASED (see page 322)	Down key released.
SCAN_END_PRESSED (see page 323)	End key pressed.
SCAN_END_RELEASED (see page 323)	End key released.
SCAN_HOME_PRESSED (see page 323)	Home key pressed.
SCAN_HOME_RELEASED (see page 323)	Home key released.
SCAN_LEFT_PRESSED (see page 323)	Left key pressed.

SCAN_LEFT_RELEASED (see page 324)	Left key released.
SCAN_PGDOWN_PRESSED (see page 324)	Page down key pressed.
SCAN_PGDOWN_RELEASED (see page 324)	Page down key released.
SCAN_PGUP_PRESSED (see page 324)	Page up key pressed.
SCAN_PGUP_RELEASED (see page 325)	Page up key released.
SCAN_RIGHT_PRESSED (see page 325)	Right key pressed.
SCAN_RIGHT_RELEASED (see page 325)	Right key released.
SCAN_SPACE_PRESSED (see page 325)	Space key pressed.
SCAN_SPACE_RELEASED (see page 325)	Space key released.
SCAN_TAB_PRESSED (see page 326)	Tab key pressed.
SCAN_TAB_RELEASED (see page 326)	Tab key released.
SCAN_UP_PRESSED (see page 326)	Up key pressed.
SCAN_UP_RELEASED (see page 326)	Up key released.

Description

This is file ScanCodes.h.

Body Source

```
/*****************************************************************************
 *
 * Module for Microchip Graphics Library
 * The header file contains scan codes for the standard AT keyboard
 *
 *****************************************************************************
 * FileName:        ScanCodes.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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 SHORTegrated SHORTo 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 07/20/07     ...
 ***************************************************************************/
#ifndef _SCANCODES_H
    #define _SCANCODES_H

/*********************************************************************
* Overview: The following are the defined scan codes for AT keyboard.
*
*********************************************************************/

// Carriage return pressed.
    #define SCAN_CR_PRESSED 0x1C

// Carriage return released.
    #define SCAN_CR_RELEASED    0x9C

// Carriage return alternate pressed.
    #define SCAN_CRA_PRESSED    0x2C

// Carriage return alternate released.
    #define SCAN_CRA_RELEASED   0xAC

// Delete key pressed.
    #define SCAN_DEL_PRESSED    0x53

// Delete key released.
    #define SCAN_DEL_RELEASED   0xD3

// Back space key pressed.
    #define SCAN_BS_PRESSED 0x0E

// Back space key released.
    #define SCAN_BS_RELEASED    0x8E

// Tab key pressed.
    #define SCAN_TAB_PRESSED    0x0F

// Tab key released.
    #define SCAN_TAB_RELEASED   0x8F

// Home key pressed.
    #define SCAN_HOME_PRESSED   0x47

// Home key released.
    #define SCAN_HOME_RELEASED  0xC7

// End key pressed.
    #define SCAN_END_PRESSED    0x4F

// End key released.
    #define SCAN_END_RELEASED   0xCF

// Page up key pressed.
    #define SCAN_PGUP_PRESSED   0x49

// Page up key released.
    #define SCAN_PGUP_RELEASED  0xC9

// Page down key pressed.
    #define SCAN_PGDOWN_PRESSED 0x51

// Page down key released.
    #define SCAN_PGDOWN_RELEASED    0xD1

// Up key pressed.
    #define SCAN_UP_PRESSED 0x48

// Up key released.
```

```
    #define SCAN_UP_RELEASED    0xC8

// Down key pressed.
    #define SCAN_DOWN_PRESSED   0x50

// Down key released.
    #define SCAN_DOWN_RELEASED  0xD0

// Left key pressed.
    #define SCAN_LEFT_PRESSED   0x4B

// Left key released.
    #define SCAN_LEFT_RELEASED  0xCB

// Right key pressed.
    #define SCAN_RIGHT_PRESSED  0x4D

// Right key released.
    #define SCAN_RIGHT_RELEASED 0xCD

// Space key pressed.
    #define SCAN_SPACE_PRESSED  0x39

// Space key released.
    #define SCAN_SPACE_RELEASED 0xB9
#endif // _SCANCODES_H
```

14.1.6 AnalogClock.c

This is file AnalogClock.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Analog Clock
 ******************************************************************************
 * FileName:        AnalogClock.c
 * Dependencies:    AnalogClock.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 Version 3.00, C32
 * Linker:          MPLAB LINK30, 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.
```

```
 ***************************************************************************/
#include "Graphics/Graphics.h"

#if defined(USE_ANALOGCLOCK)

#define THICKNESS_SECOND 4

#define SECOND 1
#define MINUTE 2
#define HOUR 3

/*********************************************************************
* Function: 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)
*
*
* Notes: Creates an ANALOGCLOCK object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*********************************************************************/
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
)
{
    ANALOGCLOCK  *pAc = NULL;
    pAc = (ANALOGCLOCK *)GFX_malloc(sizeof(ANALOGCLOCK));
    if(pAc == NULL)
        return (NULL);

    pAc->hdr.ID = ID;                   // unique id assigned for referencing
    pAc->hdr.pNxtObj = NULL;            // initialize pointer to NULL
    pAc->hdr.type = OBJ_ANALOGCLOCK;    // set object type
    pAc->hdr.left = left;               // left position
    pAc->hdr.top = top;                 // top position
    pAc->hdr.right = left;              // left position
    pAc->hdr.bottom = top;              // top position
    pAc->radius = radius;
    pAc->valueS = 0;
    pAc->prev_valueS = pAc->valueS;
    pAc->valueM = minute;
    pAc->prev_valueM = pAc->valueM-1;
    pAc->valueH = (hour*5) + (minute/12);
    pAc->prev_valueH = pAc->valueH-1;
    pAc->centerx = (left + ((right-left)>>1));
    pAc->centery = (top + ((bottom-top)>>1));
    pAc->pBitmap = pBitmap;      // location of bitmap
    pAc->hdr.state = state;      // state
    pAc->hdr.DrawObj = AcDraw;          // draw function
    pAc->hdr.MsgObj = NULL;//StTranslateMsg;   // message function
    pAc->hdr.MsgDefaultObj = NULL;      // default message function
    pAc->hdr.FreeObj = NULL;

    // Set the color scheme to be used
    if(pScheme == NULL)
        pAc->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pAc->hdr.pGolScheme = (GOL_SCHEME *)pScheme;
```

```
    GOLAddObject((OBJ_HEADER *)pAc);

    return (pAc);
}


/*********************************************************************
* Function: AcMsgDefault(WORD translatedMsg, ANALOGCLOCK *pAc, GOL_MSG* pMsg)
*
*
* Notes: This the default operation to change the state of the clock.
*        Called inside GOLMsg() when GOLMsgCallback() returns a 1.
*
*********************************************************************/
void AcMsgDefault(WORD translatedMsg, ANALOGCLOCK *pAc, GOL_MSG *pMsg)
{

    switch(translatedMsg)
    {
        case AC_MSG_PRESSED:
            SetState(pAc, AC_PRESSED | AC_DRAW);   // set pressed and redraw
            break;

        case AC_MSG_RELEASED:
            ClrState(pAc, AC_PRESSED);              // reset pressed
            SetState(pAc, AC_DRAW);                 // redraw
            break;

        default:

            // catch all for clock messages added by users and
            // behavior defined by users in message callback
            break;
    }
}

/*********************************************************************
* Function: WORD AcTranslateMsg(ANALOGCLOCK *pAc, GOL_MSG *pMsg)
*
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*********************************************************************/
WORD AcTranslateMsg(ANALOGCLOCK *pAc, GOL_MSG *pMsg)
{

    // Evaluate if the message is for the clock
    // Check if disabled first
    if(GetState(pAc, AC_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the clock's face
        if
        (
            (pAc->hdr.left < pMsg->param1) &&
            (pAc->hdr.right > pMsg->param1) &&
            (pAc->hdr.top < pMsg->param2) &&
            (pAc->hdr.bottom > pMsg->param2)
        )
        {

                if(pMsg->uiEvent == EVENT_RELEASE)
                    return (AC_MSG_RELEASED);

                if(!GetState(pAc, AC_PRESSED))
                    return (AC_MSG_PRESSED);
```

```
        }

        return (OBJ_MSG_INVALID);
    }

        #endif

    return (OBJ_MSG_INVALID);
}

/*********************************************************************
* Function: WORD AcDraw(ANALOGCLOCK *pAc)
*
*
* Notes: This is the state machine to draw the clock.
*
*********************************************************************/
WORD AcDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        BEVEL_DRAW,
        TICK
    } AC_DRAW_STATES;

    static AC_DRAW_STATES state = REMOVE;
    static SHORT width, height, radius;

    static GFX_COLOR faceClr, handClr;

    ANALOGCLOCK *pAc;

    pAc = (ANALOGCLOCK *)pObj;


   if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(IsDeviceBusy())
                return (0);

            if(GetState(pAc, AC_HIDE))     // Hide the button (remove from screen)
            {
                SetColor(pAc->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pAc->hdr.left, pAc->hdr.top, pAc->hdr.right, pAc->hdr.bottom))
                {
                    return (0);
                }
            }

            if(GetState(pAc, AC_TICK))
            {
                   state = TICK;
                   return(1); //buthave to do goto erase
                //goto erase_current_pos;
            }

            if(GetState(pAc, UPDATE_HOUR))
            {
                   AcHandsDraw(pAc, HOUR, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
                   state = TICK;
                   return(1); //buthave to do goto erase
                //goto erase_current_pos;
            }

            if(GetState(pAc, UPDATE_MINUTE))
            {
```

```
                AcHandsDraw(pAc, MINUTE, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
                state = TICK;
                return(1); //buthave to do goto erase
            //goto erase_current_pos;
        }

        if(GetState(pAc, UPDATE_SECOND))
        {
                AcHandsDraw(pAc, SECOND, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
                state = TICK;
                return(1); //buthave to do goto erase
            //goto erase_current_pos;
        }



        radius = pAc->radius;    // get radius
        width = (pAc->hdr.right - pAc->hdr.left) - (radius * 2);  // get width
        height = (pAc->hdr.bottom - pAc->hdr.top) - (radius * 2); // get height
        state = BEVEL_DRAW;

    case BEVEL_DRAW:

        faceClr = pAc->hdr.pGolScheme->Color0;
        handClr = pAc->hdr.pGolScheme->Color1;

        SetLineThickness(NORMAL_LINE);
        SetLineType(SOLID_LINE);

        SetColor(faceClr);

        FillCircle(pAc->centerx,pAc->centery, radius);  //Draw Face of Analog Clock

        SetColor(handClr);
        FillCircle(pAc->centerx,pAc->centery, 8);  //Draw Middle of Analog Clock

        AcHandsDraw(pAc, SECOND, THICKNESS_SECOND, handClr, pAc->pBitmap);
        AcHandsDraw(pAc, HOUR, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
        AcHandsDraw(pAc, MINUTE, THICKNESS_SECOND+4, handClr, pAc->pBitmap);

        state = REMOVE;
        break;

    case TICK:

        AcHandsDraw(pAc, SECOND, THICKNESS_SECOND, handClr, pAc->pBitmap);

        if(pAc->valueS++ == 60)
        {
            pAc->valueS = 1;
            if(pAc->valueM++ == 60)
            {
                pAc->valueM = 1;
            }
            if(pAc->valueM%12 == 0)
            {
                pAc->valueH++;
                if(pAc->valueH == 60)
                    pAc->valueH = 0;
                AcHandsDraw(pAc, HOUR, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
            }
            AcHandsDraw(pAc, MINUTE, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
        }


        if( ((pAc->valueS - pAc->valueM)<4 && (pAc->valueS - pAc->valueM)>=0) ||
((pAc->valueS - pAc->valueM)<-56 && (pAc->valueS - pAc->valueM)>-59) )
            AcHandsDraw(pAc, MINUTE, THICKNESS_SECOND+4, handClr, pAc->pBitmap);

        if( ((pAc->valueS - pAc->valueH)<4  && (pAc->valueS - pAc->valueH)>=0) ||
((pAc->valueS - pAc->valueH)<-56 && (pAc->valueS - pAc->valueH)>-59) )
            AcHandsDraw(pAc, HOUR, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
```

```
            if(pAc->valueS-1 == pAc->valueM)
                AcHandsDraw(pAc, MINUTE, THICKNESS_SECOND+4, handClr, pAc->pBitmap);
            if(pAc->valueS-1 == pAc->valueH)
                AcHandsDraw(pAc, HOUR, THICKNESS_SECOND+4, handClr,
pAc->pBitmap);

            state = REMOVE;
            return (1);
    }

    return (1);
}



/* */
/*********************************************************************
* Function: WORD AcHandsDraw(ANALOGCLOCK *pAc, SHORT hand, SHORT thickness, WORD color,
void *pBitmap)
*
*
* Notes: This is the state machine to draw the current hand positions of the clock.
*
*********************************************************************/
WORD AcHandsDraw(ANALOGCLOCK *pAc, SHORT hand, SHORT thickness, WORD color, void *pBitmap)
{
    unsigned int templ=0,tempr, temps;
    SHORT   x1, y1, x2, y2, xi, yi;
    SHORT   x2_prev, y2_prev, xi_prev, yi_prev;
    static SHORT    value=0, prev_value=0;
    SHORT   deltaX, deltaY;
    SHORT   error, stepErrorLT, stepErrorGE;
    SHORT   stepX, stepY;
    SHORT   steep;
    SHORT   temp;
    static SHORT    cnt_thick=0;
    SHORT   radius = pAc->radius;
    BYTE    colorDepth;

    ////////////////////////////////////////////////////////
    register FLASH_WORD *flashAddress = 0;
    register FLASH_WORD *tempFlashAddress;
    static WORD                sizeX=0, sizeY=0;



    tempFlashAddress = (FLASH_WORD *)pBitmap;
    // Move pointer to size information
    if(tempFlashAddress != NULL)
    {
    flashAddress = (FLASH_WORD *)((IMAGE_FLASH *)pBitmap)->address + 1;
    colorDepth = *(flashAddress+1);
    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;
    }

    switch(hand)
    {
        case SECOND:
            value = pAc->valueS;
            prev_value = pAc->prev_valueS;
            pAc->prev_valueS = pAc->valueS;
            radius -=10;
            break;

        case MINUTE:
```

```
            value = pAc->valueM;
            prev_value = pAc->prev_valueM;
            pAc->prev_valueM = pAc->valueM;
            radius -= 3;
            break;

        case HOUR:
            value = pAc->valueH;
            prev_value = pAc->prev_valueH;
            pAc->prev_valueH = pAc->valueH;
            radius -= 30;
            break;

        default:
            break;
    }

    x1=pAc->centerx;
    y1=pAc->centery;

    x2_prev= pAc->centerx;
    y2_prev=pAc->centery;
    x2= pAc->centerx;
    y2=pAc->centery;

    prev_value += 45;
    value += 45;

    //Added for proper orientation
    if(prev_value >= 60)
        prev_value -= 60;

    if(value >= 60)
        value -= 60;

    GetCirclePoint(radius, (prev_value*6),&x2_prev,&y2_prev);
    GetCirclePoint(radius, (value*6),&x2,&y2);

    x2_prev += pAc->centerx;
    y2_prev +=pAc->centery;
    x2      += pAc->centerx;
    y2      +=pAc->centery;

    xi=pAc->hdr.left;
    yi=pAc->hdr.top;
    xi_prev=(((pAc->hdr.right - radius + pAc->hdr.left + radius) - sizeX) >> 1) + 1;
    yi_prev=(((pAc->hdr.bottom - radius + pAc->hdr.top + radius) - sizeY) >> 1) + 1;

    if(value == 15)
        x2 += 1;

    if(prev_value == 15)
        x2_prev += 1;


    if(value == 30)
        y2 += 1;

    if(prev_value == 30)
        y2_prev += 1;

    tempr = (thickness*thickness)>>1;
    temps = ((radius-(11))*(radius-(11)));
    ////////////////////////////////////////////////////////

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    // Ready
        #else
    if(IsDeviceBusy() != 0)
        return (0);
```

```
        #endif


    //#######################################################################################
###
    //Draw the bitmap of area covered by prev hand

    if(x1 == x2_prev)
    {
        if(y1 > y2_prev)
        {
            temp = y1;
            y1 = y2_prev;
            y2_prev = temp;
        }


        for(temp = y1; temp < y2_prev + 1; temp++)   //reDraws at top and bottom
        {

                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {
                    if(prev_value==45)
                    templ = ((x1 + cnt_thick-x2_prev )*(x1 + cnt_thick-x2_prev ) + (temp -
y1 )*(temp - y1 ));
                    else if(prev_value==15)
                    templ = ((x1 + cnt_thick-x2_prev )*(x1 + cnt_thick-x2_prev ) + (temp -
y2_prev )*(temp - y2_prev ));


                    if(templ>tempr && templ<temps)
                    {
                    if(tempFlashAddress != NULL)
                    {SetColor(*(flashAddress + sizeX*(temp-yi_prev)+(x1 + cnt_thick -
xi_prev)));}
                    else SetColor(pAc->hdr.pGolScheme->Color0);
                    PutPixel(x1 + cnt_thick, temp);
                    }
                }
        }

        goto current_hand;
    }

    if(y1 == y2_prev)
    {
        if(x1 > x2_prev)
        {
            temp = x1;
            x1 = x2_prev;
            x2_prev = temp;
        }

        for(temp = x1; temp < x2_prev + 1; temp++)    //redraws at left and right
        {


                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {

                    if(prev_value==30)
                    templ = ((temp-x1 )*(temp-x1 ) + (y1 + cnt_thick - y2_prev )*(y1 +
cnt_thick - y2_prev ));
                    else
                    templ = ((temp-x2_prev )*(temp-x2_prev ) + (y1 + cnt_thick - y2_prev
)*(y1 + cnt_thick - y2_prev ));


                    if(templ>tempr && templ<temps)
                    {
                    if(tempFlashAddress != NULL)
```

```
                        {SetColor(*(flashAddress + sizeX*(y1 + cnt_thick -yi_prev)+(temp -
xi_prev)));}
                        else SetColor(pAc->hdr.pGolScheme->Color0);
                        PutPixel(temp, y1 + cnt_thick);
                        }
                    }

            }

            goto current_hand;
        }

        stepX = 0;
        deltaX = x2_prev - x1;
        if(deltaX < 0)
        {
            deltaX = -deltaX;
            --stepX;
        }
        else
        {
            ++stepX;
        }

        stepY = 0;
        deltaY = y2_prev - y1;
        if(deltaY < 0)
        {
            deltaY = -deltaY;
            --stepY;
        }
        else
        {
            ++stepY;
        }

        steep = 0;
        if(deltaX < deltaY)
        {
            ++steep;
            temp = deltaX;
            deltaX = deltaY;
            deltaY = temp;
            temp = x1;
            x1 = y1;
            y1 = temp;
            temp = stepX;
            stepX = stepY;
            stepY = temp;
            PutPixel(y1, x1);
        }
        else
        {
            PutPixel(x1, y1);
        }

        // If the current error greater or equal zero
        stepErrorGE = deltaX << 1;

        // If the current error less than zero
        stepErrorLT = deltaY << 1;

        // Error for the first pixel
        error = stepErrorLT - deltaX;

        while(--deltaX >= 0)
        {
            if(error >= 0)
            {
                y1 += stepY;
                error -= stepErrorGE;
            }
```

```
        x1 += stepX;
        error += stepErrorLT;


            if(steep)
            {
                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {
                    templ = ((y1 + cnt_thick-x2_prev) *(y1 + cnt_thick-x2_prev) + (x1 -
y2_prev)*(x1 - y2_prev));

                    if(templ>tempr && templ<temps)
                    {

                    if(tempFlashAddress != NULL)
                    {SetColor(*(flashAddress + sizeX*(x1 -yi_prev)+(y1 + cnt_thick -
xi_prev)));}
                    else SetColor(pAc->hdr.pGolScheme->Color0);

                    PutPixel(y1 + cnt_thick,x1);
                    }
                }

            }
            else
            {

                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {
                    templ = ((x1-x2_prev)*(x1-x2_prev) + (y1 + cnt_thick - y2_prev)*(y1 +
cnt_thick - y2_prev));

                    if(templ>tempr && templ<temps)
                    {
                        if(tempFlashAddress != NULL){
                        SetColor(*(flashAddress + sizeX*(y1 + cnt_thick -yi_prev)+(x1 -
xi_prev)));
                        }
                        else SetColor(pAc->hdr.pGolScheme->Color0);

                        PutPixel(x1,y1 + cnt_thick);
                    }
                }

            }
    }
    //#######################################################################################
###

    //Draw the current hand
current_hand:
    SetColor(color);

            x1=pAc->centerx;
            y1=pAc->centery;


    if(x1 == x2)
    {
        if(y1 > y2)
        {
            temp = y1;
            y1 = y2;
            y2 = temp;
        }

        for(temp = y1; temp < y2 + 1; temp++)
        {

                    if(value==45)
                    templ = ((x1 + cnt_thick-x2)*(x1 + cnt_thick-x2) + (temp - y1)*(temp -
```

```
y1));
                    else if(value==15)
                    templ = ((x1 + cnt_thick-x2)*(x1 + cnt_thick-x2) + (temp - y2)*(temp -
y2));


                        if(templ>tempr && templ<temps)
                    for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                    {
                        PutPixel(x1 + cnt_thick, temp);
                    }

        }

        return (1);
    }

    if(y1 == y2)
    {
        if(x1 > x2)
        {
            temp = x1;
            x1 = x2;
            x2 = temp;
        }

        for(temp = x1; temp < x2 + 1; temp++)
        {

                    if(value==30)
                    templ = ((temp-x1)*(temp-x1) + (y1 + cnt_thick - y2)*(y1 + cnt_thick -
y2));
                    else// if(value==0)
                    templ = ((temp-x2)*(temp-x2) + (y1 + cnt_thick - y2)*(y1 + cnt_thick -
y2));


                    if(templ>tempr && templ<temps)

                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {
                    PutPixel(temp, y1 + cnt_thick);
                }
        }

        return (1);
    }

    stepX = 0;
    deltaX = x2 - x1;
    if(deltaX < 0)
    {
        deltaX = -deltaX;
        --stepX;
    }
    else
    {
        ++stepX;
    }

    stepY = 0;
    deltaY = y2 - y1;
    if(deltaY < 0)
    {
        deltaY = -deltaY;
        --stepY;
    }
    else
    {
        ++stepY;
    }
```

```
    steep = 0;
    if(deltaX < deltaY)
    {
        ++steep;
        temp = deltaX;
        deltaX = deltaY;
        deltaY = temp;
        temp = x1;
        x1 = y1;
        y1 = temp;
        temp = stepX;
        stepX = stepY;
        stepY = temp;
        PutPixel(y1, x1);
    }
    else
    {
        PutPixel(x1, y1);
    }

    // If the current error greater or equal zero
    stepErrorGE = deltaX << 1;

    // If the current error less than zero
    stepErrorLT = deltaY << 1;

    // Error for the first pixel
    error = stepErrorLT - deltaX;


    while(--deltaX >= 0)
    {
        if(error >= 0)
        {
            y1 += stepY;
            error -= stepErrorGE;
        }

        x1 += stepX;
        error += stepErrorLT;

            if(steep)
            {
                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {

                    templ = ((y1 + cnt_thick-x2) *(y1 + cnt_thick-x2) + (x1 - y2)*(x1 -
y2));

                    if(templ>tempr && templ<temps)
                    PutPixel(y1 + cnt_thick,x1);

                }
            }
            else
            {
                for(cnt_thick=-thickness>>2;cnt_thick<thickness>>2;cnt_thick++)
                {
                    templ = ((x1-x2)*(x1-x2) + (y1 + cnt_thick - y2)*(y1 + cnt_thick - y2));

                    if(templ>tempr && templ<temps)
                    PutPixel(x1,y1 + cnt_thick);
                }
            }
    }   // end of while

    return (1);
}

/*****************************************************************
* Function: AcSetHour(STATICTEXT *pAc, XCHAR *pText)
*
```

```
* Notes: Sets the string that will be used.
*
*****************************************************************/
void AcSetHour(ANALOGCLOCK *pAc, SHORT hour)
{

     pAc->valueH = (hour*5) + (pAc->valueM/12);

}

/*****************************************************************
* Function: AcSetMinute(STATICTEXT *pSt, XCHAR *pText)
*
* Notes: Sets the string that will be used.
*
*****************************************************************/
void AcSetSecond(ANALOGCLOCK *pAc, SHORT second)
{

     pAc->valueS = second;

}

/*****************************************************************
* Function: AcSetMinute(STATICTEXT *pSt, XCHAR *pText)
*
* Notes: Sets the string that will be used.
*
*****************************************************************/
void AcSetMinute(ANALOGCLOCK *pAc, SHORT minute)
{

     pAc->valueM = minute;

}

#endif
```

14.1.7 AnalogClock.h

Functions

	Name	Description
	AcCreate (see page 56)	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 (see page 58)	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.
	AcHandsDraw (see page 59)	Draws the current position of the clock hands with the given thickness and color.
	AcMsgDefault (see page 61)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	AcSetHour (see page 60)	Sets the hour of the analog clock.
	AcSetMinute (see page 60)	Sets the minute of the analog clock.
	AcSetSecond (see page 61)	Sets the second of the analog clock.
	AcTranslateMsg (see page 62)	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
AC_DISABLED (see page 55)	Bit for disabled state.
AC_DRAW (see page 55)	Bit to indicate button must be redrawn.
AC_HIDE (see page 55)	Bit to indicate button must be removed from screen.
AC_PRESSED (see page 55)	Bit for press state.
AC_TICK (see page 56)	Bit to tick second hand
UPDATE_HOUR (see page 56)	Bit to indicate hour hand must be redrawn
UPDATE_MINUTE (see page 56)	Bit to indicate minute hand must be redrawn
UPDATE_SECOND (see page 56)	Bit to indicate minute hand must be redrawn

Structures

Name	Description
ANALOGCLOCK (see page 64)	Defines the parameters required for a clock Object. The following relationships of the parameters determines the general shape of the button: 1. Width (see page 448) is determined by right - left. 2. Height (see page 447) 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*radius = height = width, the button is a circular button.

Description

This is file AnalogClock.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Analog Clock
 *****************************************************************************
 * FileName:        AnalogClock.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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.
```

```
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ***************************************************************************/
#ifndef _ANALOGCLOCK_H
    #define _ANALOGCLOCK_H

    #include <Graphics\GOL.h>
    #include "GenericTypeDefs.h"
/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define AC_DISABLED    0x0002   // Bit for disabled state.
    #define AC_PRESSED     0x0004   // Bit for press state.
    #define AC_TICK        0x1000   // Bit to tick second hand
    #define UPDATE_HOUR    0x2000   // Bit to indicate hour hand must be redrawn
    #define UPDATE_MINUTE  0x0100   // Bit to indicate minute hand must be redrawn
    #define UPDATE_SECOND  0x0200   // Bit to indicate minute hand must be redrawn
    #define AC_DRAW        0x4000   // Bit to indicate button must be redrawn.
    #define AC_HIDE        0x8000   // Bit to indicate button must be removed from screen.
    #define AC_REMOVE      0x8000

/*********************************************************************
* Overview: Defines the parameters required for a clock 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*radius = height = width, the button is a circular button.
*
*********************************************************************/
typedef struct
{
    OBJ_HEADER    hdr;        // Generic header for all Objects (see OBJ_HEADER).
    SHORT         radius;     // Radius of the clock.
    SHORT       centerx;
    SHORT       centery;
    SHORT       valueS;
    SHORT       prev_valueS;
    SHORT       valueM;
    SHORT       prev_valueM;
    SHORT       valueH;
    SHORT       prev_valueH;
    void        *pBitmap;   // Pointer to bitmap used.
} ANALOGCLOCK;


/*********************************************************************
* Function: ANALOGCLOCK *AcCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                               SHORT bottom, SHORT radius, void *pBitmap, XCHAR *pText,
*                               GOL_SCHEME *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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the object.
*        top - Top most position of the object.
*      right - Right most position of the object.
*      bottom - Bottom most position of the object.
*        hour - set the current hour.
*        minute - set the current minute.
*        radius - radius of the clock.
*        sechand - Flag to draw cecond hand or not.
*        state - Sets the initial state of the object.
```

```
*        pBitmap - Pointer to the bitmap used on the face of the button
*                  dimension of the bitmap must match the dimension of the
*                  button.
*        pText - Pointer to the text of the button.
*        pScheme - Pointer to the style scheme used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   GOL_SCHEME *pScheme;
*   WORD state;
*
*       pScheme = GOLCreateScheme();
*       state = AC_DRAW;
*
*       AnalogClock = AcCreate(1,20,64,50,118,0, state, NULL, "ON", pScheme);

* Side Effects: none
*
*********************************************************************/
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
);

/*********************************************************************
* Function: AcTranslateMsg(BUTTON *pB, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       Translated Message     Input Source     Set/Clear State Bit         Description
*       ##################     ############     #######################
##########################################################################################
#######################################################################
*       AC_MSG_PRESSED        Touch Screen     EVENT_PRESS, EVENT_MOVE
*                              Keyboard         EVENT_KEYSCAN
*       AC_MSG_STILLPRESSED   Touch Screen     EVENT_STILLPRESS
*       AC_MSG_RELEASED    Touch Screen
EVENT_RELEASE
*                              Keyboard
EVENT_KEYSCAN
*       AC_MSG_CANCELPRESS     Touch Screen
EVENT_MOVE
*       AC_MSG_INVALID         Any
Any
*   </TABLE>
*
* PreCondition: none
*
* Input: pAc    - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: Returns the translated message depending on the received GOL message:
*         - AC_MSG_PRESSED – Clock is pressed
```

```
*          - AC_MSG_RELEASED – Clock is released
*          - AC_MSG_CANCELPRESS – Clock will be released, user cancels press action on the
clock
*          - OBJ_MSG_INVALID – Clock is not affected
*
* Example:
*   <CODE>
*   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)){
*               switch(pCurrentObj->type){
*                   case OBJ_ANALOGCLOCK:
*                       objMsg = AcTranslateMsg((ANALOGCLOCK*)pCurrentObj, pMsg);
*                       if(objMsg == OBJ_MSG_INVALID)
*                           break;
*                       if(GOLMsgCallback(objMsg,pCurrentObj,pMsg))
*                           AcMsgDefault(objMsg,(ANALOGCLOCK*)pCurrentObj);
*                       break;
*                   case OBJ_SLIDER:
*                       objMsg = SldTranslateMsg((SLIDER*)pCurrentObj, pMsg);
*                       if(objMsg == OBJ_MSG_INVALID)
*                           break;
*                       if(GOLMsgCallback(objMsg,pCurrentObj,pMsg))
*                           SldMsgDefault(objMsg,(SLIDER*)pCurrentObj);
*                       break;
*                   default: break;
*               }
*           }
*       }
*       pCurrentObj = pCurrentObj->pNxtObj;
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD    AcTranslateMsg(ANALOGCLOCK *pAc, GOL_MSG *pMsg);

/*********************************************************************
* Function: AcMsgDefault(WORD translatedMsg, ANALOGCLOCK *pAc, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       Translated Message   Input Source   Set/Clear State Bit     Description
*       ##################   #############   ######                  ############
*       AC_MSG_PRESSED      Touch Screen,   Set AC_PRESSED
*                             Keyboard
*       AC_MSG_RELEASED  Touch Screen,  Clear AC_PRESSED
*                             Keyboard
*       AC_MSG_CANCELPRESS   Touch Screen,  Clear AC_PRESSED
*
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message.
*        pAc            - The pointer to the object whose state will be modified.
*        pMsg          - The pointer to the GOL message.
*
* Output: none
```

```
 *
 * Example:
 *   See BtnTranslateMsg() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
void    AcMsgDefault(WORD translatedMsg, ANALOGCLOCK *pAc, GOL_MSG *pMsg);

/********************************************************************
 * Function: WORD AcDraw(ANALOGCLOCK *pB)
 *
 * 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.
 *
 *
 * PreCondition: Object must be created before this function is called.
 *
 * Input: pAc - Pointer to the object to be rendered.
 *
 * Output: 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.
 *
 * Example:
 *   <CODE>
 *   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;
 *       }
 *   }
```

```
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD    AcDraw(void *pObj);

/*********************************************************************
* Function: AcHandsDraw(ANALOGCLOCK *pAc, SHORT hand, SHORT thickness, WORD color, void
*pBitmap);
*
* Overview: Draws the current position of the clock hands with the given thickness and
color.
*
* PreCondition: none
*
* Input: pAc - The pointer to the object whose hands will be modified.
*        hand - which hand to be drawn (second, minute, hour)
*        thickness - thickness to draw the hand
*        color - color to draw the hand
*        *pBitmap - bitmap background to be redrawn
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
WORD AcHandsDraw(ANALOGCLOCK *pAc, SHORT hand, SHORT thickness, WORD color, void *pBitmap);


/*********************************************************************
* Function: AcSetHour(ANALOGCLOCK *pAc, SHORT hour)
*
* Overview: Sets the hour of the analog clock.
*
* PreCondition: none
*
* Input: pAc - The pointer to the object whose hands will be modified.
*        hour - current hour of the analog clock
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void AcSetHour(ANALOGCLOCK *pAc, SHORT hour);

/*********************************************************************
* Function: AcSetMinute(ANALOGCLOCK *pAc, SHORT minute)
*
* Overview: Sets the minute of the analog clock.
*
* PreCondition: none
*
* Input: pAc - The pointer to the object whose hands will be modified.
*        minute - current minute of the analog clock
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void AcSetMinute(ANALOGCLOCK *pAc, SHORT minute);

/*********************************************************************
* Function: AcSetMinute(ANALOGCLCOK *pAc, SHORT second)
*
* Overview: Sets the second of the analog clock.
*
* PreCondition: none
*
* Input: pAc -    The pointer to the object whose hands will be modified.
```

```
*        second - current minute of the analog clock
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void AcSetSecond(ANALOGCLOCK *pAc, SHORT second);

#endif // _BUTTON_H
```

14.1.8 Button.c

This is file Button.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Button
 *****************************************************************************
 * FileName:        Button.c
 * Dependencies:    Button.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Linker:          MPLAB LINK30, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 06/26/09     Added message ID BTN_MSG_STILLPRESSED
 *              to signify that a continuous touch on
 *              the button through touch screen.
 * 06/29/09     Added multi-line text support on buttons
 *              must set USE_BUTTON_MULTI_LINE in
 *              GraphicsConfig.h file.
 * 10/04/10     Added BTN_NOPANEL property state bit to make the buttons with
 *              bitmaps that is of the same size or larger than the button
 *              dimension to skip drawing of the panel.
 * 04/20/11     Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 *****************************************************************************/
#include "Graphics/Graphics.h"
```

```
#if defined(USE_BUTTON) || defined(USE_BUTTON_MULTI_LINE)

/*********************************************************************
* Function: BUTTON  *BtnCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                              SHORT bottom, SHORT radius, void *pBitmap, XCHAR *pText,
*                              GOL_SCHEME *pScheme)
*
*
* Notes: Creates a BUTTON object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*********************************************************************/
BUTTON *BtnCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    SHORT       radius,
    WORD        state,
    void        *pBitmap,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    BUTTON  *pB = NULL;
    pB = (BUTTON *)GFX_malloc(sizeof(BUTTON));
    if(pB == NULL)
        return (NULL);

    pB->hdr.ID = ID;                        // unique id assigned for referencing
    pB->hdr.pNxtObj = NULL;                 // initialize pointer to NULL
    pB->hdr.type = OBJ_BUTTON;              // set object type
    pB->hdr.left = left;                    // left position
    pB->hdr.top = top;                      // top position
    pB->hdr.right = right;                  // right position
    pB->hdr.bottom = bottom;                // bottom position
    pB->radius = radius;                    // radius
    pB->pBitmap = pBitmap;                  // location of bitmap
    pB->pText = pText;                      // location of the text
    pB->hdr.state = state;                  // state
    pB->hdr.DrawObj = BtnDraw;              // draw function
    pB->hdr.MsgObj = BtnTranslateMsg;       // message function
    pB->hdr.MsgDefaultObj = BtnMsgDefault;  // default message function
    pB->hdr.FreeObj = NULL;                 // free function

    // Set the color scheme to be used
    if(pScheme == NULL)
        pB->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pB->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pB->textWidth = 0;
    pB->textHeight = 0;
    if(pB->pText != NULL)
    {
        BtnSetText(pB, pText);
    }

    #ifdef USE_ALPHABLEND
    if(pB->hdr.pGolScheme->AlphaValue > 0)
            CopyPageWindow(_GFXDestinationPage,
                        _GFXBackgroundPage,
                        pB->hdr.left, pB->hdr.top,pB->hdr.left, pB->hdr.top,
                        pB->hdr.right - pB->hdr.left,
                        pB->hdr.bottom - pB->hdr.top);
    #endif

    GOLAddObject((OBJ_HEADER *)pB);
```

```
        #ifdef USE_FOCUS
    if(GetState(pB, BTN_FOCUSED))
        GOLSetFocus((OBJ_HEADER *)pB);
        #endif
    return (pB);
}

/*********************************************************************
* Function: BtnSetText(BUTTON *pB, XCHAR *pText)
*
*
* Notes: Sets the text used in the button.
*
*********************************************************************/
void BtnSetText(BUTTON *pB, XCHAR *pText)
{
    #ifdef USE_BUTTON_MULTI_LINE
        int     width = 0, chCtr = 0, lineCtr = 1;
        XCHAR   ch, *pParser;
    #endif

    pB->pText = pText;

    #ifdef USE_BUTTON_MULTI_LINE

        // calculate width and height taking into account the multiple lines of text
        pParser = pB->pText;
        ch = *pText;

        // calculate the width (taken from the longest line)
        while(1)
        {
            if((ch == 0x000A) || (ch == 0x0000))
            {
                if(width < GetTextWidth(pParser, pB->hdr.pGolScheme->pFont))
                {
                    width = GetTextWidth(pParser, pB->hdr.pGolScheme->pFont);
                }

                if(ch == 0x000A)
                {
                    pParser = pText + chCtr + 1;
                    lineCtr++;
                }
                else
                {
                    break;
                }
            }

            chCtr++;
            ch = *(pText + chCtr);
        }

        pB->textWidth = width;
        pB->textHeight = GetTextHeight(pB->hdr.pGolScheme->pFont) * lineCtr;
    #else
        pB->textWidth = GetTextWidth(pText, pB->hdr.pGolScheme->pFont);
        pB->textHeight = GetTextHeight(pB->hdr.pGolScheme->pFont);
    #endif // #ifdef USE_BUTTON_MULTI_LINE
}

/*********************************************************************
* Function: BtnMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
*
* Notes: This the default operation to change the state of the button.
*        Called inside GOLMsg() when GOLMsgCallback() returns a 1.
*
*********************************************************************/
void BtnMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
```

```
{

    BUTTON *pB;

    pB = (BUTTON *)pObj;

        #ifdef USE_FOCUS
            #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
        if(!GetState(pB, BTN_FOCUSED))
        {
            GOLSetFocus((OBJ_HEADER *)pB);
        }
    }

            #endif
        #endif
    switch(translatedMsg)
    {
        case BTN_MSG_PRESSED:
            SetState(pB, BTN_PRESSED | BTN_DRAW);   // set pressed and redraw
            break;

        case BTN_MSG_RELEASED:
        case BTN_MSG_CANCELPRESS:
            ClrState(pB, BTN_PRESSED);              // reset pressed
            SetState(pB, BTN_DRAW);                 // redraw
            break;

        default:

            // catch all for button messages added by users and
            // behavior defined by users in message callback
            break;
    }
}

/*****************************************************************
* Function: WORD BtnTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*****************************************************************/
WORD BtnTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    BUTTON *pB;

    pB = (BUTTON *)pObj;


    // Evaluate if the message is for the button
    // Check if disabled first
    if(GetState(pB, BTN_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the button's face
        if
        (
            (pB->hdr.left < pMsg->param1) &&
            (pB->hdr.right > pMsg->param1) &&
            (pB->hdr.top < pMsg->param2) &&
            (pB->hdr.bottom > pMsg->param2)
        )
        {
            if(GetState(pB, BTN_TOGGLE))
```

```
            {
                if(pMsg->uiEvent == EVENT_RELEASE)
                {
                    if(GetState(pB, BTN_PRESSED))
                        return (BTN_MSG_RELEASED);
                    else
                        return (BTN_MSG_PRESSED);
                }
            }
            else
            {
                if(pMsg->uiEvent == EVENT_RELEASE)
                    return (BTN_MSG_RELEASED);
                if(pMsg->uiEvent == EVENT_STILLPRESS)
                {
                    if(GetState(pB, BTN_PRESSED))
                        return (BTN_MSG_STILLPRESSED);
                }

                if(!GetState(pB, BTN_PRESSED))
                    return (BTN_MSG_PRESSED);
            }
        }
        else
        {
            if(!GetState(pB, BTN_TOGGLE))
            {
                if((pMsg->uiEvent == EVENT_MOVE) && (GetState(pB, BTN_PRESSED)))
                    return (BTN_MSG_CANCELPRESS);
            }
        }

        return (OBJ_MSG_INVALID);
    }

        #endif
        #ifdef USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD)
    {
        if((WORD)pMsg->param1 == pB->hdr.ID)
        {
            if(pMsg->uiEvent == EVENT_KEYSCAN)
            {
                if(GetState(pB, BTN_TOGGLE))
                {
                    if((pMsg->param2 == SCAN_SPACE_RELEASED) || (pMsg->param2 ==
SCAN_CR_RELEASED))
                    {
                        if(GetState(pB, BTN_PRESSED))
                            return (BTN_MSG_RELEASED);
                        else
                            return (BTN_MSG_PRESSED);
                    }
                }
                else
                {
                    if((pMsg->param2 == SCAN_SPACE_PRESSED) || (pMsg->param2 ==
SCAN_CR_PRESSED))
                    {
                        return (BTN_MSG_PRESSED);
                    }

                    if((pMsg->param2 == SCAN_SPACE_RELEASED) || (pMsg->param2 ==
SCAN_CR_RELEASED))
                    {
                        return (BTN_MSG_RELEASED);
                    }
                }

                if((pMsg->param2 == SCAN_SPACE_PRESSED) || (pMsg->param2 ==
SCAN_CRA_PRESSED))
                {
```

```
                    return (BTN_MSG_PRESSED);
                }

                if((pMsg->param2 == SCAN_SPACE_RELEASED) || (pMsg->param2 ==
SCAN_CRA_RELEASED))
                {
                    return (BTN_MSG_RELEASED);
                }
            }
        }

        return (OBJ_MSG_INVALID);
    }

        #endif
    return (OBJ_MSG_INVALID);
}
/*********************************************************************
* Button draw states
*********************************************************************/
typedef enum
{
    REMOVE,
    RNDBUTTON_DRAW,
    #ifdef USE_BUTTON_MULTI_LINE
    CHECK_TEXT_DRAW,
    #else
    TEXT_DRAW,
    #endif
    TEXT_DRAW_RUN,
    FOCUS_DRAW,
} BTN_DRAW_STATES;
/*********************************************************************
* Function: inline WORD __attribute__((always_inline)) DrawButtonFocus(BUTTON *button,
SHORT radius, BTN_DRAW_STATES *current_state)
*********************************************************************/
inline WORD __attribute__((always_inline)) DrawButtonFocus(BUTTON *button, SHORT radius,
BTN_DRAW_STATES *current_state)
{

    *current_state = FOCUS_DRAW;

    if(IsDeviceBusy())
        return (0);

    if(GetState(button, BTN_FOCUSED))
    {
        SetLineType(FOCUS_LINE);
        if(GetState(button, BTN_PRESSED))
        {
            SetColor(button->hdr.pGolScheme->TextColor1);
        }
        else
        {
            SetColor(button->hdr.pGolScheme->TextColor0);
        }

        // check if the object has rounded corners or not
        if(!button->radius)
        {
            if
            (
                !Rectangle
                    (
                        button->hdr.left + GOL_EMBOSS_SIZE + 2,
                        button->hdr.top + GOL_EMBOSS_SIZE + 2,
                        button->hdr.right - GOL_EMBOSS_SIZE - 2,
                        button->hdr.bottom - GOL_EMBOSS_SIZE - 2
                    )
            )
            {
                return (0);
```

```
            }
        }
        else
        {

            // original center is still the same, but radius is reduced
            if
            (
                !Bevel
                    (
                        button->hdr.left + radius,
                        button->hdr.top + radius,
                        button->hdr.right - radius,
                        button->hdr.bottom - radius,
                        radius - 2 - GOL_EMBOSS_SIZE
                    )
            )
            {
                return (0);
            }
        }

        SetLineType(SOLID_LINE);
    }

    #ifdef USE_ALPHABLEND
    if(button->hdr.pGolScheme->AlphaValue > 0)
    {
        AlphaBlendWindow(GFXGetPageXYAddress(_GFXForegroundPage, button->hdr.left,
button->hdr.top),
                         GFXGetPageXYAddress(_GFXBackgroundPage, button->hdr.left,
button->hdr.top),
                         GFXGetPageXYAddress(_GFXDestinationPage, button->hdr.left,
button->hdr.top),
                         button->hdr.right - button->hdr.left,
                         button->hdr.bottom - button->hdr.top,
                         button->hdr.pGolScheme->AlphaValue);
        SetActivePage(_GFXDestinationPage);

    }
    #endif

    *current_state = REMOVE;
    return 1;
}
/*********************************************************************
* Function: WORD BtnDraw(void *pObj)
*********************************************************************/
#ifdef USE_BUTTON_MULTI_LINE
inline void __attribute__((always_inline)) SetButtonTextPosition(BUTTON *button, XCHAR
*pCurLine, SHORT lineCtr)
{
    WORD xText, yText;
    SHORT textWidth;

    SetFont(button->hdr.pGolScheme->pFont);
    textWidth = GetTextWidth(pCurLine, button->hdr.pGolScheme->pFont);

    // check text alignment
    if(GetState(button, BTN_TEXTRIGHT))
    {
        xText = button->hdr.right - (textWidth + GOL_EMBOSS_SIZE + 2);
    }
    else if(GetState(button, BTN_TEXTLEFT))
    {
        xText = button->hdr.left + GOL_EMBOSS_SIZE + 2;
    }
    else
    {

        // centered text in x direction
        xText = (button->hdr.left + button->hdr.right - textWidth) >> 1;
```

```
    }

    if(GetState(button, BTN_TEXTTOP))
    {
        yText = button->hdr.top + GOL_EMBOSS_SIZE + (lineCtr *
GetTextHeight(button->hdr.pGolScheme->pFont));
    }
    else if(GetState(button, BTN_TEXTBOTTOM))
    {
        yText = button->hdr.bottom - (GOL_EMBOSS_SIZE + button->textHeight) + (lineCtr *
GetTextHeight(button->hdr.pGolScheme->pFont));
    }
    else
    {

        // centered text in y direction
        yText = ((button->hdr.bottom + button->hdr.top - button->textHeight) >> 1) +
(lineCtr * GetTextHeight(button->hdr.pGolScheme->pFont));
    }

    MoveTo(xText, yText);

}
#endif
/*****************************************************************************
* Function: WORD BtnDraw(void *pObj)
*
*
* Notes: This is the state machine to draw the button.
*
*****************************************************************************/
WORD BtnDraw(void *pObj)
{

    static BTN_DRAW_STATES state = REMOVE;
    static SHORT width, height, radius;

    #ifdef USE_BUTTON_MULTI_LINE
    static SHORT charCtr = 0, lineCtr = 0;
    static XCHAR *pCurLine = NULL;
    XCHAR ch = 0;
    #else
    WORD xText, yText;
    #endif
    WORD faceClr, embossLtClr, embossDkClr;
    BUTTON *pB;

    pB = (BUTTON *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(IsDeviceBusy())
                return (0);

            if(GetState(pB, BTN_HIDE))
            {                       // Hide the button (remove from screen)

                #ifdef USE_ALPHABLEND
                if(pB->hdr.pGolScheme->AlphaValue > 0)
                {

                    CopyPageWindow(_GFXBackgroundPage,
                                   _GFXDestinationPage,
                                   pB->hdr.left, pB->hdr.top,pB->hdr.left, pB->hdr.top,
                                   pB->hdr.right - pB->hdr.left,
                                   pB->hdr.bottom - pB->hdr.top);
                }
                else
```

```
                #endif

                SetColor(pB->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pB->hdr.left, pB->hdr.top, pB->hdr.right, pB->hdr.bottom))
                {
                    return (0);
                }

                return (1);
            }

            /* Note: that width and height adjustment considers the following assumptions:
                    1. if circular width = height = radius*2
                    2. if vertical capsule width = radius*2
                    3. if horizontal capsule height = radius*2
                    4. radius must be less than or equal to width if height is greater than
width

                    5. radius must be less than or equal to height if width is greater than
height
                    6. if button is cornered, radius must be zero
            */
            radius = pB->radius;    // get radius
            width = (pB->hdr.right - pB->hdr.left) - (radius * 2);  // get width
            height = (pB->hdr.bottom - pB->hdr.top) - (radius * 2); // get height

            if(!GetState(pB, BTN_NOPANEL))
            {
                if(!GetState(pB, BTN_DISABLED))
                {
                    if(GetState(pB, BTN_PRESSED))
                    {
                        embossDkClr = pB->hdr.pGolScheme->EmbossLtColor;
                        embossLtClr = pB->hdr.pGolScheme->EmbossDkColor;
                        faceClr = pB->hdr.pGolScheme->Color1;
                    }
                    else
                    {
                        embossLtClr = pB->hdr.pGolScheme->EmbossLtColor;
                        embossDkClr = pB->hdr.pGolScheme->EmbossDkColor;
                        faceClr = pB->hdr.pGolScheme->Color0;
                    }
                }
                else
                {
                    embossLtClr = pB->hdr.pGolScheme->EmbossLtColor;
                    embossDkClr = pB->hdr.pGolScheme->EmbossDkColor;
                    faceClr = pB->hdr.pGolScheme->ColorDisabled;
                }

                #ifdef USE_ALPHABLEND
                if(pB->hdr.pGolScheme->AlphaValue > 0)
                {


                    CopyPageWindow(_GFXBackgroundPage,
                               _GFXForegroundPage,
                               pB->hdr.left, pB->hdr.top,pB->hdr.left, pB->hdr.top,
                               pB->hdr.right - pB->hdr.left,
                               pB->hdr.bottom - pB->hdr.top);

                    SetActivePage(_GFXForegroundPage);
                }
                #endif

                SetLineThickness(NORMAL_LINE);
                SetLineType(SOLID_LINE);

                #ifdef USE_GRADIENT
                GOLGradientPanelDraw(pB->hdr.pGolScheme);
                #endif

                GOLPanelDraw
```

```
                (
                    pB->hdr.left + radius,
                    pB->hdr.top + radius,
                    pB->hdr.right - radius,
                    pB->hdr.bottom - radius,
                    radius,
                    faceClr,
                    embossLtClr,
                    embossDkClr,
                    pB->pBitmap,
                    GOL_EMBOSS_SIZE
                );
            }
            state = RNDBUTTON_DRAW;

        case RNDBUTTON_DRAW:
            if (GetState(pB, BTN_NOPANEL))
            {
                // check if there is an image to be drawn
                if (pB->pBitmap != NULL)
                {
                    if
                    (
                        !PutImage
                            (
                                ((pB->hdr.right + pB->hdr.left - GetImageWidth((void
*)pB->pBitmap)) >> 1) + 1,
                                ((pB->hdr.top + pB->hdr.bottom - GetImageHeight((void
*)pB->pBitmap)) >> 1) + 1,
                                pB->pBitmap,IMAGE_NORMAL
                            )
                    )
                    {
                        return (0);
                    }
                }
            }
            else
            {
                if (GetState(pB, BTN_TWOTONE))
                {
                    if(!GOLTwoTonePanelDrawTsk())
                    {
                        return (0);
                    }
                }
                else
                {
                    if(!GOLPanelDrawTsk())
                    {
                        return (0);
                    }
                }
            }

            #ifdef USE_BUTTON_MULTI_LINE
            state = CHECK_TEXT_DRAW;
            #else
            state = TEXT_DRAW;
            #endif

        #ifdef USE_BUTTON_MULTI_LINE
        case CHECK_TEXT_DRAW:
            if(pB->pText != NULL)
            {
                if(!GetState(pB, BTN_DISABLED))
                {
                    if(GetState(pB, BTN_PRESSED))
                    {
                        SetColor(pB->hdr.pGolScheme->TextColor1);
                    }
                    else
```

```
                    {
                        SetColor(pB->hdr.pGolScheme->TextColor0);
                    }
                }
                else
                {
                    SetColor(pB->hdr.pGolScheme->TextColorDisabled);
                }

                pCurLine = pB->pText;
                lineCtr = 0;
                charCtr = 0;
                SetButtonTextPosition(pB, pCurLine, lineCtr);
                state = TEXT_DRAW_RUN;
            }
            else
            {
                return DrawButtonFocus(pB, radius, &state);
            }

        case TEXT_DRAW_RUN:
            ch = *(pCurLine + charCtr);

            // output one character at time until a newline character or a NULL character
is sampled
            while(0x0000 != ch)
            {
                if(!OutChar(ch))
                    return (0);
                // render the character
                charCtr++;                          // update to next character
                ch = *(pCurLine + charCtr);

                if(ch == 0x000A)
                {                                       // new line character
                    pCurLine = pCurLine + charCtr + 1;  // go to first char of next line
                    lineCtr++;                          // update line counter
                    charCtr = 0;                        // reset char counter
                    SetButtonTextPosition(pB, pCurLine, lineCtr);
                    ch = *(pCurLine + charCtr);
                }
            }

            SetClip(CLIP_DISABLE);              // remove clipping
            state = FOCUS_DRAW;                 // go back to IDLE state

        #else

        case TEXT_DRAW:
            if(pB->pText != NULL)
            {
                if(!GetState(pB, BTN_DISABLED))
                {
                    if(GetState(pB, BTN_PRESSED))
                    {
                        SetColor(pB->hdr.pGolScheme->TextColor1);
                    }
                    else
                    {
                        SetColor(pB->hdr.pGolScheme->TextColor0);
                    }
                }
                else
                {
                    SetColor(pB->hdr.pGolScheme->TextColorDisabled);
                }

                SetFont(pB->hdr.pGolScheme->pFont);

                // check text alignment
                if(GetState(pB, BTN_TEXTRIGHT))
                {
```

```
                    xText = pB->hdr.right - (pB->textWidth + GOL_EMBOSS_SIZE + 2);
                }
                else if(GetState(pB, BTN_TEXTLEFT))
                {
                    xText = pB->hdr.left + GOL_EMBOSS_SIZE + 2;
                }
                else
                {

                    // centered text in x direction
                    xText = (pB->hdr.left + pB->hdr.right - pB->textWidth) >> 1;
                }

                if(GetState(pB, BTN_TEXTTOP))
                {
                    yText = pB->hdr.top + GOL_EMBOSS_SIZE + 2;
                }
                else if(GetState(pB, BTN_TEXTBOTTOM))
                {
                    yText = pB->hdr.bottom - (pB->textHeight + GOL_EMBOSS_SIZE);
                }
                else
                {

                    // centered text in y direction
                    yText = (pB->hdr.bottom + pB->hdr.top - pB->textHeight) >> 1;
                }

                MoveTo(xText, yText);
                state = TEXT_DRAW_RUN;
            }
            else
            {
                return DrawButtonFocus(pB, radius, &state);
            }

        case TEXT_DRAW_RUN:
            if(!OutText(pB->pText))
                return (0);
            state = FOCUS_DRAW;
                #endif // #ifdef USE_BUTTON_MULTI_LINE

        case FOCUS_DRAW:
            return DrawButtonFocus(pB, radius, &state);
    }

    return (1);
}

#endif //#if defined (USE_BUTTON) || defined (USE_BUTTON_MULTI_LINE)
```

14.1.9 Button.h

Functions

	Name	Description
	BtnCreate (see page 69)	This function creates a BUTTON (see page 77) 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 (see page 70)	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 (see page 70)
	BtnMsgDefault (see page 74)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	BtnSetText (see page 72)	This function sets the string used for the object.
	BtnTranslateMsg (see page 75)	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
BTN_DISABLED (see page 66)	Bit for disabled state.
BTN_DRAW (see page 67)	Bit to indicate button must be redrawn.
BTN_DRAW_FOCUS (see page 67)	Bit to indicate focus must be redrawn.
BTN_FOCUSED (see page 67)	Bit for focus state.
BTN_HIDE (see page 67)	Bit to indicate button must be removed from screen.
BTN_NOPANEL (see page 69)	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).
BTN_PRESSED (see page 67)	Bit for press state.
BTN_TEXTBOTTOM (see page 68)	Bit to indicate text is top aligned.
BTN_TEXTLEFT (see page 68)	Bit to indicate text is left aligned.
BTN_TEXTRIGHT (see page 68)	Bit to indicate text is right aligned.
BTN_TEXTTOP (see page 68)	Bit to indicate text is bottom aligned.
BTN_TOGGLE (see page 68)	Bit to indicate button will have a toggle behavior.
BTN_TWOTONE (see page 69)	Bit to indicate the button is a two tone type.
BtnGetBitmap (see page 73)	This macro returns the location of the currently used bitmap for the object.
BtnGetText (see page 72)	This macro returns the address of the current text string used for the object.
BtnSetBitmap (see page 74)	This macro sets the bitmap used in the object. The size of the bitmap must match the face of the button.

Structures

Name	Description
BUTTON (see page 77)	Defines the parameters required for a button Object. The following relationships of the parameters determines the general shape of the button: 1. Width (see page 448) is determined by right - left. 2. Height (see page 447) 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*radius = height = width, the button is a circular button.

Description

This is file Button.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
```

```
 *  Button
 ****************************************************************************
 * FileName:        Button.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Linker:          MPLAB LINK30, MPLAB LINK32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2010 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07 Version 1.0 release
 ****************************************************************************/
#ifndef _BUTTON_H
    #define _BUTTON_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define BTN_FOCUSED     0x0001  // Bit for focus state.
    #define BTN_DISABLED    0x0002  // Bit for disabled state.
    #define BTN_PRESSED     0x0004  // Bit for press state.
    #define BTN_TOGGLE      0x0008  // Bit to indicate button will have a toggle behavior.
    #define BTN_TEXTRIGHT   0x0010  // Bit to indicate text is right aligned.
    #define BTN_TEXTLEFT    0x0020  // Bit to indicate text is left aligned.
    #define BTN_TEXTBOTTOM  0x0040  // Bit to indicate text is top aligned.
    #define BTN_TEXTTOP     0x0080  // Bit to indicate text is bottom aligned.
    #define BTN_TWOTONE     0x0100  // Bit to indicate the button is a two tone type.
    #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).

// Note that if bits[7:4] are all zero text is centered.
    #define BTN_DRAW_FOCUS  0x2000  // Bit to indicate focus must be redrawn.
    #define BTN_DRAW        0x4000  // Bit to indicate button must be redrawn.
    #define BTN_HIDE        0x8000  // Bit to indicate button must be removed from screen.
    #define BTN_REMOVE      0x8000

/*********************************************************************
* 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*radius = height = width, the button is a circular button.
*
*********************************************************************/
typedef struct
{
    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.
} BUTTON;

/*********************************************************************
* Macros:  BtnSetBitmap(pB, pBitmap)
*
* Overview: This macro sets the bitmap used in the object.
*           The size of the bitmap must match the face of the button.
*
* PreCondition: none
*
* Input: pB - Pointer to the object.
*        pBitmap - Pointer to the bitmap to be used.
*
* Output: none
*
* Example:
*   <CODE>
*   extern BITMAP_FLASH myIcon;
*   BUTTON *pButton;
*
*       BtnSetBitmap(pButton , &myIcon);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define BtnSetBitmap(pB, pBtmap)    ((BUTTON *)pB)->pBitmap = pBtmap

/*********************************************************************
* Macros:  BtnGetBitmap(pB)
*
* Overview: This macro returns the location of the currently
*           used bitmap for the object.
*
* PreCondition: none
*
* Input: pB - Pointer to the object.
*
* Output: Returns the pointer to the current bitmap used.
*
* Example:
*   <CODE>
*   BUTTON *pButton;
*   BITMAP_FLASH *pUsedBitmap;
*
*       pUsedbitmap = BtnGetBitmap(pButton);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define BtnGetBitmap(pB)    ((BUTTON *)pB)->pBitmap

/*********************************************************************
* Macros:  BtnGetText(pB)
*
* Overview: This macro returns the address of the current
*           text string used for the object.
*
* PreCondition: none
```

```
 *
 * Input: pB - Pointer to the object.
 *
 * Output: Returns pointer to the text string being used.
 *
 * Example:
 *   <CODE>
 *   XCHAR *pChar;
 *   BUTTON Button[2];
 *
 *   pChar = BtnGetText(Button[0]);
 *   </CODE>
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define BtnGetText(pB)  ((BUTTON *)pB)->pText

/********************************************************************
 * Function: BtnSetText(BUTTON *pB, XCHAR *pText)
 *
 * Overview: This function sets the string used for the object.
 *
 * PreCondition: none
 *
 * Input: pB - The pointer to the object whose text will be modified.
 *        pText - Pointer to the text that will be used.
 *
 * Output: none
 *
 * Example:
 *   <CODE>
 *   XCHAR Label0[] = “ON”;
 *   XCHAR Label1[] = “OFF”;
 *   BUTTON Button[2];
 *
 *       BtnSetText(Button[0], Label0);
 *       BtnSetText(Button[1], Label1);
 *   </CODE>
 *
 * Side Effects: none
 *
 ********************************************************************/
void BtnSetText(BUTTON * pB, XCHAR * pText);

/********************************************************************
 * Function: BUTTON *BtnCreate(WORD ID, SHORT left, SHORT top, SHORT right,
 *                             SHORT bottom, SHORT radius, void *pBitmap, XCHAR *pText,
 *                             GOL_SCHEME *pScheme)
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: ID - Unique user defined ID for the object instance.
 *        left - Left most position of the object.
 *        top - Top most position of the object.
 *        right - Right most position of the object.
 *        bottom - Bottom most position of the object.
 *        radius - Radius of the rounded edge.
 *        state - Sets the initial state of the object.
 *        pBitmap - Pointer to the bitmap used on the face of the button
 *                  dimension of the bitmap must match the dimension of the
 *                  button.
 *        pText - Pointer to the text of the button.
 *        pScheme - Pointer to the style scheme used.
 *
 * Output: Returns the pointer to the object created.
 *
 * Example:
```

```
*   <CODE>
*   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;
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
BUTTON  *BtnCreate
        (
            WORD        ID,
            SHORT       left,
            SHORT       top,
            SHORT       right,
            SHORT       bottom,
            SHORT       radius,
            WORD        state,
            void        *pBitmap,
            XCHAR       *pText,
            GOL_SCHEME  *pScheme
        );

/*********************************************************************
* Function: BtnTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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 and the x,y position falls in the face of the button. Current state of the
button is not checked.
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pObj  - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   <CODE>
*   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;
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD    BtnTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: BtnMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       Translated Message   Input Source   Set/Clear State Bit     Description
*       ##################   #############   ######                  ############
*       BTN_MSG_PRESSED      Touch Screen,   Set BTN_PRESSED         Button will be redrawn
in the pressed state.
*                             Keyboard
*       BTN_MSG_RELEASED     Touch Screen,   Clear BTN_PRESSED       Button will be redrawn
in the unpressed state.
*                             Keyboard
*       BTN_MSG_CANCELPRESS  Touch Screen,   Clear BTN_PRESSED       Button will be redrawn
in the unpressed state.
```

```
 *
 *   </TABLE>
 *
 * PreCondition: none
 *
 * Input: translatedMsg - The translated message.
 *        pObj          - The pointer to the object whose state will be modified.
 *        pMsg          - The pointer to the GOL message.
 *
 * Output: none
 *
 * Example:
 *   See BtnTranslateMsg() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
void    BtnMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/********************************************************************
 * Function: WORD BtnDraw(void *pObj)
 *
 * 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.
 *
 * PreCondition: Object must be created before this function is called.
 *
 * Input: pB - Pointer to the object to be rendered.
 *
 * Output: 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.
 *
 * Example:
 *   <CODE>
 *   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;
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD BtnDraw(void *pObj);
#endif // _BUTTON_H
```

14.1.10 Chart.c

This is file Chart.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Chart
 *****************************************************************************
 * FileName:        Chart.c
 * Dependencies:    Chart.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 Version 3.00, C32
 * Linker:          MPLAB LINK30, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paolo A. Tamayo
 * Anton Alkhimenok     4/8/08      ...
 * PAT                  8/8/08      Centered values displayed on bar charts
 *                                  Removed line drawn on pie chart when no slices
 *                                  are present.
 * PAT                  9/30/08     3-D bar depth is now equal to chart depth.
 *                                  Flushed 2-D Bars to equal max height of chart
 *                                  when equal or greater than.
```

```
 * PAT                  6/29/09     Modified Draw Sector function to be state based.
 ****************************************************************************/
#include "Graphics/Graphics.h"
#include <math.h>

#ifdef USE_CHART

// internal functions and macros
WORD        word2xchar(WORD pSmple, XCHAR *xcharArray, WORD cnt);
void        GetCirclePoint(SHORT radius, SHORT angle, SHORT *x, SHORT *y);
WORD        DrawSector(SHORT cx, SHORT cy, SHORT radius, SHORT angleFrom, SHORT angleTo,
WORD outLineColor);
WORD        GetColorShade(WORD color, BYTE shade);
WORD        ChParseShowData(DATASERIES *pData);
DATASERIES  *ChGetNextShowData(DATASERIES *pData);
SHORT       ChSetDataSeries(CHART *pCh, WORD seriesNum, BYTE status);

// array used to define the default colors used to draw the bars or sectors of the chart
const WORD  ChartVarClr[16] = {  CH_CLR0, CH_CLR1, CH_CLR2, CH_CLR3,
    CH_CLR4, CH_CLR5, CH_CLR6, CH_CLR7,
    CH_CLR8, CH_CLR9, CH_CLR10,CH_CLR11,
    CH_CLR12,CH_CLR13,CH_CLR14,CH_CLR15};

/*******************************************************************
* Function: CHART  *ChCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                            SHORT bottom, WORD state ,CHARTDATA *pData,
*                            GOL_SCHEME *pScheme)
*
*
* Notes: Creates a CHART object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*******************************************************************/
CHART *ChCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    DATASERIES  *pData,
    CHARTPARAM  *pParam,
    GOL_SCHEME  *pScheme
)
{
    CHART   *pCh = NULL;

    pCh = (CHART *)GFX_malloc(sizeof(CHART));

    if(pCh == NULL)
        return (NULL);

    pCh->hdr.ID = ID;           // unique id assigned for referencing
    pCh->hdr.pNxtObj = NULL;    // initialize pointer to NULL
    pCh->hdr.type = OBJ_CHART;  // set object type
    pCh->hdr.left = left;       // left position
    pCh->hdr.top = top;         // top position
    pCh->hdr.right = right;     // right position
    pCh->hdr.bottom = bottom;   // bottom position
    pCh->hdr.state = state;     // state
    pCh->hdr.DrawObj = ChDraw;              // draw function
    pCh->hdr.MsgObj = ChTranslateMsg;       // message function
    pCh->hdr.MsgDefaultObj = NULL;          // default message function
    pCh->hdr.FreeObj = ChFreeDataSeries;    // free function

    if(pParam != NULL)
    {
        pCh->prm.pTitle = pParam->pTitle;
        pCh->prm.pSmplLabel = pParam->pSmplLabel;
        pCh->prm.pValLabel = pParam->pValLabel;
```

```
        pCh->prm.smplStart = pParam->smplStart;
        pCh->prm.smplEnd = pParam->smplEnd;
        pCh->prm.valMax = pParam->valMax;
        pCh->prm.valMin = pParam->valMin;
        pCh->prm.pColor = pParam->pColor;
        pCh->prm.pTitleFont = pParam->pTitleFont;
        pCh->prm.pAxisLabelsFont = pParam->pAxisLabelsFont;
        pCh->prm.pGridLabelsFont = pParam->pGridLabelsFont;
    }
    else
    {
        pCh->prm.pTitle = NULL;
        pCh->prm.pSmplLabel = NULL;
        pCh->prm.pValLabel = NULL;
        pCh->prm.smplStart = 0;
        pCh->prm.smplEnd = 0;
        pCh->prm.valMax = 0;
        pCh->prm.valMin = 0;

        // use the default color table
        pCh->prm.pColor = (WORD *)ChartVarClr;
        pCh->prm.pTitleFont = _pDefaultGolScheme->pFont;
        pCh->prm.pAxisLabelsFont = _pDefaultGolScheme->pFont;
        pCh->prm.pGridLabelsFont = _pDefaultGolScheme->pFont;
    }

    pCh->pChData = pData;       // assign the chart data

    // check if how variables have SHOW_DATA flag set
    pCh->prm.seriesCount = ChParseShowData(pData);

    // Set the color scheme to be used
    if(pScheme == NULL)
    {
        pCh->hdr.pGolScheme = _pDefaultGolScheme;
    }
    else
    {
        pCh->hdr.pGolScheme = (GOL_SCHEME *)pScheme;
        pCh->prm.pTitleFont = pCh->hdr.pGolScheme->pFont;
        pCh->prm.pAxisLabelsFont = pCh->hdr.pGolScheme->pFont;
        pCh->prm.pGridLabelsFont = pCh->hdr.pGolScheme->pFont;
    }

    GOLAddObject((OBJ_HEADER *)pCh);

    return (pCh);
}

/*****************************************************************
* Function: WORD ChTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*****************************************************************/
WORD ChTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    CHART *pCh;

    pCh = (CHART *)pObj;

    // Evaluate if the message is for the static text
    // Check if disabled first
    if(GetState(pCh, CH_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
```

```
        // Check if it falls in static text control borders
        if
        (
            (pCh->hdr.left < pMsg->param1) &&
            (pCh->hdr.right > pMsg->param1) &&
            (pCh->hdr.top < pMsg->param2) &&
            (pCh->hdr.bottom > pMsg->param2)
        )
        {
            return (CH_MSG_SELECTED);
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

////////////////////////////////////////////////

// internal functions
////////////////////////////////////////////////
DATASERIES *ChGetNextShowData(DATASERIES *pData)
{
    DATASERIES  *pVar = pData;

    // find the next data series that will be shown
    while(pVar->show != SHOW_DATA)
    {
        if((pVar = (DATASERIES *)pVar->pNextData) == NULL)
            return (NULL);
    }

    return (pVar);
}

/* */
WORD ChParseShowData(DATASERIES *pData)
{
    DATASERIES  *pParse;
    WORD        sCnt = 0;

    if(pData != NULL)
    {
        pParse = pData;
        while(pParse != NULL)
        {
            if(pParse->show == SHOW_DATA)
                sCnt++;
            pParse = (DATASERIES *)pParse->pNextData;
        }
    }

    return (sCnt);
}

/* */
WORD GetLongestNameLength(CHART *pCh)
{
    WORD        temp = 0;
    DATASERIES  *pVar;

    if(!GetState(pCh, CH_LEGEND))
        return (0);

    // find the data series with the longest name
    pVar = pCh->pChData;

    while(pVar)
    {

        // check if the data series is to be shown
        if(pVar->show == SHOW_DATA)
```

```
        {
            if(temp < GetTextWidth((XCHAR *)pVar->pSData, pCh->hdr.pGolScheme->pFont))
                temp = GetTextWidth((XCHAR *)pVar->pSData, pCh->hdr.pGolScheme->pFont);
        }

        pVar = pVar->pNextData;
    }

    return (temp);
}

/* */
WORD word2xchar(WORD pSmple, XCHAR *xcharArray, WORD cnt)
{
    WORD            j, z;
    static XCHAR    *pXchar;

    pXchar = xcharArray;

    // this implements sprintf(strVal, "%d", temp); faster
    // note that this is just for values >= 0, while sprintf covers negative values.
    j = 1;
    z = pSmple;

    pXchar = &(*xcharArray) + (cnt - j);
    do
    {
        *pXchar = (z % 10) + '0';
        *pXchar--;
        if((z /= 10) == 0)
            break;
        j++;
    } while(j <= cnt);
    return (j);
}

/*****************************************************************
* Function: WORD GetColorShade(WORD color, BYTE shade)
*
*
* Notes: This function gets the given color in 5-6-5 (RGB) format
*        and computes the shade of the same color by shifting
*        the rgb values depending on the shade value.
*        The idea here is to get the given r,g and b colors and
*        make them approach the gray color by shifting the each value
*        closer to 128. If rgb values are > 128 we subtract and add
*        if < 128 we add.
*
*****************************************************************/
WORD GetColorShade(WORD color, BYTE shade)
{
    WORD    newColor;
    BYTE    rgb[3];

    rgb[0] = ((color >> 11) << 3);  // red
    rgb[1] = ((color >> 5) << 2);   // green
    rgb[2] = ((color) << 3);        // blue
    BYTE    i;

    for(i = 0; i < 3; i++)
    {
        if(rgb[i] > 128)
            rgb[i] = rgb[i] - ((rgb[i] - 128) >> (shade));
        else
            rgb[i] = rgb[i] + ((128 - rgb[i]) >> (shade));
    }

    newColor = RGB565CONVERT(rgb[0], rgb[1], rgb[2]);
    return (newColor);
}

/*****************************************************************
```

```
* Function: WORD ChDraw(void *pObj)
*
*
* Notes: This is the state machine to draw the button.
*
*********************************************************************/
    #define STR_CHAR_CNT        11
    #define DCLR_STR_CHAR_CNT   (STR_CHAR_CNT + 1)

/* */

WORD ChDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        FRAME_DRAW_PREP,
        FRAME_DRAW,
        CHECK_CHART_TYPE,

        // BAR type states
        GRID_PREP,
        SAMPLE_GRID_DRAW1,
        VALUE_GRID_DRAW1,
        SAMPLE_GRID_DRAW2,
        VALUE_GRID_DRAW2,
        VALUE_GRID_3D_DRAW,
        TITLE_LABEL_DRAW_SET,
        TITLE_LABEL_DRAW_RUN,
        SAMPLE_LABEL_DRAW_SET,
        SAMPLE_LABEL_DRAW_RUN,
        VALUE_LABEL_DRAW_INIT,
        VALUE_LABEL_DRAW_SET,
        VALUE_LABEL_DRAW_RUN,
        XAXIS_LABEL_DRAW_RUN,
        XAXIS_LABEL_DRAW_SET,
        YAXIS_LABEL_DRAW_RUN,
        YAXIS_LABEL_DRAW_SET,
        LEGEND_DRAW_BOX,
        LEGEND_DRAW_RUN,
        LEGEND_DRAW_UPDATE_VAR,
        DATA_DRAW_INIT,
        DATA_DRAW_SET,
        BAR_DATA_DRAW,
        BAR_DATA_DRAW_CHECK,
        BAR_DATA_DRAW_3D_PREP,
        BAR_DATA_DRAW_3D_LOOP_1,
        BAR_DATA_DRAW_3D_LOOP_2,
        BAR_DATA_DRAW_3D_OUTLINE1,
        BAR_DATA_DRAW_3D_OUTLINE2,
        BAR_DATA_DRAW_3D_OUTLINE3,
        BAR_DATA_DRAW_3D_OUTLINE4,
        BAR_DATA_DRAW_3D_OUTLINE5,
        PIE_DONUT_HOLE_DRAW,
        BAR_DATA_DRAW_VALUE,
        BAR_DATA_DRAW_VALUE_RUN,

        // PIE type states
        PIE_PREP,
        PIE_DRAW_OUTLINE1,
        PIE_DRAW_SECTOR,
        PIE_DRAW_SECTOR_LOOP,
        PIE_DRAW_SECTOR_ACTUAL,
        PIE_DRAW_SECTOR_LOOP_CONTINUE,
        PIE_DRAW_SECTOR_LOOP_CREATE_STRINGS,
        PIE_DRAW_SECTOR_LOOP_STRINGS_RUN,
    } CH_DRAW_STATES;

    static XCHAR tempXchar[2] = {'B',0};
    static XCHAR temp2Xchar[2] = {'M',0};
    static XCHAR tempStr[DCLR_STR_CHAR_CNT] = {'0','0','0','0','0','0','0','0','0','0',0};
```

```
    static CH_DRAW_STATES state = REMOVE;
    static WORD x, y, z, xStart, yStart, ctr, ctry, samplesMax, temp;
    static SHORT uLocator;
    static WORD splDelta, valDelta;
    static WORD barWidth, barDepth, chart3DDepth;

    static DATASERIES *pVar;
    static WORD *pSmple;
    static XCHAR *pXcharTemp;
    static DWORD dTemp;
    static SHORT varCtr, pieX, pieY;
    static DWORD dPercent;
    static WORD j = 0, k = 0, h = 0, i, m;
    static void *pVarFont;

    static WORD pieLabelXPos;
    static WORD pieLabelYPos, pieLabelYPos2, pieLabelYPos3;
    static WORD pieSectorXPos;
    static WORD pieSectorYPos;

    CHART *pCh;

    pCh = (CHART *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(IsDeviceBusy())
                return (0);

            if(GetState(pCh, CH_HIDE))
            {   // Hide the Chart (remove from screen)
                SetColor(pCh->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pCh->hdr.left, pCh->hdr.top, pCh->hdr.right, pCh->hdr.bottom))
                    return (0);
                return (1);
            }

            SetLineThickness(NORMAL_LINE);
            SetLineType(SOLID_LINE);

            // check if we only need to refresh the data on the chart
            if(GetState(pCh, CH_DRAW_DATA))
            {

                // this is only performed when refreshing data in the chart
                // erase the current contents
                SetColor(pCh->hdr.pGolScheme->CommonBkColor);

                // get the ending x position where redraw will take place (if legend is
drawn,
                // we do not need to redraw this area)
                i = GetLongestNameLength(pCh) + GetTextHeight(pCh->hdr.pGolScheme->pFont) +
GOL_EMBOSS_SIZE + (CH_MARGIN << 1);

                if(GetState(pCh, CH_BAR))
                {

                    // get the starting x position where redraw will take place
                    h = xStart - GetTextWidth(tempXchar, pCh->prm.pGridLabelsFont) -
(GetTextWidth(temp2Xchar, pCh->prm.pGridLabelsFont) >> 1);

                    // get the starting y position
                    j = yStart - ((ChGetSampleRange(pCh) + 1) * splDelta);
                    if(GetState(pCh, CH_3D_ENABLE))
                    {
                        if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                        {
```

```
                                // adjust for 3D effects
                                j -= chart3DDepth;
                                if((GetState(pCh, CH_LEGEND)) && (ChGetShowSeriesCount(pCh) !=
1))
                                {
                                    if(!Bar(h, j, pCh->hdr.right - i, yStart))
                                        return (0);
                                }
                                else
                                {
                                    if(!Bar(h, j, pCh->hdr.right - CH_MARGIN, yStart))
                                        return (0);
                                }
                            }
                            else
                            {
                                if
                                (
                                    !Bar
                                        (
                                            xStart,
                                            yStart - ((CH_YGRIDCOUNT - 1) * valDelta) -
chart3DDepth - (GetTextHeight(pCh->hdr.pGolScheme->pFont)),
                                            xStart + splDelta * (ChGetSampleRange(pCh) + 1) +
chart3DDepth,
                                            yStart + GetTextHeight(pCh->prm.pGridLabelsFont)
                                        )
                                ) return (0);
                            }
                        }
                        else
                        {
                            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                            {
                                if((GetState(pCh, CH_LEGEND)) && (ChGetShowSeriesCount(pCh) !=
1))
                                {
                                    if(!Bar(h, j, pCh->hdr.right - i, yStart))
                                        return (0);
                                }
                                else
                                {
                                    if(!Bar(h, j, pCh->hdr.right - CH_MARGIN, yStart))
                                        return (0);
                                }
                            }
                            else
                            {
                                if
                                (
                                    !Bar
                                        (
                                            xStart,
                                            yStart - ((CH_YGRIDCOUNT - 1) * valDelta) -
(GetTextHeight(pCh->hdr.pGolScheme->pFont)),
                                            xStart + splDelta * (ChGetSampleRange(pCh) + 1),
                                            yStart + GetTextHeight(pCh->prm.pGridLabelsFont)
                                        )
                                ) return (0);
                            }
                        }
                    }
                    else
                    {
                        if((GetState(pCh, CH_LEGEND)) && (ChGetShowSeriesCount(pCh) != 1))
                            i = pCh->hdr.right - i;
                        else
                            i = pCh->hdr.right - GOL_EMBOSS_SIZE - CH_MARGIN;
                        h = pCh->hdr.left + GOL_EMBOSS_SIZE;
                        j = pCh->hdr.top + GOL_EMBOSS_SIZE + CH_MARGIN +
GetTextHeight(pCh->prm.pTitleFont);
```

```
                    // erase the current pie chart drawn
                    if(!Bar(h, j, i, pCh->hdr.bottom - CH_MARGIN))
                        return (0);
                }

                // check the type of chart
                if(GetState(pCh, CH_BAR))
                {
                    state = GRID_PREP;
                    goto chrt_grid_prep;
                }
                else
                {
                    state = PIE_PREP;
                    goto chrt_pie_prep;
                }
            }

            state = FRAME_DRAW_PREP;

        /*============================================================================*/
        //                      Draw the frame
        /*============================================================================*/
        case FRAME_DRAW_PREP:

            // check how many data series do we need to display
            pCh->prm.seriesCount = ChParseShowData(pCh->pChData);

            // set up the frame drawing
            GOLPanelDraw
            (
                pCh->hdr.left,
                pCh->hdr.top,
                pCh->hdr.right,
                pCh->hdr.bottom,
                0,
                pCh->hdr.pGolScheme->CommonBkColor,
                pCh->hdr.pGolScheme->EmbossLtColor,
                pCh->hdr.pGolScheme->EmbossDkColor,
                NULL,
                1
            );

            state = FRAME_DRAW;

        case FRAME_DRAW:
            if(!GOLPanelDrawTsk())
            {
                return (0);
            }

            state = TITLE_LABEL_DRAW_SET;

        /*============================================================================*/
        //                      Draw the Chart Title
        /*============================================================================*/
        case TITLE_LABEL_DRAW_SET:

            // find the location of the title
            MoveTo
            (
                pCh->hdr.left + ((pCh->hdr.right + pCh->hdr.left - GetTextWidth((XCHAR
*)pCh->prm.pTitle, pCh->prm.pTitleFont)) >> 1),
                pCh->hdr.top + CH_MARGIN
            );
            state = TITLE_LABEL_DRAW_RUN;

        case TITLE_LABEL_DRAW_RUN:

            // Set the font
            SetFont(pCh->prm.pTitleFont);
```

```
            // NOTE: we use the emboss dark color here to draw the chart title.
            SetColor(pCh->hdr.pGolScheme->EmbossDkColor);

            if(!OutText(pCh->prm.pTitle))
                return (0);

            // check if legend will be drawn
            if(GetState(pCh, CH_LEGEND) && (ChGetShowSeriesCount(pCh) != 1))
            {

                // position the x and y points to the start of the first variable
                temp = GetLongestNameLength(pCh);

                x = pCh->hdr.right - (CH_MARGIN << 1) - temp -
GetTextHeight(pCh->hdr.pGolScheme->pFont);
                y = ((pCh->hdr.bottom + pCh->hdr.top) >> 1) - ((pCh->prm.seriesCount *
GetTextHeight(pCh->hdr.pGolScheme->pFont)) >> 1);

                // initialize the variable counter for the legend drawing
                temp = 0;
                pVar = (DATASERIES *)pCh->pChData;
                state = LEGEND_DRAW_BOX;
            }
            else
            {

                // legend will not be drawn, go to data drawing next
                state = CHECK_CHART_TYPE;
                goto chrt_check_chart_type;
            }

            /*=========================================================================*/
            //                      Draw the Legend
            /*=========================================================================*/
    chrt_draw_legend:

        case LEGEND_DRAW_BOX:

            // check if we will be showing this data series
            if(ChGetShowSeriesStatus(pVar) == SHOW_DATA)
            {
                SetColor(*(&(*pCh->prm.pColor) + temp));
                if((ChGetShowSeriesCount(pCh) == 1) && (GetState(pCh, CH_PIE)))
                { }
                else
                {
                    if
                    (
                        !Bar
                            (
                                x,
                                y + (GetTextHeight(pCh->hdr.pGolScheme->pFont) >> 2),
                                x + (GetTextHeight(pCh->hdr.pGolScheme->pFont) >> 1),
                                y +
                                    (
                                        GetTextHeight(pCh->hdr.pGolScheme->pFont) -
                                            (GetTextHeight(pCh->hdr.pGolScheme->pFont) >> 2)
                                    )
                            )
                    ) return (0);
                }

                MoveTo(x + 2 + (GetTextHeight(pCh->hdr.pGolScheme->pFont) >> 1), y);
                state = LEGEND_DRAW_RUN;
            }
            else
            {
                state = LEGEND_DRAW_UPDATE_VAR;
                goto chrt_draw_legend_update;
            }

        case LEGEND_DRAW_RUN:
```

```
            SetColor(pCh->hdr.pGolScheme->TextColor1);
            SetFont(pCh->hdr.pGolScheme->pFont);

            if(!OutText(pVar->pSData))
                return (0);

            // increment the variable counter
            temp++;
            if(temp == ChGetShowSeriesCount(pCh))
            {
                state = CHECK_CHART_TYPE;
                goto chrt_check_chart_type;
            }
            else
                state = LEGEND_DRAW_UPDATE_VAR;

    chrt_draw_legend_update:

        case LEGEND_DRAW_UPDATE_VAR:

            // update the data series pointer and y position
            if(ChGetShowSeriesStatus(pVar) == SHOW_DATA)
                y += GetTextHeight(pCh->hdr.pGolScheme->pFont);
            pVar = (DATASERIES *)pVar->pNextData;
            state = LEGEND_DRAW_BOX;
            goto chrt_draw_legend;

    chrt_check_chart_type:

        case CHECK_CHART_TYPE:
            if(GetState(pCh, CH_BAR))
            {
                state = GRID_PREP;
            }
            else if(GetState(pCh, CH_PIE))
            {
                state = PIE_PREP;
                goto chrt_pie_prep;
            }
            else
            {
                state = REMOVE;
                return (1);
            }

            /***************************************************************************/
            //                    BAR CHART states
            /***************************************************************************/

            /*=========================================================================*/
            //                      Draw the grids
            /*=========================================================================*/
    chrt_grid_prep:

        case GRID_PREP:

            /* NOTE: X or Y Grid Labels - label for each division in the x or y axis
                     X or Y Axis Labels - label to name the x or y axis. Text is
                                          user given in the CHART structure,
                                          CHARTPARAM member.
            */

            // count the number of characters needed for the axis label that
            // represents the value of the samples (or bars)
            // get the width of one character
            temp = GetTextWidth((XCHAR *)tempXchar, pCh->prm.pGridLabelsFont);

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {

                // if in the horizontal orientation width will only be
                // max of 2 characters (1, 2, 3...10, 11... or A, B, C...)
```

```
                if((ChGetSampleEnd(pCh) - ChGetSampleStart(pCh)) > 9)
                    y = 2;
                else
                    y = 1;
            }
            else
            {
                if(GetState(pCh, CH_PERCENT))
                {

                    // include in the computation the space that will be occupied by '%'
sign
                    y = 4;
                }
                else
                {
                    x = ChGetValueRange(pCh);
                    y = 1;

                    // count the number of characters needed
                    while(x /= 10)
                        ++y;
                }
            }

            // estimate the space that will be occupied by the y grid labels
            temp = temp * y;

            // get x starting position
            xStart = CH_MARGIN + temp + pCh->hdr.left;

            // adjust x start to accomodate Y axis label
            xStart += (GetTextHeight(pCh->prm.pAxisLabelsFont) + (GetTextWidth(temp2Xchar,
pCh->prm.pGridLabelsFont) >> 1));

            // now get y starting position
            yStart = pCh->hdr.bottom - (GetTextHeight(pCh->prm.pGridLabelsFont) +
GetTextHeight(pCh->prm.pAxisLabelsFont) + CH_MARGIN);

            // ========================================================================
            // Sample delta (splDelta) and Value delta (valDelta) will depend if the
            // chart is drawn horizontally or vertically.
            // ========================================================================
            // find the variable with the longest name
            // to add space for the names of variables in the legend
            // Text Height here refers to the legend for colors (the drawn filled rectangle)
            if((ChGetShowSeriesCount(pCh) == 1) || (GetState(pCh, CH_LEGEND) != CH_LEGEND))
                temp = 0;
            else
                temp = GetLongestNameLength(pCh) +
GetTextHeight(pCh->hdr.pGolScheme->pFont);

            // get sample delta (space between data) and value delta (defines the grid for
the value)
            // check first if we compute in the x-axis or y-axis
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {

                // if horizontally drawn sample delta is not affected by the legend
                splDelta = (yStart - (pCh->hdr.top + CH_MARGIN +
GetTextHeight(pCh->prm.pTitleFont)));

                // adjust space for displayed values
                if(GetState(pCh, CH_VALUE))
                {
                    temp += (GetTextWidth(tempXchar, pCh->hdr.pGolScheme->pFont) * 4);
                }

                // get the value delta,
                valDelta = (pCh->hdr.right - xStart - CH_MARGIN - temp);
            }
            else
```

```
            {
                if(GetState(pCh, CH_LEGEND) && (ChGetShowSeriesCount(pCh) != 1))
                {
                    splDelta =
                        (
                            pCh->hdr.right -
                            xStart -
                            ((CH_MARGIN << 2) + temp +
GetTextHeight(pCh->hdr.pGolScheme->pFont))
                        );
                }
                else
                {
                    splDelta = (pCh->hdr.right - xStart - (CH_MARGIN << 2));
                }

                // get the value delta
                valDelta =
                    (
                        yStart -
                        (pCh->hdr.top + CH_MARGIN + GetTextHeight(pCh->prm.pTitleFont) +
GetTextHeight(pCh->hdr.pGolScheme->pFont))
                    );
            }

            // adjust the splDelta for 3D effects, 12 here is the maximum depth of a bar
for the 3D effect
            if(GetState(pCh, CH_3D_ENABLE))
            {
                splDelta -= 12;
                valDelta -= 12;
            }

            // get the final splDelta value by checking the number of samples to
display
            splDelta /= (ChGetSampleRange(pCh) + 1);

            // get the final valDelta value by checking the number of samples to
display
            valDelta /= (CH_YGRIDCOUNT - 1);

            // initilize the counter for the sample axis drawing
            temp = ChGetSampleRange(pCh) + 2;
            x = xStart;
            y = yStart;
            state = SAMPLE_GRID_DRAW1;

        case SAMPLE_GRID_DRAW1:

            // draw the small grids on the x-axis
            while(temp)
            {
                SetColor(pCh->hdr.pGolScheme->Color0);
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if(!Bar(x, y, x + 3, y + 1))
                        return (0);
                    y -= splDelta;
                }
                else
                {
                    if(!Bar(x, y, x + 1, y + 3))
                        return (0);
                    x += splDelta;
                }

                --temp;
            }

            // get the bar width (bar here refers to the sample data represented as bars,
where the height
            // of the bar represents the value of the sample)
```

```
            barWidth = splDelta / (2 + ChGetShowSeriesCount(pCh));

            temp = CH_YGRIDCOUNT;
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                y = yStart;
            else
                x = xStart;

            if(GetState(pCh, CH_3D_ENABLE))
            {

                // limit the 3-D depth to only 12 pixels.
                chart3DDepth = (barWidth > 12) ? 12 : barWidth;
                chart3DDepth = chart3DDepth >> 1;

                // set the bar 3-D depth.
                barDepth = chart3DDepth;
                state = VALUE_GRID_3D_DRAW;
            }
            else
            {
                state = VALUE_GRID_DRAW1;
                goto chrt_value_grid_draw1;
            }

        case VALUE_GRID_3D_DRAW:

            // draw the 3D grids on the value-axis
            while(temp)
            {
                SetColor(pCh->hdr.pGolScheme->Color0);
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if
                    (
                        !Bar
                            (
                                x + (chart3DDepth),
                                y - (chart3DDepth) - (splDelta * (ChGetSampleRange(pCh) +
1)),
                                x + (chart3DDepth),
                                y - (chart3DDepth)
                            )
                    ) return (0);
                    x += valDelta;
                }
                else
                {
                    if
                    (
                        !Bar
                            (
                                x + (chart3DDepth),
                                y - (chart3DDepth),
                                x + (chart3DDepth) + (splDelta * (ChGetSampleRange(pCh) +
1)),
                                y - (chart3DDepth)
                            )
                    ) return (0);
                    y -= valDelta;
                }

                --temp;
            }

            temp = CH_YGRIDCOUNT;
            x = xStart;
            y = yStart;
            state = VALUE_GRID_DRAW1;

    chrt_value_grid_draw1:
```

```
        case VALUE_GRID_DRAW1:

            // draw the grids on the y-axis
            if(GetState(pCh, CH_3D_ENABLE))
            {

                // just draw the first one to define the x-axis
                SetColor(pCh->hdr.pGolScheme->Color0);
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if(!Bar(x, y - (splDelta * (ChGetSampleRange(pCh) + 1)), x, y))
                        return (0);
                }
                else
                {
                    if(!Bar(x, y, x + (splDelta * (ChGetSampleRange(pCh) + 1)), y))
                        return (0);
                }
            }
            else
            {
                while(temp)
                {
                    SetColor(pCh->hdr.pGolScheme->Color0);
                    if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                    {
                        if(!Bar(x, y - (splDelta * (ChGetSampleRange(pCh) + 1)), x, y))
                            return (0);
                        x += valDelta;
                    }
                    else
                    {
                        if(!Bar(x, y, x + (splDelta * (ChGetSampleRange(pCh) + 1)), y))
                            return (0);
                        y -= valDelta;
                    }

                    --temp;
                }
            }

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                x = xStart;
            }
            else
            {
                y = yStart;
            }

            if(GetState(pCh, CH_3D_ENABLE))
            {
                temp = CH_YGRIDCOUNT + 1;
                state = VALUE_GRID_DRAW2;
            }
            else
            {
                temp = 2;
                state = SAMPLE_GRID_DRAW2;
                goto chrt_xgrid_draw2;
            }

        case VALUE_GRID_DRAW2:

            // draw the 3-D effect on the y axis of the chart
            SetColor(pCh->hdr.pGolScheme->Color0);
            while(temp)
            {
                if(temp == (CH_YGRIDCOUNT + 1))
                {
                    if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                    {
```

```
                        if
                        (
                            !Line
                                (
                                    x,
                                    y - (splDelta * (ChGetSampleRange(pCh) + 1)),
                                    x + chart3DDepth,
                                    y - (splDelta * (ChGetSampleRange(pCh) + 1)) -
chart3DDepth
                                )
                        ) return (0);
                    }
                    else
                    {
                        if
                        (
                            !Line
                                (
                                    x + (splDelta * (ChGetSampleRange(pCh) + 1)),
                                    y,
                                    x + (chart3DDepth) + (splDelta * (ChGetSampleRange(pCh)
+ 1)),
                                    y - chart3DDepth
                                )
                        ) return (0);
                    }

                    --temp;
                    continue;
                }
                else if(!Line(x, y, x + chart3DDepth, y - chart3DDepth))
                    return (0);

                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    x += valDelta;
                }
                else
                {
                    y -= valDelta;
                }

                --temp;
            }

            temp = 3;
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                x = xStart;
            }
            else
            {
                y = yStart;
            }

            state = SAMPLE_GRID_DRAW2;

    chrt_xgrid_draw2:

        case SAMPLE_GRID_DRAW2:

            // draw the left and right edges of the chart
            while(temp)
            {
                if(GetState(pCh, CH_3D_ENABLE))
                {
                    if(temp == 3)
                    {
                        if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                        {
                            if(!Bar(x, y, x + ((CH_YGRIDCOUNT - 1) * valDelta), y))
                                return (0);
```

```
                    }
                    else
                    {
                        if(!Bar(x, y - ((CH_YGRIDCOUNT - 1) * valDelta), x, y))
                            return (0);
                    }

                    --temp;
                    continue;
                }
                else if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if
                    (
                        !Bar
                            (
                                x + chart3DDepth,
                                y - chart3DDepth,
                                x + chart3DDepth + ((CH_YGRIDCOUNT - 1) * valDelta),
                                y - chart3DDepth
                            )
                    ) return (0);
                }
                else
                {
                    if
                    (
                        !Bar
                            (
                                x + chart3DDepth,
                                y - ((CH_YGRIDCOUNT - 1) * valDelta) - chart3DDepth,
                                x + chart3DDepth,
                                y - chart3DDepth
                            )
                    ) return (0);
                }
            }
            else
            {
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if(!Bar(x, y, x + ((CH_YGRIDCOUNT - 1) * valDelta), y))
                        return (0);
                }
                else
                {
                    if(!Bar(x, y - ((CH_YGRIDCOUNT - 1) * valDelta), x, y))
                        return (0);
                }
            }

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                y -= (splDelta * (ChGetSampleRange(pCh) + 1));
            }
            else
            {
                x += (splDelta * (ChGetSampleRange(pCh) + 1));
            }

            --temp;
        }

        if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                #ifdef USE_HORZ_ASCENDING_ORDER
            y = yStart - (ChGetSampleRange(pCh) * splDelta);
            #else
        y = yStart;
            #endif
        else
            x = xStart;
        ctr = ChGetSampleStart(pCh);
```

```
            state = SAMPLE_LABEL_DRAW_SET;

            /*============================================================================*/
            //                      Draw the Sample Grid labels
            /*============================================================================*/
    chrt_sample_label_draw_set:

        case SAMPLE_LABEL_DRAW_SET:

            // for data only redraw, we need to refresh the x-axis labels to indicate
            // the correct sample points being shown
            SetFont(pCh->prm.pGridLabelsFont);

            if(GetState(pCh, CH_NUMERIC))
            {
                j = word2xchar(ctr, tempStr, STR_CHAR_CNT);
            }
            else
            {

                // note that we will only have A-Z labels.
                tempStr[STR_CHAR_CNT - 1] = 'A' + (ctr - 1);
                j = 1;
            }

            temp = GetTextWidth((&tempStr[STR_CHAR_CNT - j]), pCh->prm.pGridLabelsFont);
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                MoveTo(x - temp, y - ((splDelta + GetTextHeight(pCh->prm.pGridLabelsFont))
>> 1));
            }
            else
            {
                MoveTo(x + ((splDelta - temp) >> 1), y);
            }

            state = SAMPLE_LABEL_DRAW_RUN;

        case SAMPLE_LABEL_DRAW_RUN:

            // draw the x axis grid numbers
            SetColor(pCh->hdr.pGolScheme->TextColor0);
            if(!OutText(&tempStr[STR_CHAR_CNT - j]))
                return (0);
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                    #ifdef USE_HORZ_ASCENDING_ORDER
                y += splDelta;
                    #else
                y -= splDelta;
                    #endif
            }
            else
            {
                x += splDelta;
            }

            ctr++;
            if(ctr > ChGetSampleEnd(pCh))
            {

                // check if we only need to redraw the data
                if(GetState(pCh, CH_DRAW_DATA))
                {
                    state = DATA_DRAW_INIT;
                    goto chrt_data_draw_init;
                }
                else
                {
                    if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                    {
```

```
                        state = YAXIS_LABEL_DRAW_SET;
                        goto chrt_yaxis_label_draw_set;
                    }
                    else
                    {
                        state = XAXIS_LABEL_DRAW_SET;
                    }
                }
            }
            else
            {
                temp = x;
                goto chrt_sample_label_draw_set;
            }

            /*==========================================================================*/
            //                      Draw the X - Axis labels
            /*==========================================================================*/
chrt_xaxis_label_draw_set:

        case XAXIS_LABEL_DRAW_SET:

            // find the location of the label
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                pXcharTemp = pCh->prm.pValLabel;
                uLocator = valDelta * (CH_YGRIDCOUNT - 1);
            }
            else
            {
                pXcharTemp = pCh->prm.pSmplLabel;
                uLocator = splDelta * (ChGetSampleRange(pCh) + 1);
            }

            temp = GetTextWidth(pXcharTemp, pCh->prm.pAxisLabelsFont);

            if(temp > uLocator)
                temp = xStart;
            else
                temp = xStart + ((uLocator - temp) >> 1);

            MoveTo(temp, yStart + GetTextHeight(pCh->prm.pGridLabelsFont));
            state = XAXIS_LABEL_DRAW_RUN;

        case XAXIS_LABEL_DRAW_RUN:
            SetFont(pCh->prm.pAxisLabelsFont);

            // enable clipping and set region
            SetClip(CLIP_ENABLE);
            SetClipRgn(pCh->hdr.left, pCh->hdr.top, pCh->hdr.right, pCh->hdr.bottom);

            SetColor(pCh->hdr.pGolScheme->TextColor1);
            if(!OutText(pXcharTemp))
                return (0);

            SetClip(CLIP_DISABLE);
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                state = DATA_DRAW_INIT;
                goto chrt_data_draw_init;
            }
            else
            {
                state = VALUE_LABEL_DRAW_INIT;
            }

chrt_value_label_draw_init:

        case VALUE_LABEL_DRAW_INIT:
            ctr = 0;

            // x is used here to represent change in value grid labels
```

```
            // Note that we add a multiplier of 10 to compute for round off errors.
            // It will not be perfect but approximations is better unless you work with
            // figures divisible by 5.
            if(GetState(pCh, CH_PERCENT))
            {

                // Scaling of the labels is included here
                x = ChGetPercentRange(pCh) * 100 / (CH_YGRIDCOUNT - 1);
            }
            else
            {
                x = (ChGetValueRange(pCh) * 100) / (CH_YGRIDCOUNT - 1);
            }

            // compute for round off error, the adjustment for the factor 100 is done in
            // conversion of the integer to string below.
            if((x % 10) < 5)
                x += 10;

            state = VALUE_LABEL_DRAW_SET;

            /*=========================================================================*/
            //                      Draw the Value Grid labels
            /*=========================================================================*/
    chrt_value_label_draw_set:

        case VALUE_LABEL_DRAW_SET:

            // note that the adjustment of the 100 factor is done here.
            if(GetState(pCh, CH_PERCENT))
            {

                // add the percent sign on the label
                tempStr[STR_CHAR_CNT - 1] = '%';

                // we have a plus 1 here since we add '%' sign already
                j = 1 + word2xchar((ctr * x / 100) + ChGetPercentMin(pCh), tempStr,
STR_CHAR_CNT - 1);
            }
            else
            {
                j = word2xchar((ctr * x / 100) + ChGetValueMin(pCh), tempStr, STR_CHAR_CNT);
            }

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                temp = GetTextWidth((&tempStr[STR_CHAR_CNT - j]), pCh->prm.pGridLabelsFont);

                MoveTo(xStart + ((valDelta * ctr) - (temp >> 1)), yStart);
            }
            else
            {
                temp = GetTextHeight(pCh->prm.pGridLabelsFont);
                MoveTo
                (
                    xStart - GetTextWidth((&tempStr[STR_CHAR_CNT - j]),
pCh->prm.pGridLabelsFont),
                    yStart - ((valDelta * ctr) + (temp >> 1))
                );
            }

            SetFont(pCh->prm.pGridLabelsFont);
            state = VALUE_LABEL_DRAW_RUN;

        case VALUE_LABEL_DRAW_RUN:

            // draw the y axis grid numbers
            SetColor(pCh->hdr.pGolScheme->TextColor0);
            if(!OutText(&tempStr[STR_CHAR_CNT - j]))
                return (0);
            ctr++;
            if(ctr >= CH_YGRIDCOUNT)
```

```
            {
                state = XAXIS_LABEL_DRAW_SET;
            }
            else
            {
                goto chrt_value_label_draw_set;
            }

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                state = XAXIS_LABEL_DRAW_SET;
                goto chrt_xaxis_label_draw_set;
            }
            else
            {
                state = YAXIS_LABEL_DRAW_SET;
            }

            /*==========================================================================*/
            //                     Draw the Y - Axis labels
            /*==========================================================================*/
    chrt_yaxis_label_draw_set:

        case YAXIS_LABEL_DRAW_SET:

            // find the location of the label
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                pXcharTemp = pCh->prm.pSmplLabel;
                uLocator = splDelta * (ChGetSampleRange(pCh) + 1);
            }
            else
            {
                pXcharTemp = pCh->prm.pValLabel;
                uLocator = valDelta * CH_YGRIDCOUNT;
            }

            temp = GetTextWidth(pXcharTemp, pCh->prm.pAxisLabelsFont);

            if(temp > uLocator)
                temp = (pCh->hdr.bottom + pCh->hdr.top + temp) >> 1;
            else
            {
                temp = yStart - ((uLocator - temp) >> 1);
            }

            MoveTo
            (
                xStart - GetTextWidth((&tempStr[STR_CHAR_CNT - j]),
pCh->prm.pGridLabelsFont) - (GetTextWidth(temp2Xchar, pCh->prm.pGridLabelsFont) >> 1) -
GetTextHeight(pCh->prm.pAxisLabelsFont),
                temp
            );

            state = YAXIS_LABEL_DRAW_RUN;

        case YAXIS_LABEL_DRAW_RUN:
            SetFont(pCh->prm.pAxisLabelsFont);

            // enable clipping and set region
            SetClip(CLIP_ENABLE);
            SetClipRgn(pCh->hdr.left, pCh->hdr.top, pCh->hdr.right, pCh->hdr.bottom);

            // set the orientation of text to vertical
            SetFontOrientation(ORIENT_VER);
            SetColor(pCh->hdr.pGolScheme->TextColor1);
            if(!OutText(pXcharTemp))
                return (0);

            SetClip(CLIP_DISABLE);

            // place back the orientation of text to horizontal
```

```
            SetFontOrientation(ORIENT_HOR);

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                state = VALUE_LABEL_DRAW_INIT;
                goto chrt_value_label_draw_init;
            }
            else
            {
                state = DATA_DRAW_INIT;
            }

            /*============================================================================*/
            //                    Draw the bars representing the data/samples
            /*============================================================================*/
    chrt_data_draw_init:

        case DATA_DRAW_INIT:

            /* pSmple - points to the sample data of the variables
               ctr - the sample counter
               varCtr - variable counter
               temp - the width of the bars
               dTemp - the height of the bars
               x or y - the location of the first bar per sample. For single variables
                   there will be only one bar per sample. x for vertical bars and y for
horizontal bars.
            */
            ctr = 0;
            temp = splDelta / (2 + ChGetShowSeriesCount(pCh));  // <---- note this! this
can be used to calculate the minimum size limit of the chart
            state = DATA_DRAW_SET;

    chrt_data_draw_set:

        case DATA_DRAW_SET:
            varCtr = 0;

            pVar = ChGetNextShowData(pCh->pChData);

            // set the position to start bar drawing
            // x and y here are used in horizontal drawing and vertical drawing as the
variable
            // that refers to the position of the bar being drawn.
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                    #ifdef USE_HORZ_ASCENDING_ORDER
                y = yStart - temp - ((ChGetSampleRange(pCh) - ctr) * splDelta);
                    #else
                y = yStart - (ctr * splDelta) - temp;
                    #endif
            }
            else
            {
                x = xStart + (ctr * splDelta) + temp;
            }

            state = BAR_DATA_DRAW;

    chrt_data_draw:

        case BAR_DATA_DRAW:

            // get sample data from current variable
            pSmple = (&(*pVar->pData) + (ctr + ChGetSampleStart(pCh) - 1));

            // calculate the total value of the samples to compute the percentages
            if(GetState(pCh, CH_PERCENT))
            {
                j = ChGetSampleStart(pCh);
                samplesMax = 0;
```

```
                while(j <= (ChGetSampleRange(pCh) + ChGetSampleStart(pCh)))
                {
                    samplesMax += (*(&(*pVar->pData) + (j - 1)));
                    j++;
                }

                // Get the percentage of the sample
                dTemp = ((DWORD) (*pSmple) * 100) / samplesMax;

                // check if scaling is needed
                if(ChGetPercentMax(pCh) <= dTemp)
                    dTemp = ChGetPercentRange(pCh);
                else
                {
                    if(dTemp < ChGetPercentMin(pCh))
                        dTemp = 0;
                    else
                        dTemp = dTemp - ChGetPercentMin(pCh);
                }

                dTemp = ((DWORD) (dTemp) * (valDelta * (CH_YGRIDCOUNT - 1))) /
(ChGetPercentRange(pCh));
            }
            else
            {

                // get the height of the current bar to draw
                // this should be adjusted to the min and max set values
                if(ChGetValueMax(pCh) <= (*pSmple))
                {
                    dTemp = ChGetValueRange(pCh);
                }
                else
                {
                    if((*pSmple) < ChGetValueMin(pCh))
                        dTemp = 0;
                    else
                        dTemp = (*pSmple) - ChGetValueMin(pCh);
                }

                dTemp = ((DWORD) (dTemp) * (valDelta * (CH_YGRIDCOUNT - 1)) /
ChGetValueRange(pCh));
            }

            // draw the front side of the bar
            if(ChGetShowSeriesCount(pCh) > 1)
            {
                SetColor(*(&(*pCh->prm.pColor) + varCtr));
            }
            else
            {
                SetColor(*(&(*pCh->prm.pColor) + ctr));
            }

            if(GetState(pCh, CH_3D_ENABLE))
            {
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if(!Bar(xStart, y - (temp * (varCtr + 1)), xStart + dTemp, y - (temp *
varCtr)))
                        return (0);
                }
                else
                {
                    if(!Bar(x + 1 + 1 + (temp * varCtr), yStart - dTemp, x + (temp *
(varCtr + 1)), yStart))
                        return (0);
                }

                state = BAR_DATA_DRAW_3D_PREP;
            }
            else
```

```
            {
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    if(!Bar(xStart, y - (temp * (varCtr + 1)), xStart + dTemp, y - 1 -
(temp * varCtr)))
                        return (0);
                }
                else
                {
                    if(!Bar(x + 1 + (temp * varCtr), yStart - dTemp, x + (temp * (varCtr +
1)), yStart))
                        return (0);
                }

                if((GetState(pCh, CH_VALUE)) || (GetState(pCh, CH_PERCENT)))
                {
                    state = BAR_DATA_DRAW_VALUE;
                    goto chrt_bar_data_draw_value;
                }
                else
                {
                    state = BAR_DATA_DRAW_CHECK;
                    goto chrt_bar_data_draw_check;
                }
            }

        case BAR_DATA_DRAW_3D_PREP:

            // draw the 3-D component
            // draw the 45 degree lines
            // we will use y here as the variable to move the line drawn
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                z = y;
                x = xStart + dTemp;
            }
            else
            {
                z = x + 1;
                y = yStart - dTemp;
            }

            state = BAR_DATA_DRAW_3D_LOOP_1;

    chrt_bar_data_draw_3d_loop_1:

        case BAR_DATA_DRAW_3D_LOOP_1:
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if(x <= xStart + dTemp + barDepth)
                {
                    if((x == (xStart + dTemp)) || (x == (xStart + dTemp + barDepth)))
                    {
                        SetColor(BLACK);
                    }
                    else
                    {
                        if(ChGetShowSeriesCount(pCh) > 1)
                        {
                            SetColor(GetColorShade(*(&(*pCh->prm.pColor) + varCtr), 2));
                        }
                        else
                        {
                            SetColor(GetColorShade(*(&(*pCh->prm.pColor) + ctr), 2));
                        }
                    }

                    if(!Bar(x, z - (temp * (varCtr + 1)), x, z - (temp * varCtr)))
                        return (0);

                    state = BAR_DATA_DRAW_3D_LOOP_2;
                }
```

```
                else
                {
                    state = BAR_DATA_DRAW_3D_OUTLINE1;
                    goto chrt_bar_data_draw_3d_outline_1;
                }
            }
            else
            {
                if(y >= yStart - dTemp - barDepth)
                {
                    if((y == (yStart - dTemp)) || (y == (yStart - dTemp - barDepth)))
                    {
                        SetColor(BLACK);
                    }
                    else
                    {
                        if(ChGetShowSeriesCount(pCh) > 1)
                        {
                            SetColor(GetColorShade(*(&(*pCh->prm.pColor) + varCtr), 2));
                        }
                        else
                        {
                            SetColor(GetColorShade(*(&(*pCh->prm.pColor) + ctr), 2));
                        }
                    }

                    if(!Bar(z + 1 + (temp * varCtr), y, z - 1 + (temp * (varCtr + 1)), y))
                        return (0);

                    state = BAR_DATA_DRAW_3D_LOOP_2;
                }
                else
                {
                    state = BAR_DATA_DRAW_3D_OUTLINE1;
                    goto chrt_bar_data_draw_3d_outline_1;
                }
            }

        case BAR_DATA_DRAW_3D_LOOP_2:

            // check if we are going to draw the outline or the shade
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if((x == (xStart + dTemp)) || (x == (xStart + dTemp + barDepth)))
                {
                    SetColor(BLACK);
                }
                else
                {
                    if(ChGetShowSeriesCount(pCh) > 1)
                    {
                        SetColor(GetColorShade(*(&(*pCh->prm.pColor) + varCtr), 1));
                    }
                    else
                    {
                        SetColor(GetColorShade(*(&(*pCh->prm.pColor) + ctr), 1));
                    }
                }
            }
            else
            {
                if((y == (yStart - dTemp)) || (y == (yStart - dTemp - barDepth)))
                {
                    SetColor(BLACK);
                }
                else
                {
                    if(ChGetShowSeriesCount(pCh) > 1)
                    {
                        SetColor(GetColorShade(*(&(*pCh->prm.pColor) + varCtr), 1));
                    }
                    else
```

```
                    {
                        SetColor(GetColorShade(*(&(*pCh->prm.pColor) + ctr), 1));
                    }
                }
            }

            // draw the outline or shade
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if(!Bar(x - dTemp, z - (temp * (varCtr + 1)), x, z - (temp * (varCtr + 1))))
                    return (0);
            }
            else
            {
                if(!Bar(z + (temp * (varCtr + 1)), y, z + (temp * (varCtr + 1)), y + dTemp))
                    return (0);
            }

            // update the loop variables
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                x++;
                z--;
            }
            else
            {
                y--;
                z++;
            }

            state = BAR_DATA_DRAW_3D_LOOP_1;
            goto chrt_bar_data_draw_3d_loop_1;

    chrt_bar_data_draw_3d_outline_1:

        case BAR_DATA_DRAW_3D_OUTLINE1:
            SetColor(BLACK);

            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if(!Line(x - 1, y - (temp * varCtr) - barDepth, x - 1 - barDepth, y - (temp
* varCtr)))
                    return (0);
            }
            else
            {
                if
                (
                    !Line
                        (
                            x + 1 + (temp * varCtr),
                            yStart - dTemp,
                            x + 1 + (temp * varCtr) + barDepth,
                            yStart - dTemp - barDepth
                        )
                ) return (0);
            }

            state = BAR_DATA_DRAW_3D_OUTLINE2;

        case BAR_DATA_DRAW_3D_OUTLINE2:
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if(!Line(x - 1, y - (temp * (varCtr + 1)) - barDepth, x - 1 - barDepth, y -
(temp * (varCtr + 1))))
                    return (0);
            }
            else
            {
                if
                (
                    !Line
```

```
                        (
                            x + 1 + (temp * (varCtr + 1)),
                            (yStart - dTemp),
                            x + 1 + (temp * (varCtr + 1)) + barDepth,
                            (yStart - dTemp) - barDepth
                        )
                ) return (0);
            }

            state = BAR_DATA_DRAW_3D_OUTLINE3;

        case BAR_DATA_DRAW_3D_OUTLINE3:
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if
                (
                    !Line
                        (
                            x - dTemp - 1,
                            y - (temp * (varCtr + 1)) - barDepth,
                            x - dTemp - barDepth - 1,
                            y - (temp * (varCtr + 1))
                        )
                ) return (0);
            }
            else
            {
                if
                (
                    !Line
                        (
                            x + 1 + (temp * (varCtr + 1)),
                            yStart,
                            x + 1 + (temp * (varCtr + 1)) + barDepth,
                            yStart - barDepth
                        )
                ) return (0);
            }

            state = BAR_DATA_DRAW_3D_OUTLINE4;

        case BAR_DATA_DRAW_3D_OUTLINE4:
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if
                (
                    !Bar
                        (
                            x - dTemp - barDepth - 1,
                            y - (temp * (varCtr + 1)),
                            x - dTemp - barDepth - 1,
                            y - (temp * (varCtr + 1)) + temp
                        )
                ) return (0);
            }
            else
            {
                if(!Bar(x + 1 + (temp * varCtr), yStart - dTemp, x + 1 + (temp * varCtr),
yStart))
                    return (0);
            }

            state = BAR_DATA_DRAW_3D_OUTLINE5;

        case BAR_DATA_DRAW_3D_OUTLINE5:

            // draw the horizontal lines
            if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
            {
                if
                (
                    !Bar
```

```
                        (
                            x - dTemp - barDepth - 1,
                            y - (temp * (varCtr + 1)) + temp,
                            x - barDepth - 1,
                            y - (temp * (varCtr + 1)) + temp
                        )
                ) return (0);
            }
            else
            {
                if(!Bar(x + 1 + (temp * varCtr), yStart, x + 1 + (temp * (varCtr + 1)),
yStart))
                    return (0);
            }

            if((GetState(pCh, CH_VALUE)) || (GetState(pCh, CH_PERCENT)))
                state = BAR_DATA_DRAW_VALUE;
            else
            {
                state = BAR_DATA_DRAW_CHECK;
                goto chrt_bar_data_draw_check;
            }

    chrt_bar_data_draw_value:

        case BAR_DATA_DRAW_VALUE:
            if(GetState(pCh, CH_VALUE))
            {
                j = word2xchar(*pSmple, tempStr, STR_CHAR_CNT);
            }
            else
            {

                // add the percent sign on the label
                tempStr[STR_CHAR_CNT - 1] = '%';

                // compute for the percentage
                if(ChGetValueMax(pCh) <= (*pSmple))
                    dPercent = ChGetValueRange(pCh);
                else
                {
                    if((*pSmple) < ChGetValueMin(pCh))
                        dPercent = 0;
                    else
                        dPercent = (*pSmple) - ChGetValueMin(pCh);
                }

                dPercent = ((DWORD) (dPercent * 1000)) / samplesMax;

                // check if we need to round up or not
                if((dPercent % 10) < 5)
                    dPercent = (dPercent / 10);                 // do not round up to next
number
                else
                    dPercent = (dPercent / 10) + 1;             // round up the value

                // we have a plus 1 here since we add '%' sign already
                j = 1 + word2xchar(dPercent, tempStr, STR_CHAR_CNT - 1);
            }

            if(GetState(pCh, CH_3D_ENABLE))
            {
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    MoveTo
                    (
                        xStart + dTemp + barDepth + 3,
                        y - (temp * varCtr) - ((temp +
GetTextHeight(pCh->hdr.pGolScheme->pFont)) >> 1) - barDepth
                    );
                }
                else
```

```
                {
                    MoveTo
                    (
                        x + barDepth + (temp * varCtr) + (temp >> 1) -
(GetTextWidth(&tempStr[STR_CHAR_CNT - j], pCh->hdr.pGolScheme->pFont) >> 1),
                        (yStart - 1) - dTemp - GetTextHeight(pCh->hdr.pGolScheme->pFont) -
(barDepth)
                    );
                }
            }
            else
            {
                if(GetState(pCh, CH_BAR_HOR) == CH_BAR_HOR)
                {
                    MoveTo
                    (
                        xStart + dTemp + 3,
                        y - (temp * varCtr) - (temp >> 1) -
(GetTextHeight(pCh->hdr.pGolScheme->pFont) >> 1)
                    );
                }
                else
                {
                    MoveTo
                    (
                        x +
                            ((temp >> 1) - (GetTextWidth(&tempStr[STR_CHAR_CNT - j],
pCh->hdr.pGolScheme->pFont) >> 1)) +
                            (temp * varCtr),
                        (yStart - 1) - dTemp - (GetTextHeight(pCh->hdr.pGolScheme->pFont))
                    );
                }
            }

            state = BAR_DATA_DRAW_VALUE_RUN;

        case BAR_DATA_DRAW_VALUE_RUN:

            // draw the values on top of the bars
            // NOTE: we use the emboss light color here to draw the values.
            SetColor(pCh->hdr.pGolScheme->EmbossLtColor);
            SetFont(pCh->hdr.pGolScheme->pFont);
            if(!OutText(&tempStr[STR_CHAR_CNT - j]))
                return (0);
            state = BAR_DATA_DRAW_CHECK;

    chrt_bar_data_draw_check:

        case BAR_DATA_DRAW_CHECK:

            // find the next data series that will be shown
            pVar = ChGetNextShowData(pVar);
            if(pVar == NULL)
            {
                state = REMOVE;
                return (1);
            }

            // increment the variable counter
            varCtr++;
            if(varCtr < ChGetShowSeriesCount(pCh))
            {
                pVar = (DATASERIES *)pVar->pNextData;
                state = BAR_DATA_DRAW;
                goto chrt_data_draw;
            }

            // increment the sample counter
            ctr++;
            if(ctr < ChGetSampleRange(pCh) + 1)
            {
                x += (splDelta + 1);
```

```
                state = DATA_DRAW_SET;
                goto chrt_data_draw_set;
            }

            state = REMOVE;
            return (1);

            /*************************************************************************/
            //                    PIE CHART states
            /*************************************************************************/

            /*=========================================================================*/
            //                      Draw the pie
            /*=========================================================================*/
chrt_pie_prep:

        case PIE_PREP:

            /* If more than two data series have their SHOW_DATA flag set
               the pie chart is drawn to represent the values of each variable
               at Start sample point . End sample point is ignored in this case.
               If only one data series is set to be shown the pie chart is drawn
               from the sample start point and sample end point (inclusive).

               Pie chart is drawn as a percentage of the data samples.
               Therefore: percentage is computed depending on the sample
               points used.
               Only 1 data series is set to be shown
                 total =    summation of the variable sample points from
                            sample start point to sample end point inclusive.
               More than 1 data series is set to be shown
                 total =    summation of the sample points for each variable
                            using the sample start point.
            */

            /* For PIE chart variables used are the following
                ctr = x position of the center of the pie chart
                ctry = y position of the center of the pie chart
                z   = radius of the pie chart
                j   = start angle
                k   = end angle
            */

            // calculate the needed variables
            // radius z is affected by the following: CH_LEGEND, CH_VALUE and CH_PERCENT
            temp = CH_MARGIN << 1;

            // check the largest/longest sample
            // use x here as a counter
            varCtr = ChGetShowSeriesCount(pCh);
            if(varCtr == 1)
            {
                varCtr = ChGetSampleRange(pCh) + 1;
            }

            pVar = ChGetNextShowData(pCh->pChData);

            // y and z is used here as a temporary variable
            y = 0;

            // find the sample value that is largest (place in y)
            // also while doing this get the total of the selected data (put in dTemp)
            dTemp = 0;
            while(varCtr >= 0)
            {
                if(ChGetShowSeriesCount(pCh) > 1)
                {
                    z = *(&(*pVar->pData) + ChGetSampleStart(pCh) - 1);
                }
                else
                {
                    z = *(&(*pVar->pData) + ChGetSampleEnd(pCh) - varCtr);
```

```
                }

                dTemp += z;
                varCtr--;

                // check if we get a larger value
                if(z > y)
                    y = z;
                if(varCtr == 0)
                    break;

                if(ChGetShowSeriesCount(pCh) > 1)
                {
                    pVar = ChGetNextShowData((DATASERIES *)pVar->pNextData);
                }

                // this is insurance (in case the link list is corrupted)
                if(pVar == NULL)
                {
                    break;
                }
            }

            // initialize pVarFont to use pGridLabelsFont
            pVarFont = pCh->prm.pGridLabelsFont;

            // get the space occupied by the value
            if(GetState(pCh, CH_VALUE))
            {
                z = word2xchar(y, tempStr, STR_CHAR_CNT);
                x = (GetTextWidth(&tempStr[STR_CHAR_CNT - z], pVarFont));
            }
            else
                x = 0;

            // get the space occupied by percent value
            if(GetState(pCh, CH_PERCENT))
            {

                // 7 is derived from "X:100%," - seven possible characters, 1 is added for
buffer
                y = (8 * GetTextWidth((XCHAR *)tempXchar, pVarFont));
            }
            else
                y = 0;

            // get the space occupied by the legend
            if((GetState(pCh, CH_LEGEND)) && (ChGetShowSeriesCount(pCh) > 1))
                temp += ((GetLongestNameLength(pCh) +
(GetTextHeight(pCh->hdr.pGolScheme->pFont) >> 1)));

            // calculate the center of the pie chart
            ctr = (pCh->hdr.left + pCh->hdr.right - temp) >> 1;
            ctry = ((pCh->hdr.bottom + (pCh->hdr.top + GetTextHeight(pVarFont))) >> 1) +
CH_MARGIN;

            // radius size is checked against the x and y area
            if
            (
                ((pCh->hdr.right - pCh->hdr.left) - temp - ((x + y) << 1)) <
                    (
                        (pCh->hdr.bottom - pCh->hdr.top -
GetTextHeight(pCh->prm.pTitleFont)) - temp -
                            (GetTextHeight(pVarFont) << 1)
                    )
            )
            {

                // use dimension in the x axis
                if(x + y)
                {
                    z = ctr - (pCh->hdr.left + (x + y) + CH_MARGIN);
```

```
                }
                else
                {
                    z = ctr - pCh->hdr.left + CH_MARGIN;
                }
            }
            else
            {

                // use dimension in the y axis
                if(x + y)
                    z = ctry -
                        (
                            pCh->hdr.top + (CH_MARGIN << 1) +
GetTextHeight(pCh->prm.pTitleFont) +
                                (GetTextHeight(pVarFont) << 1)
                        );
                else
                    z = ctry - (pCh->hdr.top + (CH_MARGIN << 1) +
(GetTextHeight(pCh->prm.pTitleFont) << 1));
            }

            state = PIE_DRAW_OUTLINE1;

        case PIE_DRAW_OUTLINE1:

            // Required items before the pie chart can be drawn
            SetColor(LIGHTGRAY);

            // Draw pie-chart outline
            if(!Circle(ctr, ctry, z))
                return (0);
            state = PIE_DRAW_SECTOR;

        /*=========================================================================*/
        //                    Draw the sectors of the pie
        /*=========================================================================*/
        case PIE_DRAW_SECTOR:

            // now we are ready to draw the sectors
            // calculate the sector that a value will need
            k = 0;

            // check if more than one data series set to be shown , draw the pie chart of
            // the data series that are set to be shown.
            pVar = ChGetNextShowData(pCh->pChData);
            y = dTemp;
            varCtr = ChGetShowSeriesCount(pCh);
            if(varCtr == 1)
            {
                varCtr = ChGetSampleRange(pCh) + 1;
            }
            else if(varCtr == 0)
            {
                pVar = NULL;
                y = 0;
            }

            // we start at 0 degree.
            j = 0;

            // initialize the variables that position the pie sector labels.
            // this is used to minimize the occurrence of overlapped labels.
            pieLabelYPos = ctry;
            pieLabelYPos2 = pCh->hdr.bottom - (GOL_EMBOSS_SIZE + 1) -
GetTextHeight(pVarFont);
            pieLabelYPos3 = pCh->hdr.top + GOL_EMBOSS_SIZE + CH_MARGIN +
GetTextHeight(pCh->prm.pTitleFont) + GetTextHeight(pVarFont);

            state = PIE_DRAW_SECTOR_LOOP;

    chrt_pie_draw_sector_loop:
```

```
        case PIE_DRAW_SECTOR_LOOP:
            if(varCtr >= 0)
            {

                // get the value to be computed
                if(ChGetShowSeriesCount(pCh) > 1)
                {
                    temp = *(&(*pVar->pData) + ChGetSampleStart(pCh) - 1);
                }
                else
                {
                    temp = *(&(*pVar->pData) + ChGetSampleEnd(pCh) - varCtr);
                }

                // calculate the sector that the value will occupy
                dTemp = ((DWORD) (temp) * (3600)) / y;

                // check if we need to round up or not
                if((dTemp % 10) < 5)
                    dTemp = (dTemp / 10);                       // do not round up to next
number
                else
                    dTemp = (dTemp / 10) + 1;                   // round up the value

                // set the color to the color of the variable
                SetColor(*(&(*pCh->prm.pColor) + k));

                // check if the sector has zero angle if it is zero just draw the
                // line.
                if(dTemp == 0)
                {
                    state = PIE_DRAW_SECTOR_LOOP_CONTINUE;
                    goto chrt_pie_draw_sector_loop_continue;
                }

                // go to the state that draws only. Doing this separates the setup of
static variables
                // and rendering. So in cases when the rendering cannot continue, the
variables
                // are still set to correct values.
                state = PIE_DRAW_SECTOR_ACTUAL;
                goto pie_draw_sector_actual;
            }
            else
            {
                if(GetState(pCh, CH_DONUT) == CH_DONUT)
                {
                    state = PIE_DONUT_HOLE_DRAW;
                    goto chrt_pie_donut_hole_draw;
                }
                else
                {
                    state = REMOVE;
                    return (1);
                }
            }

        case PIE_DRAW_SECTOR_ACTUAL:
            pie_draw_sector_actual :

            // check if it is the last sector to be drawn
            if((varCtr == 1) || ((j + dTemp) >= 358))
            {
                if(!DrawSector(ctr, ctry, z, j, 360, LIGHTGRAY))
                    return (0);
            }
            else
            {
                if(!DrawSector(ctr, ctry, z, j, (j + dTemp), LIGHTGRAY))
                    return (0);
            }
```

```
            state = PIE_DRAW_SECTOR_LOOP_CREATE_STRINGS;

        case PIE_DRAW_SECTOR_LOOP_CREATE_STRINGS:

            // create the strings of the values if needed
            if(GetState(pCh, CH_VALUE) || GetState(pCh, CH_PERCENT))
            {
                h = 0;
                GetCirclePoint(z, (j + (dTemp >> 1)), &pieX, &pieY);

                pieX += ctr;
                pieY += ctry;

                // do we need to show the values? create the strings here
                if(GetState(pCh, CH_VALUE))
                {
                    h = word2xchar(temp, tempStr, STR_CHAR_CNT);
                }

                // do we need to show the percentage? create the strings here
                if(GetState(pCh, CH_PERCENT))
                {

                    // add the % sign
                    // check if we need to add comma
                    if(GetState(pCh, CH_VALUE))
                    {
                        h += 1;                          // adjust h
                        tempStr[STR_CHAR_CNT - h] = ',';
                    }

                    h += 1;                              // adjust h
                    tempStr[STR_CHAR_CNT - h] = '%';

                    // now add the percentage
                    dPercent = (DWORD) (dTemp * 1000) / 360;

                    // check if we need to round up or not
                    if((dPercent % 10) < 5)
                        dPercent = (dPercent / 10);     // do not round up to next number
                    else
                        dPercent = (dPercent / 10) + 1; // round up the value
                    i = word2xchar((WORD) dPercent, tempStr, STR_CHAR_CNT - h);

                    // adjust the h position
                    h += i;
                }

                    #ifdef USE_PIE_ENABLE_LABEL

                // add the labels
                h += 1; // adjust h
                tempStr[STR_CHAR_CNT - h] = ':';
                h += 1; // adjust h
                tempStr[STR_CHAR_CNT - h] = 'A' + (ChGetSampleStart(pCh) - 1 + k);
                    #endif
                SetColor(pCh->hdr.pGolScheme->EmbossLtColor);

                m = j;

                // we have to relocate the text depending on the position
                if((m + (dTemp >> 1) >= 0) && (m + (dTemp >> 1) <= 90))
                {

                    // check if we need to draw a line
                    if((dTemp < GetTextHeight(pVarFont)) || (pieLabelYPos > pieY))
                    {
                        pieLabelXPos = ctr + z + GetTextHeight(pVarFont);
                        pieSectorXPos = pieX + 3;
                        if((m + (dTemp >> 1)) < 45)
                            pieSectorYPos = pieY + 1;
```

```
                        else
                            pieSectorYPos = pieY + 3;

                        // The label will now exceed the chart dimension, so force the
value to be printed near the sector
                        pieLabelXPos = ctr + z + GetTextHeight(pVarFont);
                        if((pieLabelYPos + GetTextHeight(pVarFont)) > (pCh->hdr.bottom -
(GOL_EMBOSS_SIZE + 1)))
                        {
                            MoveTo(pieX, pieY + 1);
                        }
                        else
                        {

                            // draw the line
                            if
                            (
                                !Line
                                    (
                                        pieSectorXPos,
                                        pieSectorYPos,
                                        pieLabelXPos,
                                        pieLabelYPos + (GetTextHeight(pVarFont) >> 1)
                                    )
                            ) return (0);

                            MoveTo(pieLabelXPos, pieLabelYPos);
                            pieLabelYPos += GetTextHeight(pVarFont);
                        }
                    }
                    else
                    {
                        MoveTo(pieX + GetTextWidth(tempXchar, pVarFont), pieY);
                        pieLabelYPos = pieY + GetTextHeight(pVarFont);
                    }
                }
                else if((m + (dTemp >> 1) > 90) && (m + (dTemp >> 1) <= 180))
                {

                    // check if we need to draw a line
                    if((dTemp < GetTextHeight(pVarFont)) || (pieLabelYPos2 < pieY))
                    {
                        pieLabelXPos = ctr - z - 3;

                        pieSectorXPos = pieX - 3;

                        if((m + (dTemp >> 1)) < 135)
                            pieSectorYPos = pieY + 3;
                        else
                            pieSectorYPos = pieY;

                        // check if slope of line is greater than -1.
                        // if it is we must adjust position of text to avoid the line
                        // intersecting the circumference of the pie chart
                        if((m + (dTemp >> 1)) < 180)
                        {

                            // make the slope equal to 1, this will make sure it does not
intersect the circumference
                            if((abs(pieY - pieLabelYPos2) / abs(pieX - pieLabelXPos)) >= 1)
                            {
                                pieLabelXPos = pieX - abs(pieY - pieLabelYPos2);
                                if
                                (
                                    (SHORT) (pieLabelXPos -
GetTextWidth(&tempStr[STR_CHAR_CNT - h], pVarFont)) <=
                                        (pCh->hdr.left + CH_MARGIN)
                                )
                                {
                                    pieLabelXPos =
                                        (
                                            pCh->hdr.left +
```

```
                                            CH_MARGIN +
                                            GetTextWidth(&tempStr[STR_CHAR_CNT - h],
pVarFont)
                                        );
                                }
                            }
                        }

                        // draw the line
                        if
                        (
                            !Line
                                (
                                    pieSectorXPos,
                                    pieSectorYPos,
                                    pieLabelXPos,
                                    pieLabelYPos2 + (GetTextHeight(pVarFont) >> 1)
                                )
                        ) return (0);

                        MoveTo(pieLabelXPos - GetTextWidth(&tempStr[STR_CHAR_CNT - h],
pVarFont), pieLabelYPos2);
                        pieLabelYPos2 -= GetTextHeight(pVarFont);
                    }
                    else
                    {
                        MoveTo
                        (
                            pieX - GetTextWidth(&tempStr[STR_CHAR_CNT - h], pVarFont) -
GetTextWidth
                                (
                                    tempXchar,
                                    pVarFont
                                ),
                            pieY
                        );
                        pieLabelYPos2 = pieY - GetTextHeight(pVarFont);
                    }
                }
                else if((m + (dTemp >> 1) > 180) && (m + (dTemp >> 1) <= 270))
                {

                    // check if we need to draw a line
                    if((dTemp < GetTextHeight(pVarFont)) || (pieLabelYPos2 < pieY))
                    {
                        pieLabelXPos = ctr - z - 5;

                        if((m + (dTemp >> 1)) < 225)
                        {
                            pieSectorXPos = pieX - 3;
                            pieSectorYPos = pieY;
                        }
                        else
                        {
                            pieSectorXPos = pieX;
                            pieSectorYPos = pieY - 3;
                        }

                        // check if slope of line is greater than -1.
                        // if it is we must adjust position of text to avoid the line
                        // intersecting the circumference of the pie chart
                        if((m + (dTemp >> 1)) < 270)
                        {

                            // make the slope equal to 1, this will make sure it does not
intersect the circumference
                            if((abs(pieY - pieLabelYPos2) / abs(pieX - pieLabelXPos)) >= 1)
                            {
                                pieLabelXPos = ctr - (z + (z >> 1)) - 5;
                                if
                                (
                                    (SHORT) (pieLabelXPos -
```

```
GetTextWidth(&tempStr[STR_CHAR_CNT - h], pVarFont)) <=
                                        (pCh->hdr.left + CH_MARGIN)
                                )
                                {
                                    pieLabelXPos =
                                        (
                                            pCh->hdr.left +
                                            CH_MARGIN +
                                            GetTextWidth(&tempStr[STR_CHAR_CNT - h],
pVarFont)
                                        );
                                }
                            }
                        }

                        // The label will now exceed the chart dimension, so force the
value to be printed aligned in
                        // y position with the previous value printed
                        if
                        (
                            (pieLabelYPos2) <
                                (pCh->hdr.top + GOL_EMBOSS_SIZE + CH_MARGIN +
GetTextHeight(pCh->prm.pTitleFont))
                        )
                        {
                            if
                            (
                                !Line
                                    (
                                        pieSectorXPos,
                                        pieSectorYPos,
                                        pieLabelXPos + (GetTextWidth(&tempStr[STR_CHAR_CNT
- h], pVarFont) >> 1),
                                        pieLabelYPos2 + ((GetTextHeight(pVarFont) >> 1) * 3)
                                    )
                            ) return (0);

                            MoveTo
                            (
                                pieLabelXPos + (GetTextWidth(&tempStr[STR_CHAR_CNT - h],
pVarFont) >> 1),
                                pieLabelYPos2 + GetTextHeight(pVarFont)
                            );

                            // adjust the next marker
                            pieLabelYPos3 += GetTextHeight(pVarFont);
                        }
                        else
                        {

                            // draw the line
                            if
                            (
                                !Line
                                    (
                                        pieSectorXPos,
                                        pieSectorYPos,
                                        pieLabelXPos,
                                        pieLabelYPos2 + (GetTextHeight(pVarFont) >> 1)
                                    )
                            ) return (0);

                            MoveTo(pieLabelXPos - GetTextWidth(&tempStr[STR_CHAR_CNT - h],
pVarFont), pieLabelYPos2);
                            pieLabelYPos2 -= GetTextHeight(pVarFont);
                        }
                    }
                    else
                    {
                        MoveTo
                        (
                            pieX - GetTextWidth(&tempStr[STR_CHAR_CNT - h], pVarFont) -
```

```
GetTextWidth
                                (
                                    tempXchar,
                                    pVarFont
                                ),
                            pieY - GetTextHeight(pVarFont)
                        );
                        pieLabelYPos2 = pieY - (GetTextHeight(pVarFont) << 1);
                    }
                }
                else if((m + (dTemp >> 1) > 270) && (m + (dTemp >> 1) <= 360))
                {

                    // check if we need to draw a line
                    if((dTemp < GetTextHeight(pVarFont)) || (pieLabelYPos3 > pieY))
                    {
                        pieLabelXPos = ctr + z + GetTextHeight(pVarFont);

                        pieSectorXPos = pieX + 3;
                        pieSectorYPos = pieY - 2;

                        // draw the line
                        if
                        (
                            !Line
                                (
                                    pieSectorXPos,
                                    pieSectorYPos,
                                    pieLabelXPos,
                                    pieLabelYPos3 - (GetTextHeight(pVarFont) >> 1)
                                )
                        ) return (0);

                        MoveTo(pieLabelXPos, pieLabelYPos3 - GetTextHeight(pVarFont));
                        pieLabelYPos3 += GetTextHeight(pVarFont);
                    }
                    else
                    {
                        MoveTo(pieX + 5, pieY - GetTextHeight(pVarFont));
                        pieLabelYPos3 = pieY + GetTextHeight(pVarFont);
                    }
                }

                state = PIE_DRAW_SECTOR_LOOP_STRINGS_RUN;
            }
            else
            {
                state = PIE_DRAW_SECTOR_LOOP_CONTINUE;
                goto chrt_pie_draw_sector_loop_continue;
            }

        case PIE_DRAW_SECTOR_LOOP_STRINGS_RUN:

            // now draw the strings of the values and/or percentages
            SetColor(pCh->hdr.pGolScheme->EmbossLtColor);
            SetFont(pVarFont);

            if(!OutText(&tempStr[STR_CHAR_CNT - h]))
                return (0);

            state = PIE_DRAW_SECTOR_LOOP_CONTINUE;

    chrt_pie_draw_sector_loop_continue:

        case PIE_DRAW_SECTOR_LOOP_CONTINUE:
            j += dTemp;
            varCtr--;
            if(varCtr == 0)
            {
                if(GetState(pCh, CH_DONUT) == CH_DONUT)
                {
                    state = PIE_DONUT_HOLE_DRAW;
```

```
                    goto chrt_pie_donut_hole_draw;
                }
                else
                {
                    state = REMOVE;
                    return (1);
                }
            }

            // check if more than one data series to be shown
            if(ChGetShowSeriesCount(pCh) > 1)
            {
                pVar = ChGetNextShowData((DATASERIES *)pVar->pNextData);
                if(pVar == NULL)
                {
                    break;
                }
            }

            k++;
            state = PIE_DRAW_SECTOR_LOOP;
            goto chrt_pie_draw_sector_loop;

    chrt_pie_donut_hole_draw:

        case PIE_DONUT_HOLE_DRAW:
            SetColor(LIGHTGRAY);
            if(!Circle(ctr, ctry, (z >> 1) - (z >> 3)))
                return (0);
            SetColor(pCh->hdr.pGolScheme->CommonBkColor);
            if(!FillCircle(ctr, ctry, ((z >> 1) - (z >> 3)) - 1))
                return (0);

            state = REMOVE;
            return (1);
    }

    return (1);
}

/*************************************************************************
* Function: ChAddDataSeries(CHART *pCh, WORD nSamples, WORD *pData, XCHAR *pName)
*
*
* Notes: Adds a new variable data structure in the linked list
*        of variable datas. Number of samples is given with the
*        array of the samples. If there is only one data
*        nSamples is set to 1 with the address of the variable data.
*
*
*************************************************************************/
DATASERIES *ChAddDataSeries(CHART *pCh, WORD nSamples, WORD *pData, XCHAR *pName)
{
    DATASERIES  *pVar = NULL, *pListVar;

    pVar = (DATASERIES *)GFX_malloc(sizeof(DATASERIES));

    if(pVar == NULL)
        return (NULL);

    // add the other parameters of the variable data
    pVar->pSData = (XCHAR *)pName;
    pVar->samples = nSamples;
    pVar->pData = (WORD *)pData;
    pVar->show = SHOW_DATA;
    pVar->pNextData = NULL;

    pListVar = pCh->pChData;
    if(pCh->pChData == NULL)
        pCh->pChData = pVar;
    else
    {
```

```
        // search the end of the list and append the new data
        while(pListVar->pNextData != NULL)
            pListVar = pListVar->pNextData;
        pListVar->pNextData = pVar;
    }

    // update the variable count before exiting
    pCh->prm.seriesCount++;
    return (DATASERIES *)pVar;
}
/*********************************************************************
* Function: void ChFreeDataSeries(void *pObj)
*
*
* Notes: Removes a data series structure in the linked list
*        of data series.
*
*********************************************************************/
void ChFreeDataSeries(void *pObj)
{
    DATASERIES  *pVar = NULL, *pPrevVar = NULL;
    CHART *pCh;

    pCh = (CHART *)pObj;

    pVar = pCh->pChData;

    // check if the list is empty
    if(pVar == NULL)
        return;

    // check if there is only one entry
    if(pVar->pNextData == NULL)
    {
        GFX_free(pVar);
        pCh->pChData = NULL;
        return;
    }

    // remove all
    while(pVar != NULL)
    {
        pPrevVar = pVar;
        pVar = pVar->pNextData;

        // free the memory used by the item
        GFX_free(pPrevVar);
    }
}

/*********************************************************************
* Function: ChRemoveDataSeries(CHART *pCh, WORD number)
*
*
* Notes: Removes a data series structure in the linked list
*        of data series.
*
*********************************************************************/
void ChRemoveDataSeries(CHART *pCh, WORD number)
{
    DATASERIES  *pVar = NULL, *pPrevVar = NULL;
    WORD        ctr = 1;

    pVar = pCh->pChData;

    // check if the list is empty
    if(pVar == NULL)
        return;

    // check if there is only one entry
    if(pVar->pNextData == NULL)
```

```
    {
        GFX_free(pVar);
        pCh->pChData = NULL;
        return;
    }

    if(number == 0)
    {

        // remove all
        while(pVar != NULL)
        {
            pPrevVar = pVar;
            pVar = pVar->pNextData;

            // free the memory used by the item
            GFX_free(pPrevVar);
        }

        return;
    }

    // there are more than one entry, remove the entry specified
    while(ctr < number)
    {
        pPrevVar = pVar;
        pVar = pVar->pNextData;
        ctr++;
    }

    // remove the item from the list
    pPrevVar->pNextData = pVar->pNextData;

    // free the memory used by the item
    GFX_free(pVar);

    return;
}

/*********************************************************************
* Function: ChSetDataSeries(CHART *pCh, WORD seriesNum, BYTE status)
*
*
* Notes: Sets the specified data series number show flag to be set to
*        SHOW_DATA or HIDE_DATA depending on the status.
*        If the seriesNum is 0, it sets all the data series
*        entries in the data series linked list. Returns the same passed
*        number if successful otherwise -1 if unsuccesful.
*
*********************************************************************/
SHORT ChSetDataSeries(CHART *pCh, WORD seriesNum, BYTE status)
{
    DATASERIES  *pListSer;
    WORD        ctr = 1;

    pListSer = pCh->pChData;

    // check if the list is empty
    if(pListSer == NULL)
        return (-1);

    while(pListSer != NULL)
    {

        // check if we need to show all
        if(seriesNum == 0)
            pListSer->show = status;
        else if(seriesNum == ctr)
        {
            pListSer->show = status;
            break;
        }
```

```
        ctr++;
        pListSer = pListSer->pNextData;
    }

    if(seriesNum == ctr)
        return (seriesNum);
    else
        return (-1);
}

/*****************************************************************
* Function: ChSetSampleRange(CHART *pCh, WORD start, WORD end)
*
*
* Notes: Sets the sampling start and end points when drawing the chart.
*        Depending on the number of data series with SHOW_DATA flag
*        set and the values of end and start samples a single
*        data series is drawn or multiple data series are drawn.
*
*****************************************************************/
void ChSetSampleRange(CHART *pCh, WORD start, WORD end)
{
    pCh->prm.smplStart = start;
    if(end < start)
        pCh->prm.smplEnd = start;
    else
        pCh->prm.smplEnd = end;
}

/*****************************************************************
* Function: ChSetValueRange(CHART *pCh, WORD min, WORD max)
*
*
* Notes: Sets the sampling start and end points when drawing the chart.
*        Depending on the number of data series with SHOW_DATA flag
*        set and the values of end and start samples a single
*        data series is drawn or multiple data series are drawn.
*
*****************************************************************/
void ChSetValueRange(CHART *pCh, WORD min, WORD max)
{
    pCh->prm.valMin = min;
    if(max < min)
        pCh->prm.valMax = min;
    else
        pCh->prm.valMax = max;
}

/*****************************************************************
* Function: ChSetPercentRange(CHART *pCh, WORD min, WORD max)
*
*
* Notes: Sets the percentage range when drawing the chart. This affects
*        bar charts only and CH_PERCENTAGE bit state is set.
*
*****************************************************************/
void ChSetPercentRange(CHART *pCh, WORD min, WORD max)
{
    pCh->prm.perMin = min;
    if(max < min)
        pCh->prm.perMax = min;
    else
        pCh->prm.perMax = max;
}


/* */

void FillSector(SHORT x, SHORT y, GFX_COLOR outLineColor)
{
```

```
    GFX_COLOR    pixel;
    SHORT   left, right;
    SHORT   top, bottom;
    SHORT   xc, yc;

    // scan down
    top = bottom = yc = y;
    left = right = xc = x;
    while(1)
    {

        pixel = GetPixel(xc, yc);
        if(pixel == outLineColor)
        {
            for(xc = left + 1; xc < right; xc++)
            {
                pixel = GetPixel(xc, yc);
                if(pixel != outLineColor)
                {
                    break;
                }
            }

            if(xc == right)
                break;
        }

        // left scan
        left = xc;
        do
        {
            left--;
            pixel = GetPixel(left, yc);
        } while(pixel != outLineColor);
        while(!Line(xc, yc, left + 1, yc));

        // right scan
        right = xc;
        pixel = GetPixel(right, yc);
        do
        {
            right++;
            pixel = GetPixel(right, yc);
        } while(pixel != outLineColor);
        while(!Line(xc, yc, right - 1, yc));

        xc = (left + right) >> 1;
        yc++;
    }

    // scan up
    yc = y;
    xc = x;
    while(1)
    {
        pixel = GetPixel(xc, yc);
        if(pixel == outLineColor)
        {
            for(xc = left + 1; xc < right; xc++)
            {
                pixel = GetPixel(xc, yc);
                if(pixel != outLineColor)
                {
                    break;
                }
            }

            if(xc == right)
                break;
        }

        // left scan
```

```
        left = xc;
        do
        {
            left--;
            pixel = GetPixel(left, yc);
        } while(pixel != outLineColor);
        while(!Line(xc, yc, left + 1, yc));

        // right scan
        right = xc;
        pixel = GetPixel(right, yc);
        do
        {
            right++;
            pixel = GetPixel(right, yc);
        } while(pixel != outLineColor);
        while(!Line(xc, yc, right - 1, yc));

        xc = (left + right) >> 1;
        yc--;
    }
}

/* */


/* */
WORD DrawSector(SHORT cx, SHORT cy, SHORT outRadius, SHORT angleFrom, SHORT angleTo,
GFX_COLOR outLineColor)
{
    typedef enum
    {
        SEC_DRAW_IDLE,
        SEC_DRAW_EDGE1,
        SEC_DRAW_EDGE2,
        SEC_DRAW_EDGE3,
        SEC_DRAW_EDGE4,
        SEC_DRAW_EDGE5,
        SEC_DRAW_FILLINIT,
        SEC_DRAW_FILL,
    } DRAW_SECTOR_STATES;

    static SHORT x1, y1, x2, y2, x3, y3;
    static SHORT angleMid;
    static WORD tempColor;
    LONG temp;

    static DRAW_SECTOR_STATES sectorState = SEC_DRAW_IDLE;

    switch(sectorState)
    {
        case SEC_DRAW_IDLE:

            // calculate points
            angleMid = (angleTo + angleFrom) >> 1;
            GetCirclePoint(outRadius, angleFrom, &x1, &y1);
            GetCirclePoint(outRadius, angleTo, &x2, &y2);
            x1 += cx;
            y1 += cy;
            x2 += cx;
            y2 += cy;

            // grab the current color value for later use
            tempColor = GetColor();

            // check if we need to draw the edges
            // special case for single data shown on pie chart
            // we remove the line drawn at angle 0
            if(!((angleFrom == 0) && (angleTo == 360)))
            {
                sectorState = SEC_DRAW_EDGE1;
```

```
                // special case for small angles
                GetCirclePoint(outRadius - 1, angleFrom + 1, &x3, &y3);
                x3 += cx;
                y3 += cy;
                goto sec_draw_edge1;
            }
            else
            {
                sectorState = SEC_DRAW_FILLINIT;
                goto sec_draw_fillinit;
            }

        case SEC_DRAW_EDGE1:
            sec_draw_edge1 : if(!Line(x3, y3, cx, cy)) return (0);
            GetCirclePoint(outRadius - 1, angleTo - 1, &x3, &y3);
            x3 += cx;
            y3 += cy;
            sectorState = SEC_DRAW_EDGE2;

        case SEC_DRAW_EDGE2:
            if(!Line(x3, y3, cx, cy))
                return (0);
            GetCirclePoint(outRadius - 1, angleMid, &x3, &y3);
            x3 += cx;
            y3 += cy;
            sectorState = SEC_DRAW_EDGE3;

        case SEC_DRAW_EDGE3:
            if(!Line(x3, y3, cx, cy))
                return (0);
            SetColor(outLineColor);
            sectorState = SEC_DRAW_EDGE4;

        case SEC_DRAW_EDGE4:
            if(!Line(x1, y1, cx, cy))
                return (0);
            sectorState = SEC_DRAW_EDGE5;

        case SEC_DRAW_EDGE5:
            if(!Line(x2, y2, cx, cy))
                return (0);
            sectorState = SEC_DRAW_FILLINIT;

        case SEC_DRAW_FILLINIT:
            sec_draw_fillinit : SetColor(tempColor);

            temp = ((x1 - x2) * (x1 - x2));
            temp += ((y1 - y2) * (y1 - y2));

            if(((DWORD) temp <= (DWORD) 16) && ((angleTo - angleFrom) < 90))
            {
                sectorState = SEC_DRAW_IDLE;
                return (1);
            }

            GetCirclePoint(outRadius - 2, angleMid, &x3, &y3);
            x3 += cx;
            y3 += cy;
            sectorState = SEC_DRAW_FILL;

        case SEC_DRAW_FILL:

            // FillSector() is a blocking call
            // making it non-blocking will further add delay to the rendering
            FillSector(x3, y3, outLineColor);
            sectorState = SEC_DRAW_IDLE;
            break;
    }   // end of switch()

    return (1);
}
```

```
#endif // USE_CHART
```

14.1.11 Chart.h

Functions

	Name	Description
	ChAddDataSeries (see page 88)	This function creates a DATASERIES (see page 110) object and populates the structure with the given parameters.
	ChCreate (see page 86)	This function creates a CHART (see page 110) 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 (see page 87)	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 (see page 102)() function. When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is... more (see page 87)
	ChFreeDataSeries (see page 108)	This function removes DATASERIES (see page 110) object from the list of DATASERIES (see page 110) objects and frees the memory used of that removed object.
	ChRemoveDataSeries (see page 89)	This function removes DATASERIES (see page 110) object from the list of DATASERIES (see page 110) 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.
	ChSetPercentRange (see page 98)	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 (see page 99)	This function sets the sample start and sample end when drawing the chart. Together with the data series' SHOW_DATA (see page 84) flags the different way of displaying the chart data is achieved.
	ChSetValueRange (see page 95)	This function sets the minimum and maximum range of values that the bar chart will show. The criteria is that min <= max.
	ChTranslateMsg (see page 109)	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
CH_3D_ENABLE (see page 81)	Bit to indicate that bar charts are to be drawn with 3-D effect
CH_BAR (see page 82)	Bit to indicate the chart is type bar. If both PIE and BAR types
CH_BAR_HOR (see page 82)	These bits (with CH_BAR (see page 82) bit set), sets the bar chart to
CH_CLR0 (see page 112)	Bright Blue
CH_CLR1 (see page 112)	Bright Red
CH_CLR10 (see page 114)	Light Orange
CH_CLR11 (see page 114)	Light Red
CH_CLR12 (see page 114)	Light Green
CH_CLR13 (see page 115)	Light Yellow
CH_CLR14 (see page 115)	Light Orange
CH_CLR15 (see page 115)	Gold
CH_CLR2 (see page 112)	Bright Green

CH_CLR3 (see page 113)	Bright Yellow
CH_CLR4 (see page 113)	Orange
CH_CLR5 (see page 113)	Blue
CH_CLR6 (see page 113)	Red
CH_CLR7 (see page 113)	Green
CH_CLR8 (see page 114)	Yellow
CH_CLR9 (see page 114)	Dark Orange
CH_DISABLED (see page 81)	Bit for disabled state.
CH_DONUT (see page 82)	These bits (with CH_PIE (see page 83) bit set), sets the pie chart to
CH_DRAW (see page 81)	Bit to indicate chart must be redrawn.
CH_DRAW_DATA (see page 81)	Bit to indicate data portion of the chart must be redrawn.
CH_HIDE (see page 83)	Bit to indicate chart must be removed from screen.
CH_LEGEND (see page 82)	Bit to indicate that legend is to be shown. Usable only when seriesCount > 1.
CH_NUMERIC (see page 82)	This bit is used only for bar charts. If this bit is set, it indicates that the
CH_PERCENT (see page 83)	Bit to indicate that the pie chart will be drawn with percentage
CH_PIE (see page 83)	Bit to indicate the chart is type pie. If both PIE and BAR types
CH_VALUE (see page 83)	Bit to indicate that the values of the bar chart data or pie chart
ChGetAxisLabelFont (see page 106)	This macro returns the location of the font used for the X and Y axis labels of the chart.
ChGetColorTable (see page 103)	This macro returns the current color table used for the pie and bar charts.
ChGetGridLabelFont (see page 107)	This macro returns the location of the font used for the X and Y axis grid labels of the chart.
ChGetPercentMax (see page 101)	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 (see page 102)	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.
ChGetPercentRange (see page 99)	This macro gets the percentage range for bar charts. The value returned is calculated from percentage max - min. To get the minimum use ChGetPercentMin (see page 102)() and to get the maximum use ChGetPercentMax (see page 101)().
ChGetSampleEnd (see page 98)	This macro returns the sampling end value.
ChGetSampleLabel (see page 97)	This macro returns the address of the current text string used for the sample axis label of the bar chart.
ChGetSampleRange (see page 100)	This macro gets the sample range for pie or bar charts. The value returned is calculated from smplEnd - smplStart.
ChGetSampleStart (see page 97)	This macro returns the sampling start value.
ChGetShowSeriesCount (see page 91)	This macro shows the number of data series that has its show flag set to SHOW_DATA (see page 84)
ChGetShowSeriesStatus (see page 92)	This macro returns the show ID status of the DATASERIES (see page 110).
ChGetTitle (see page 104)	This macro returns the address of the current text string used for the title of the chart.
ChGetTitleFont (see page 105)	This macro returns the location of the font used for the title of the chart.
ChGetValueLabel (see page 93)	This macro returns the address of the current text string used for the value axis label of the bar chart.
ChGetValueMax (see page 94)	This macro returns the current maximum value that will be drawn for bar charts.
ChGetValueMin (see page 94)	This macro returns the current minimum value that will be drawn for bar charts.
ChGetValueRange (see page 95)	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 (see page 94)() and to get the maximum use ChGetValueMax (see page 94)().
ChHideSeries (see page 91)	This macro sets the specified data series number show flag to be set to HIDE_DATA (see page 84).

ChSetAxisLabelFont (see page 106)	This macro sets the location of the font used for the X and Y axis labels of the chart.
ChSetColorTable (see page 102)	This macro sets the color table used to draw the data in pie and bar charts.
ChSetGridLabelFont (see page 107)	This macro sets the location of the font used for the X and Y axis grid labels of the chart.
ChSetSampleLabel (see page 96)	This macro sets the address of the current text string used for the sample axis label of the bar chart.
ChSetTitle (see page 103)	This macro sets the address of the current text string used for the title of the chart.
ChSetTitleFont (see page 104)	This macro sets the location of the font used for the title of the chart.
ChSetValueLabel (see page 92)	This macro sets the address of the current text string used for the value axis label of the bar chart.
ChShowSeries (see page 90)	This macro sets the specified data series number show flag to be set to SHOW_DATA (see page 84).
HIDE_DATA (see page 84)	Macro used to reset the data series show flag or indicate that the data series will be not be shown when the chart is drawn.
SHOW_DATA (see page 84)	Macro used to set the data series show flag or indicate that the data series will be shown when the chart is drawn.

Structures

Name	Description
CHART (see page 110)	Defines the parameters required for a chart Object.
CHARTPARAM (see page 111)	Defines the parameters for the CHART (see page 110) object.
DATASERIES (see page 110)	Defines a variable for the CHART (see page 110) 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.

Description

This is file Chart.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Chart
 *****************************************************************************
 * FileName:        Chart.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 4/8/08       ...
 ***************************************************************************/
#ifndef _CHART_H
    #define _CHART_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*
        Chart Terminologies

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. Definition for Value Axis and Sample Axis
             are flipped when chart is drawn horizontally.
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. Definition for Value Axis
              and Sample Axis are flipped when chart is drawn horizontally.
Title  - The text used to define the Title of the chart.
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.
Data Series - A complete series of data, distinguished by the same color and type of sample
point.
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 represent one sector or one one sample point.
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.
Value Range - The scale for the value axis.
              Example: range of numbers from the lowest to the highest to be charted.

*/

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define CH_DISABLED 0x0002          // Bit for disabled state.
    #define CH_LEGEND   0x0001          // Bit to indicate that legend is to be shown.
Usable only when seriesCount > 1.
    #define CH_VALUE    0x0004          // Bit to indicate that the values of the bar chart
data or pie chart

// data are to be shown
    #define CH_3D_ENABLE    0x0008      // Bit to indicate that bar charts are to be drawn
with 3-D effect
    #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.
    #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.
    #define CH_PERCENT  0x0010          // 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.
```

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

// be drawn horizontally.
    #define CH_DONUT    0x0140          // These bits (with CH_PIE bit set), sets the pie
chart to

// be drawn in a donut shape.
    #define CH_NUMERIC  0x0080          // 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.
    #define CH_DRAW_DATA    0x2000      // Bit to indicate data portion of the chart must
be redrawn.
    #define CH_DRAW         0x4000      // Bit to indicate chart must be redrawn.
    #define CH_HIDE         0x8000      // Bit to indicate chart must be removed from
screen.

/*****************************************************************************
* Data Series Show Flag Definitions:
*****************************************************************************/
    #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.
    #define HIDE_DATA   0               // Macro used to reset the data series show flag or
indicate that the data series will be not be shown when the chart is drawn.

/* Internal Definitions - used to modify some specific rendering characteristics */
    #define CH_MARGIN       2           // Margin from the edges.
    #define CH_YGRIDCOUNT   6           // defines the number of grids to be drawn on the
y-axis (includes the origin at y).
    #define USE_HORZ_ASCENDING_ORDER    // use ascending order when displaying variables on
horizontal mode (bar chart only)
    #define USE_PIE_ENABLE_LABEL        // use labels:  A:10%,30 where each sample is
labeled A-Z followed by a colon.

/*****************************************************************************
* 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.
*
*****************************************************************************/
typedef struct
{
    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.
} DATASERIES;

/*****************************************************************************
* Overview: Defines the parameters for the CHART object.
*
*****************************************************************************/
typedef struct
{
    XCHAR   *pTitle;            // Pointer to the Title of the chart.
    XCHAR   *pSmplLabel;        // Pointer to the bar chart sample axis label. Depending

    // if the bar chart is drawn vertically or horizontally, the
    // location of the sample will be in the x-axis or y-axis.
    XCHAR   *pValLabel;         // Pointer to the bar chart value axis label. Depending
```

```
    // if the bar chart is drawn vertically or horizontally, the
    // location of the sample will be in the x-axis or y-axis.
    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.
    WORD    *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.
    void    *pAxisLabelsFont;   // Pointer to the font used for X and Y axis labels.
    void    *pGridLabelsFont;   // Pointer to the font used for X and Y axis grid labels.
} CHARTPARAM;

/*********************************************************************
* Overview: Defines the parameters required for a chart Object.
*
*********************************************************************/
typedef struct
{
    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.
} CHART;

/*********************************************************************
*   Data Color Table
*   - The following macroses defines the default colors used when
*     rendering chart data.
*********************************************************************/
#ifdef USE_PALETTE

    #include "PaletteColorDefines.h"

#else

    #define CH_CLR0     BRIGHTBLUE      // Bright Blue
    #define CH_CLR1     BRIGHTRED       // Bright Red
    #define CH_CLR2     BRIGHTGREEN     // Bright Green
    #define CH_CLR3     BRIGHTYELLOW    // Bright Yellow
    #define CH_CLR4     RGBConvert(0xFF, 0xBB, 0x4C) // Orange
    #define CH_CLR5     BLUE                            // Blue
    #define CH_CLR6     RED                             // Red
    #define CH_CLR7     GREEN                           // Green
    #define CH_CLR8     YELLOW                          // Yellow
    #define CH_CLR9     RGBConvert(255, 140, 0)      // Dark Orange
    #define CH_CLR10    LIGHTBLUE                       // Light Orange
    #define CH_CLR11    LIGHTRED                        // Light Red
    #define CH_CLR12    LIGHTGREEN                      // Light Green
    #define CH_CLR13    RGBConvert(255, 255, 150)    // Light Yellow
    #define CH_CLR14    RGBConvert(255, 200, 0)      // Light Orange
    #define CH_CLR15    RGBConvert(255, 215, 0)      // Gold

#endif

SHORT       ChSetDataSeries(CHART *pCh, WORD seriesNum, BYTE status);

/*********************************************************************
* Function: CHART  *ChCreate( WORD ID, SHORT left, SHORT top,
*                             SHORT right, SHORT bottom,
*                             WORD state, DATASERIES *pData,
*                             CHARTPARAM *pParam, GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the object.
*        top - Top most position of the object.
*        right - Right most position of the object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        pData - Pointer to the data for the contents of the chart. NULL can
*                be assigned initially when creating the object.
*        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.
*        pScheme - Pointer to the style scheme used. When set to NULL,
*                  the default style scheme will be used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*
*   extern const FONT_FLASH GOLSmallFont;
*   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(),
*                         GetMaxY(),
*                         state,                        // state of the chart
*                         NULL,                         // data not initialized yet
*                         NULL,                         // no parameters yet
*                         pScheme);                     // style scheme used
*
*       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*)&GOLSmallFont);
*       ChSetAxisLabelFont(pChart, (void*)&GOLSmallFont);
*
*       // 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
*
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
CHART       *ChCreate
            (
                WORD        ID,
                SHORT       left,
                SHORT       top,
                SHORT       right,
                SHORT       bottom,
                WORD        state,
                DATASERIES  *pData,
                CHARTPARAM  *pParam,
                GOL_SCHEME  *pScheme
            );

/*********************************************************************
* Function: ChAddDataSeries(CHART *pCh, WORD nSamples, WORD *pData, XCHAR *pName)
*
* Overview: This function creates a DATASERIES object and
*           populates the structure with the given parameters.
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*        nSamples - The number of samples or data points.
*        pData - Pointer to the array of samples or data points.
*        pName - Pointer to the string used to label the data series.
*
* Output: 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.
*
* Example: See ChCreate() example.
*
* 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.
*
*********************************************************************/
DATASERIES  *ChAddDataSeries(CHART *pCh, WORD nSamples, WORD *pData, XCHAR *pName);

/*********************************************************************
* Function: void ChFreeDataSeries(void *pObj)
*
* Overview: This function removes DATASERIES object from the list of
*           DATASERIES objects and frees the memory used of that removed object.
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*
* Output: none.
*
*
* Example:
*   <CODE>
*
*   void ClearChartData(CHART *pCh) {
```

```
*       if(pCh->pChData != NULL)
*           // remove the all data series
*           ChFreeDataSeries(pCh;
*   }
*   </CODE>
*
* Side Effects: none.
*
*********************************************************************/
void ChFreeDataSeries(void *pObj);

/*********************************************************************
* Function: ChRemoveDataSeries(CHART *pCh, WORD number)
*
* 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.
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*        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.
*
* Output: none.
*
*
* Example:
*   <CODE>
*
*   void ClearChartData(CHART *pCh) {
*       if(pCh->pChData != NULL)
*           // remove the all data series
*           ChRemoveDataSeries(pCh, 0);
*   }
*   </CODE>
*
* Side Effects: none.
*
*********************************************************************/
void        ChRemoveDataSeries(CHART *pCh, WORD number);

/*********************************************************************
* Macros: SHORT ChShowSeries(CHART *pCh, WORD seriesNum)
*
* Overview: This macro sets the specified data series number
*           show flag to be set to SHOW_DATA.
*
* PreCondition: none
*
* Input: 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.
*
* Output: Returns the same passed number if successful otherwise
*         -1 if unsuccesful.
*
* Example:
*   <CODE>
*   // 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();
*   .....
*   </CODE> *
* Side Effects: none
*
*********************************************************************/
    #define ChShowSeries(pCh, seriesNum)    (ChSetDataSeries(pCh, seriesNum, SHOW_DATA))

/*********************************************************************
* Macros: SHORT ChHideSeries(CHART *pCh, WORD seriesNum)
*
* Overview: This macro sets the specified data series number
*           show flag to be set to HIDE_DATA.
*
* PreCondition: none
*
* Input: 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.
*
* Output: Returns the same passed number if successful otherwise
*         -1 if unsuccesful.
*
* Example: See ChShowSeries() example.
*
* Side Effects: none
*
*********************************************************************/
    #define ChHideSeries(pCh, seriesNum)    (ChSetDataSeries(pCh, seriesNum, HIDE_DATA))

/*********************************************************************
* Macros: ChGetShowSeriesCount(pCh)
*
* Overview: This macro shows the number of data series that has
*           its show flag set to SHOW_DATA
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the number of data series with its show flag set to
*         SHOW_DATA.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetShowSeriesCount(pCh)   (pCh->prm.seriesCount)

/*********************************************************************
* Macros: ChGetShowSeriesStatus(pDSeries)
*
* Overview: This macro returns the show ID status of the DATASERIES.
*
* PreCondition: none
*
* Input: pDSeries - Pointer to the data series(DATASERIES) that is
*                   being checked.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetShowSeriesStatus(pDSeries) (pDSeries->show)

/*********************************************************************
* Macros: ChGetValueMax(pCh)
*
* Overview: This macro returns the current maximum value that will
```

```
*           be drawn for bar charts.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the maximum value set when bar charts are drawn.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetValueMax(pCh)  (pCh->prm.valMax)

/*********************************************************************
* Macros: ChGetValueMin(pCh)
*
* Overview: This macro returns the current minimum value that will
*           be drawn for bar charts.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the minimum value set when bar charts are drawn.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetValueMin(pCh)  (pCh->prm.valMin)

/*********************************************************************
* Function: ChSetValueRange(CHART *pCh, WORD min, WORD max)
*
* Overview: This function sets the minimum and maximum range of values
*           that the bar chart will show.
*           The criteria is that min <= max.
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*        min - Minimum value that will be displayed in the bar chart.
*        max - Maximum value that will be displayed in the bar chart.
*
* Output: none.
*
* Side Effects: none.
*
*********************************************************************/
void    ChSetValueRange(CHART *pCh, WORD min, WORD max);

/*********************************************************************
* Function: ChGetValueRange(pCh)
*
* 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().
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*
* Output: Value range computed from valMax-valMin.
*
* Side Effects: none.
*
*********************************************************************/
    #define ChGetValueRange(pCh)    (pCh->prm.valMax - pCh->prm.valMin)

/*********************************************************************
* Macros: ChGetPercentMax(pCh)
*
```

```
* 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.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the maximum percentage value set when bar charts are drawn.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetPercentMax(pCh)    (pCh->prm.perMax)

/*********************************************************************
* Macros: ChGetPercentMin(pCh)
*
* 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.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the minimum percentage value when bar charts are drawn.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetPercentMin(pCh)    (pCh->prm.perMin)

/*********************************************************************
* Function: ChSetPercentRange(CHART *pCh, WORD min, WORD max)
*
* 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.
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*        min - Minimum percentage value that will be displayed in the bar chart.
*        max - Maximum percentage value that will be displayed in the bar chart.
*
* Output: none.
*
* Side Effects: none.
*
*********************************************************************/
void    ChSetPercentRange(CHART *pCh, WORD min, WORD max);

/*********************************************************************
* Function: ChGetPercentRange(pCh)
*
* 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().
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*
* Output: Percentage range computed from max-min.
*
* Side Effects: none.
*
*********************************************************************/
    #define ChGetPercentRange(pCh)  (pCh->prm.perMax - pCh->prm.perMin)
```

```
/*********************************************************************
* Function: ChSetSampleRange(CHART *pCh, WORD start, WORD end)
*
* 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.
*
*   <TABLE>
*       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 = don't care     > 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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*        start - Start point of the data samples to be displayed.
*        end - End point of the data samples to be displayed.
*
* Output: none.
*
* Example: See ChCreate() example.
*
* Side Effects: none.
*
*********************************************************************/
void    ChSetSampleRange(CHART *pCh, WORD start, WORD end);

/*********************************************************************
* Function: ChGetSampleRange(pCh)
*
* Overview: This macro gets the sample range for pie or bar charts.
*           The value returned is calculated from smplEnd - smplStart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the chart object.
*
* Output: Sample range computed from smplEnd - smplStart.
*
* Side Effects: none.
*
*********************************************************************/
    #define ChGetSampleRange(pCh)   (ChGetSampleEnd(pCh) - ChGetSampleStart(pCh))

/*********************************************************************
* Macros:  ChGetSampleStart(pCh)
*
* Overview: This macro returns the sampling start value.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the sample start point.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetSampleStart(pCh)   (((CHART *)pCh)->prm.smplStart)

/*********************************************************************
* Macros:  ChGetSampleEnd(pCh)
*
* Overview: This macro returns the sampling end value.
```

```
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *
 * Output: Returns the sample end point.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChGetSampleEnd(pCh) ((CHART *)pCh)->prm.smplEnd

/********************************************************************
 * Macros: ChSetTitle(pCh, pNewTitle)
 *
 * Overview: This macro sets the address of the current
 *           text string used for the title of the chart.
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *        pNewTitle - pointer to the string to be used as a title
 *                    of the chart.
 *
 * Output: none.
 *
 * Example: See ChCreate() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChSetTitle(pCh, pNewTitle)  (((CHART *)pCh)->prm.pTitle = pNewTitle)

/********************************************************************
 * Macros: ChGetTitle(pCh)
 *
 * Overview: This macro returns the address of the current
 *           text string used for the title of the chart.
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *
 * Output: Returns the pointer to the current title text used.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChGetTitle(pCh) (((CHART *)pCh)->prm.pTitle)

/********************************************************************
 * Macros: ChSetAxisLabelFont(pCh, pNewFont)
 *
 * Overview: This macro sets the location of the font used for
 *           the X and Y axis labels of the chart.
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *        pNewFont - Pointer to the font used.
 *
 * Output: none.
 *
 * Example: See ChCreate() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChSetAxisLabelFont(pCh, pNewFont)   (((CHART *)pCh)->prm.pAxisLabelsFont =
pNewFont)

/********************************************************************
```

```
* Macros: ChGetAxisLabelFont(pCh)
*
* Overview: This macro returns the location of the font used for
*           the X and Y axis labels of the chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the address of the current font used for the title text.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetAxisLabelFont(pCh) (((CHART *)pCh)->prm.pAxisLabelsFont)

/*********************************************************************
* Macros: ChSetGridLabelFont(pCh, pNewFont)
*
* Overview: This macro sets the location of the font used for
*           the X and Y axis grid labels of the chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*        pNewFont - Pointer to the font used.
*
* Output: none.
*
* Example: See ChCreate() example.
*
* Side Effects: none
*
*********************************************************************/
    #define ChSetGridLabelFont(pCh, pNewFont)   (((CHART *)pCh)->prm.pGridLabelsFont =
pNewFont)

/*********************************************************************
* Macros: ChGetGridLabelFont(pCh)
*
* Overview: This macro returns the location of the font used for
*           the X and Y axis grid labels of the chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the address of the current font used for the title text.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetGridLabelFont(pCh) (((CHART *)pCh)->prm.pGridLabelsFont)

/*********************************************************************
* Macros: ChSetTitleFont(pCh, pNewFont)
*
* Overview: This macro sets the location of the font used for
*           the title of the chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*        pNewFont - Pointer to the font used.
*
* Output: none.
*
* Example: See ChCreate() example.
*
* Side Effects: none
*
*********************************************************************/
```

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

/*********************************************************************
* Macros: ChGetTitleFont(pCh)
*
* Overview: This macro returns the location of the font used for
*           the title of the chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the address of the current font used for the title text.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetTitleFont(pCh) (((CHART *)pCh)->prm.pTitleFont)

/*********************************************************************
* Macros: ChSetSampleLabel(pCh, pNewXLabel)
*
* Overview: This macro sets the address of the current
*           text string used for the sample axis label of the bar chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*        pNewXLabel - pointer to the string to be used as an sample
*                     axis label of the bar chart.
*
* Output: none.
*
* Example: See ChCreate() example.
*
* Side Effects: none
*
*********************************************************************/
    #define ChSetSampleLabel(pCh, pNewXLabel)   (((CHART *)pCh)->prm.pSmplLabel =
pNewXLabel)

/*********************************************************************
* Macros: ChGetSampleLabel(pCh)
*
* Overview: This macro returns the address of the current
*           text string used for the sample axis label of the bar chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*
* Output: Returns the pointer to the current sample axis label text
*         of the bar chart.
*
* Side Effects: none
*
*********************************************************************/
    #define ChGetSampleLabel(pCh)   (((CHART *)pCh)->prm.pSmplLabel)

/*********************************************************************
* Macros: ChSetValueLabel(pCh, pNewYLabel)
*
* Overview: This macro sets the address of the current
*           text string used for the value axis label of the bar chart.
*
* PreCondition: none
*
* Input: pCh - Pointer to the object.
*        pNewYLabel - pointer to the string to be used as an value
*                    axis label of the bar chart.
*
* Output: none.
```

```
 *
 * Example: See ChCreate() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChSetValueLabel(pCh, pNewValueLabel)    (((CHART *)pCh)->prm.pValLabel =
pNewValueLabel)

/********************************************************************
 * Macros: ChGetValueLabel(pCh)
 *
 * Overview: This macro returns the address of the current
 *           text string used for the value axis label of the bar chart.
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *
 * Output: Returns the pointer to the current value axis label text
 *         of the bar chart.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChGetValueLabel(pCh)    (((CHART *)pCh)->prm.pValLabel)

/********************************************************************
 * Macros: ChSetColorTable(pCh, pNewTable)
 *
 * Overview: This macro sets the color table used to draw the data
 *           in pie and bar charts.
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *        pNewTable - Pointer to the color table that will be used.
 *
 * Output: none.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChSetColorTable(pCh, pNewTable) ((((CHART *)pCh)->prm.pColor) = pNewTable)

/********************************************************************
 * Macros: ChGetColorTable(pCh)
 *
 * Overview: This macro returns the current color table used for
 *           the pie and bar charts.
 *
 * PreCondition: none
 *
 * Input: pCh - Pointer to the object.
 *
 * Output: Returns the address of the color table used.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define ChGetColorTable(pCh)    (((CHART *)pCh)->prm.pColor)

/********************************************************************
 * Function: ChTranslateMsg(void *pObj, GOL_MSG *pMsg)
 *
 * 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.
 *
 *   <TABLE>
 *       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pCh    - The pointer to the object where the message will be
*                 evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                 the user interface.
*
* Output: Returns the translated message depending on the received GOL message:
*       - CH_MSG_SELECTED – Chart area is selected
*       - OBJ_MSG_INVALID – Chart is not affected
*
* Output: none.
*
* Side Effects: none
*
*********************************************************************/
WORD    ChTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD ChDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pCh - Pointer to the object to be rendered.
*
* Output: 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.
*
* Side Effects: none.
*
*********************************************************************/
WORD ChDraw(void *pObj);
#endif // _CHART_H
```

14.1.12 CheckBox.c

This is file CheckBox.c.

Body Source

```
/*****************************************************************************
```

```
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Check Box
 ****************************************************************************
 * FileName:        CheckBox.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, C32
 * Linker:          MPLAB LINK30, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 04/20/11     Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 ****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_CHECKBOX

/*********************************************************************
* Function: CHECKBOX  *CbCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                               SHORT bottom, WORD state, XCHAR *pText,
*                               GOL_SCHEME *pScheme)
*
* Overview: Creates the check box.
*
*********************************************************************/
CHECKBOX *CbCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    CHECKBOX    *pCb = NULL;

    pCb = (CHECKBOX *)GFX_malloc(sizeof(CHECKBOX));
    if(pCb == NULL)
        return (pCb);

    pCb->hdr.ID = ID;
    pCb->hdr.pNxtObj = NULL;
```

```
    pCb->hdr.type = OBJ_CHECKBOX;
    pCb->hdr.left = left;
    pCb->hdr.top = top;
    pCb->hdr.right = right;
    pCb->hdr.bottom = bottom;
    pCb->pText = pText;
    pCb->hdr.state = state;
    pCb->hdr.DrawObj = CbDraw;              // draw function
    pCb->hdr.MsgObj = CbTranslateMsg;       // message function
    pCb->hdr.MsgDefaultObj = CbMsgDefault;  // default message function
    pCb->hdr.FreeObj = NULL;                // free function

    // Set the style scheme
    if(pScheme == NULL)
        pCb->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pCb->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    // Set the text height
    pCb->textHeight = 0;
    if(pText != NULL)
    {
        pCb->textHeight = GetTextHeight(pCb->hdr.pGolScheme->pFont);
    }

    GOLAddObject((OBJ_HEADER *)pCb);

    return (pCb);
}

/*********************************************************************
* Function: CbSetText(CHECKBOX *pCb, char *pText)
*
* Overview: Sets the text.
*
*********************************************************************/
void CbSetText(CHECKBOX *pCb, XCHAR *pText)
{
    pCb->pText = pText;
    pCb->textHeight = GetTextHeight((BYTE *)pCb->hdr.pGolScheme->pFont);
}

/*********************************************************************
* Function: CbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: Changes the state of the check box by default.
*
*********************************************************************/
void CbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    CHECKBOX *pCb;

    pCb = (CHECKBOX *)pObj;

        #ifdef USE_FOCUS
            #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
        if(!GetState(pCb, CB_FOCUSED))
        {
            GOLSetFocus((OBJ_HEADER *)pCb);
        }
    }

            #endif
        #endif
    switch(translatedMsg)
    {
        case CB_MSG_CHECKED:
            SetState(pCb, CB_CHECKED | CB_DRAW_CHECK);  // Set checked and redraw
            break;
```

```
        case CB_MSG_UNCHECKED:
            ClrState(pCb, CB_CHECKED);                  // Reset check
            SetState(pCb, CB_DRAW_CHECK);               // Redraw
            break;
    }
}

/*****************************************************************
* Function: WORD CbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Overview: Checks if the check box will be affected by the message
*           and returns translated message.
*
*****************************************************************/
WORD CbTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    CHECKBOX *pCb;

    pCb = (CHECKBOX *)pObj;

    // Evaluate if the message is for the check box
    // Check if disabled first
    if(GetState(pCb, CB_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the check box borders
        if
        (
            (pCb->hdr.left < pMsg->param1) &&
            (pCb->hdr.right > pMsg->param1) &&
            (pCb->hdr.top < pMsg->param2) &&
            (pCb->hdr.bottom > pMsg->param2)
        )
        {
            if(pMsg->uiEvent == EVENT_PRESS)
            {
                if(GetState(pCb, CB_CHECKED))
                    return (CB_MSG_UNCHECKED);
                else
                    return (CB_MSG_CHECKED);
            }
        }

        return (OBJ_MSG_INVALID);
    }

        #endif
        #ifdef USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD)
    {
        if((WORD)pMsg->param1 == pCb->hdr.ID)
        {
            if(pMsg->uiEvent == EVENT_KEYSCAN)
            {
                if((pMsg->param2 == SCAN_SPACE_PRESSED) || (pMsg->param2 ==
SCAN_CR_PRESSED))
                {
                    if(GetState(pCb, CB_CHECKED))
                        return (CB_MSG_UNCHECKED);
                    else
                        return (CB_MSG_CHECKED);
                }
            }
        }

        return (OBJ_MSG_INVALID);
    }
```

```
        #endif
    return (OBJ_MSG_INVALID);
}

/*****************************************************************
* Function: WORD CbDraw(void *pObj)
*
* Output: returns the status of the drawing
*         0 - not complete
*         1 - done
*
* Overview: Draws check box.
*
*****************************************************************/
WORD CbDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        BOX_DRAW,
        RUN_DRAW,
        TEXT_DRAW,
        TEXT_DRAW_RUN,
        CHECK_DRAW,
        FOCUS_DRAW
    } CB_DRAW_STATES;

    static CB_DRAW_STATES state = REMOVE;

    SHORT checkIndent;
    CHECKBOX *pCb;

    pCb = (CHECKBOX *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(GetState(pCb, CB_HIDE | CB_DRAW))
            {
                SetColor(pCb->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pCb->hdr.left, pCb->hdr.top, pCb->hdr.right, pCb->hdr.bottom))
                {
                    return (0);
                }

                if(GetState(pCb, CB_HIDE))
                    return (1);
            }

            state = BOX_DRAW;

        case BOX_DRAW:
            if(GetState(pCb, CB_DRAW))
            {
                if(!GetState(pCb, CB_DISABLED))
                {
                    GOLPanelDraw
                    (
                        pCb->hdr.left + CB_INDENT,
                        pCb->hdr.top + CB_INDENT,
                        pCb->hdr.left + (pCb->hdr.bottom - pCb->hdr.top) - CB_INDENT,
                        pCb->hdr.bottom - CB_INDENT,
                        0,
                        pCb->hdr.pGolScheme->Color0,
                        pCb->hdr.pGolScheme->EmbossDkColor,
                        pCb->hdr.pGolScheme->EmbossLtColor,
                        NULL,
                        GOL_EMBOSS_SIZE
```

```
                );
            }
            else
            {
                GOLPanelDraw
                (
                    pCb->hdr.left + CB_INDENT,
                    pCb->hdr.top + CB_INDENT,
                    pCb->hdr.left + (pCb->hdr.bottom - pCb->hdr.top) - CB_INDENT,
                    pCb->hdr.bottom - CB_INDENT,
                    0,
                    pCb->hdr.pGolScheme->ColorDisabled,
                    pCb->hdr.pGolScheme->EmbossDkColor,
                    pCb->hdr.pGolScheme->EmbossLtColor,
                    NULL,
                    GOL_EMBOSS_SIZE
                );
            }

            state = RUN_DRAW;

        case RUN_DRAW:
            if(!GOLPanelDrawTsk())
                return (0);
            state = TEXT_DRAW;

        case TEXT_DRAW:
            if(pCb->pText != NULL)
            {
                SetFont(pCb->hdr.pGolScheme->pFont);

                if(!GetState(pCb, CB_DISABLED))
                {
                    SetColor(pCb->hdr.pGolScheme->TextColor0);
                }
                else
                {
                    SetColor(pCb->hdr.pGolScheme->TextColorDisabled);
                }

                MoveTo
                (
                    pCb->hdr.left + pCb->hdr.bottom - pCb->hdr.top + CB_INDENT,
                    (pCb->hdr.bottom + pCb->hdr.top - pCb->textHeight) >> 1
                );

                state = TEXT_DRAW_RUN;

            case TEXT_DRAW_RUN:
                if(!OutText(pCb->pText))
                    return (0);
            }
        }

        state = CHECK_DRAW;

    case CHECK_DRAW:
        if(GetState(pCb, CB_DRAW | CB_DRAW_CHECK))
        {
            if(!GetState(pCb, CB_DISABLED))
            {
                if(GetState(pCb, CB_CHECKED))
                {
                    SetColor(pCb->hdr.pGolScheme->TextColor0);
                }
                else
                {
                    SetColor(pCb->hdr.pGolScheme->Color0);
                }
            }
            else
            {
```

```
                    if(GetState(pCb, CB_CHECKED))
                    {
                        SetColor(pCb->hdr.pGolScheme->TextColorDisabled);
                    }
                    else
                    {
                        SetColor(pCb->hdr.pGolScheme->ColorDisabled);
                    }
                }

                checkIndent = (pCb->hdr.bottom - pCb->hdr.top) >> 2;

                if
                (
                    !Bar
                        (
                            pCb->hdr.left + checkIndent + GOL_EMBOSS_SIZE,
                            pCb->hdr.top + checkIndent + GOL_EMBOSS_SIZE,
                            pCb->hdr.left + (pCb->hdr.bottom - pCb->hdr.top) - checkIndent
- GOL_EMBOSS_SIZE,
                            pCb->hdr.bottom - checkIndent - GOL_EMBOSS_SIZE
                        )
                )
                {
                    return (0);
                }
            }

            state = FOCUS_DRAW;

        case FOCUS_DRAW:
            if(GetState(pCb, CB_DRAW | CB_DRAW_FOCUS))
            {
                if(GetState(pCb, CB_FOCUSED))
                {
                    SetColor(pCb->hdr.pGolScheme->TextColor0);
                }
                else
                {
                    SetColor(pCb->hdr.pGolScheme->CommonBkColor);
                }

                SetLineType(FOCUS_LINE);
                if(!Rectangle(pCb->hdr.left, pCb->hdr.top, pCb->hdr.right, pCb->hdr.bottom))
                {
                    return (0);
                }

                SetLineType(SOLID_LINE);
            }

            state = REMOVE;
            return (1);
    }

    return (1);
}

#endif // USE_CHECKBOX
```

14.1.13 CheckBox.h

Functions

	Name	Description
	CbCreate (see page 119)	This function creates a CHECKBOX (see page 124) 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 (see page 120)	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.
	CbMsgDefault (see page 122)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	CbSetText (see page 121)	This function sets the text that will be used.
	CbTranslateMsg (see page 123)	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
CB_CHECKED (see page 117)	Checked state
CB_DISABLED (see page 118)	Disabled state
CB_DRAW (see page 118)	Whole check box must be redrawn
CB_DRAW_CHECK (see page 118)	Check box mark should be redrawn
CB_DRAW_FOCUS (see page 118)	Focus must be redrawn
CB_FOCUSED (see page 118)	Focus state
CB_HIDE (see page 119)	Check box must be removed from screen
CbGetText (see page 121)	This macro returns the location of the text used for the check box.

Structures

Name	Description
CHECKBOX (see page 124)	The structure contains check box data

Description

This is file CheckBox.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Check box
 *****************************************************************************
 * FileName:        CheckBox.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 ****************************************************************************/
#ifndef _CHECKBOX_H
    #define _CHECKBOX_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"
// This is indent from outside borders
    #define CB_INDENT   2

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define CB_FOCUSED      0x0001  // Focus state
    #define CB_DISABLED     0x0002  // Disabled state
    #define CB_CHECKED      0x0004  // Checked state
    #define CB_HIDE         0x8000  // Check box must be removed from screen
    #define CB_DRAW_FOCUS   0x2000  // Focus must be redrawn
    #define CB_DRAW         0x4000  // Whole check box must be redrawn
    #define CB_DRAW_CHECK   0x1000  // Check box mark should be redrawn

/*********************************************************************
* Overview: The structure contains check box data
*********************************************************************/
typedef struct
{
    OBJ_HEADER  hdr;        // Generic header for all Objects (see OBJ_HEADER).
    SHORT       textHeight; // Pre-computed text height
    XCHAR       *pText;     // Pointer to text
} CHECKBOX;

/*********************************************************************
* Macros: CbGetText(pCb)
*
* Overview: This macro returns the location of the text
*           used for the check box.
*
* PreCondition: none
*
* Input: pCb - Pointer to the object
*
* Output: Returns the location of the text used.
*
* Side Effects: none
*
*********************************************************************/
    #define CbGetText(pCb)  pCb->pText

/*********************************************************************
* Function: CbSetText(CHECKBOX *pCb, XCHAR *pText)
```

```
*
* Overview: This function sets the text that will be used.
*
* PreCondition: none
*
* Input: pCb - The pointer to the check box whose text will be modified.
*        pText - The pointer to the text that will be used.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void        CbSetText(CHECKBOX *pCb, XCHAR *pText);

/*********************************************************************
* Function: CHECKBOX  *CbCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                               SHORT bottom, WORD state, XCHAR *pText,
*                               GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object
*        bottom - Bottom most position of the object
*        state - Sets the initial state of the object
*        pText - Pointer to the text of the check box.
*        pScheme - Pointer to the style scheme
*
* Output: Returns the pointer to the object created
*
* Example:
*   <CODE>
*   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,           // ID
*                      170,135,300,175,        // dimension
*                      CB_DRAW,                // Draw the object
*                      "Animate",              // text
*                      pScheme);               // use this scheme
*
*       while(!CbDraw(pCb[0]));                // draw the objects
*       while(!CbDraw(pCb[1]));
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
CHECKBOX    *CbCreate
            (
                WORD        ID,
                SHORT       left,
                SHORT       top,
                SHORT       right,
                SHORT       bottom,
                WORD        state,
                XCHAR       *pText,
```

```
                GOL_SCHEME  *pScheme
            );

/*****************************************************************
* Function: WORD CbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: 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
*
* Output: 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.
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*****************************************************************/
WORD        CbTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*****************************************************************
* Function: CbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       Translated Message   Input Source     Set/Clear State Bit   Description
*       ##################   ############     ######                ###########
*       CB_MSG_CHECKED       Touch Screen,   Set CB_CHECKED         Check Box will be
redrawn in checked state.
*                             Keyboard
*       CB_MSG_UNCHECKED     Touch Screen,   Clear CB_CHECKED       Check Box will be
redrawn in un-checked state.
*                             Keyboard
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message
*        pCb           - The pointer to the object whose state will be modified
*        pMsg          - The pointer to the GOL message
*
* Output: none
*
```

```
 * Side Effects: none
 *
 ***************************************************************************/
void        CbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/***************************************************************************
 * Function: WORD CbDraw(void *pObj)
 *
 * 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.
 *
 * PreCondition: Object must be created before this function is called.
 *
 * Input: pCb - Pointer to the object to be rendered.
 *
 * Output: 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.
 *
 * Example:
 *   See CbCreate() Example.
 *
 * Side Effects: none
 *
 ***************************************************************************/
WORD CbDraw(void *pObj);
#endif // _CHECKBOX_H
```

14.1.14 DigitalMeter.c

This is file DigitalMeter.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  DigitalMeter
 *****************************************************************************
 * FileName:        DigitalMeter.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Arpan kumar        06/11/09    Version 1.0 release
 * PAT                01/18/10    Added draw state to redraw only text.
 ***************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_DIGITALMETER

/*****************************************************************************
* Function: NumberToString(DWORD Value,XCHAR *pText, BYTE NoOfDigits, BYTE DotPos )
*
* Notes: convert the number to string
*
*****************************************************************************/
static void NumberToString(DWORD Value, XCHAR *pText, BYTE NoOfDigits, BYTE DotPos)
{
    BYTE    i;
    BYTE    bIndex;

    for(bIndex = 0; bIndex < NoOfDigits; bIndex++)
    {
        pText[NoOfDigits - bIndex - 1] = '0' + (Value % 10);
        Value /= 10;
    }

    if(DotPos != 0 && DotPos <= NoOfDigits)
    {
        for(i = 0; i < DotPos; i++)
        {
            pText[NoOfDigits - i] = pText[NoOfDigits - 1 - i];
        }

        pText[NoOfDigits - DotPos] = '.';
        pText[NoOfDigits + 1] = '\0';
    }

    //If dot position is 0 or greater than number of digits, then don't put dot in the
display
    else
    {
        pText[NoOfDigits] = '\0';
    }
}

/*****************************************************************************
**************
* Function: DIGITALMETER  *DmCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT
bottom, WORD state,
*                               DWORD Value, BYTE NoOfDigits, BYTE DotPos, GOL_SCHEME
*pScheme)
*
* Notes: Creates a DIGITALMETER object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*****************************************************************************
**************/
DIGITALMETER *DmCreate
(
```

```
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    DWORD       Value,
    BYTE        NoOfDigits,
    BYTE        DotPos,
    GOL_SCHEME  *pScheme
)
{
    DIGITALMETER    *pDm = NULL;
    pDm = GFX_malloc(sizeof(DIGITALMETER));
    if(pDm == NULL)
        return (pDm);

    pDm->hdr.ID = ID;                   // unique id assigned for referencing
    pDm->hdr.pNxtObj = NULL;            // initialize pointer to NULL
    pDm->hdr.type = OBJ_DIGITALMETER;   // set object type
    pDm->hdr.left = left;               // left,top corner
    pDm->hdr.top = top;
    pDm->hdr.right = right;             // right buttom corner
    pDm->hdr.bottom = bottom;
    pDm->Cvalue = Value;                // initial value to be displayed
    pDm->hdr.state = state;
    pDm->NoOfDigits = NoOfDigits;       // number of digits to be displayed
    pDm->DotPos = DotPos;               // position of decimal point
    pDm->hdr.DrawObj = DmDraw;          // draw function
    pDm->hdr.MsgObj = DmTranslateMsg;   // message function
    pDm->hdr.MsgDefaultObj = NULL;      // default message function
    pDm->hdr.FreeObj = NULL;            // free function

    // Set the color scheme to be used
    if(pScheme == NULL)
        pDm->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pDm->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pDm->textHeight = 0;
    if(pDm->Cvalue != 0)
    {

        // Set the text height
        pDm->textHeight = GetTextHeight(pDm->hdr.pGolScheme->pFont);
    }

    GOLAddObject((OBJ_HEADER *)pDm);

    return (pDm);
}

/*********************************************************************
* Function: DmSetValue(DIGITALMETER *pDm, DWORD Value)
*
* Notes: Sets the value to be displayed.
*
*********************************************************************/
void DmSetValue(DIGITALMETER *pDm, DWORD Value)
{

    // store the previous and current value to be displayed
    pDm->Pvalue = pDm->Cvalue;
    pDm->Cvalue = Value;

    pDm->textHeight = GetTextHeight(pDm->hdr.pGolScheme->pFont);
}

/*********************************************************************
* Function: WORD DmTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Evaluates the message if the object will be affected by the
```

```
*        message or not.
*
*********************************************************************/
WORD DmTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    DIGITALMETER *pDm;

    pDm = (DIGITALMETER *)pObj;

    // Evaluate if the message is for the static text
    // Check if disabled first
    if(GetState(pDm, DM_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in static text control borders
        if
        (
            (pDm->hdr.left < pMsg->param1) &&
            (pDm->hdr.right > pMsg->param1) &&
            (pDm->hdr.top < pMsg->param2) &&
            (pDm->hdr.bottom > pMsg->param2)
        )
        {
            return (DM_MSG_SELECTED);
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*********************************************************************
* Function: WORD DmDraw(void *pObj)
*
* Notes: This is the state machine to display the changing numbers.
*
*********************************************************************/
WORD DmDraw(void *pObj)
{
    typedef enum
    {
        DM_STATE_IDLE,
        DM_STATE_FRAME,
        DM_STATE_INIT,
        DM_STATE_SETALIGN,
        DM_STATE_DRAWTEXT
    } DM_DRAW_STATES;

    static DIGITALMETER *pDm = NULL;
    static DM_DRAW_STATES state = DM_STATE_IDLE;
    static SHORT charCtr = 0, lineCtr = 0;
    static XCHAR CurValue[DM_WIDTH], PreValue[DM_WIDTH];
    SHORT textWidth = 0;
    XCHAR ch = 0, pch = 0;

    pDm = (DIGITALMETER *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case DM_STATE_IDLE:
            SetClip(CLIP_DISABLE);

            if(GetState(pDm, DM_HIDE) || GetState(pDm, DM_DRAW))
            {
```

```
                SetColor(pDm->hdr.pGolScheme->CommonBkColor);
                if(Bar(pDm->hdr.left, pDm->hdr.top, pDm->hdr.right, pDm->hdr.bottom) == 0)
                    return (0);
            }
            // if the draw state was to hide then state is still IDLE STATE so no need to
change state
            if (GetState(pDm, DM_HIDE))
                return (1);
            state = DM_STATE_FRAME;

        case DM_STATE_FRAME:
            if(GetState(pDm, DM_DRAW | DM_FRAME) == (DM_DRAW | DM_FRAME))
            {
                // show frame if specified to be shown
                SetLineType(SOLID_LINE);
                SetLineThickness(NORMAL_LINE);
                if(!GetState(pDm, DM_DISABLED))
                {

                    // show enabled color
                    SetColor(pDm->hdr.pGolScheme->Color1);
                    if(Rectangle(pDm->hdr.left, pDm->hdr.top, pDm->hdr.right,
pDm->hdr.bottom) == 0)
                        return (0);
                }
                else
                {

                    // show disabled color
                    SetColor(pDm->hdr.pGolScheme->ColorDisabled);
                    if(Rectangle(pDm->hdr.left, pDm->hdr.top, pDm->hdr.right,
pDm->hdr.bottom) == 0)
                        return (0);
                }
            }

            // set clipping area, text will only appear inside the static text area.
            SetClip(CLIP_ENABLE);
            SetClipRgn(pDm->hdr.left + DM_INDENT, pDm->hdr.top, pDm->hdr.right - DM_INDENT,
pDm->hdr.bottom);
            state = DM_STATE_INIT;

        case DM_STATE_INIT:
            if(IsDeviceBusy())
                return (0);

            // set the text color
            if(!GetState(pDm, DM_DISABLED))
            {
                SetColor(pDm->hdr.pGolScheme->TextColor0);
            }
            else
            {
                SetColor(pDm->hdr.pGolScheme->TextColorDisabled);
            }

            // convert the values to be displayed in string format
            NumberToString(pDm->Pvalue, PreValue, pDm->NoOfDigits, pDm->DotPos);
            NumberToString(pDm->Cvalue, CurValue, pDm->NoOfDigits, pDm->DotPos);

            // use the font specified in the object
            SetFont(pDm->hdr.pGolScheme->pFont);

            state = DM_STATE_SETALIGN;  // go to drawing of text

        case DM_STATE_SETALIGN:
            if(!charCtr)
            {

                // set position of the next character (based on alignment and next
character)
                textWidth = GetTextWidth(CurValue, pDm->hdr.pGolScheme->pFont);
```

```
                // Display text with center alignment
                if(GetState(pDm, (DM_CENTER_ALIGN)))
                {
                    MoveTo((pDm->hdr.left + pDm->hdr.right - textWidth) >> 1, pDm->hdr.top
+ (lineCtr * pDm->textHeight));
                }

                // Display text with right alignment
                else if(GetState(pDm, (DM_RIGHT_ALIGN)))
                {
                    MoveTo((pDm->hdr.right - textWidth - DM_INDENT), pDm->hdr.top +
(lineCtr * pDm->textHeight));
                }

                // Display text with left alignment
                else
                {
                    MoveTo(pDm->hdr.left + DM_INDENT, pDm->hdr.top + (lineCtr *
pDm->textHeight));
                }
            }

            state = DM_STATE_DRAWTEXT;

        case DM_STATE_DRAWTEXT:
            pch = *(PreValue + charCtr);
            ch = *(CurValue + charCtr);

            // output one character at time until a newline character or a NULL character
is sampled
            while((0x0000 != ch))
            {
                if(IsDeviceBusy())
                {
                    return (0);         // device is busy return
                }

                if(GetState(pDm, DM_DRAW))
                {
                    SetColor(pDm->hdr.pGolScheme->CommonBkColor);
                    if(Bar(GetX(), pDm->hdr.top + 1, GetX() + textWidth, pDm->hdr.bottom -
1) == 0)
                        return (0);
                }
                else if(GetState(pDm, DM_UPDATE))
                {
                    if(pch != ch)
                    {
                        SetColor(pDm->hdr.pGolScheme->CommonBkColor);
                        if(Bar(GetX(), pDm->hdr.top + 1, GetX() + textWidth,
pDm->hdr.bottom - 1) == 0)
                            return (0);
                    }
                }

                SetColor(pDm->hdr.pGolScheme->TextColor0);

                // render the character
                while(!OutChar(ch));
                charCtr++;              // update to next character
                ch = *(CurValue + charCtr);
                pch = *(PreValue + charCtr);
            }

            // end of text string is reached no more lines to display
            lineCtr = 0;
            charCtr = 0;
            SetClip(CLIP_DISABLE);      // remove clipping
            state = DM_STATE_IDLE;      // go back to IDLE state
            return (1);
    }
```

```
    return (1);
}

#endif // USE_DIGITALMETER
```

14.1.15 DigitalMeter.h

Functions

	Name	Description
	DmCreate (see page 138)	This function creates a DIGITALMETER (see page 143) 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 (see page 139)	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 (see page 140)	This function sets the value that will be used for the object.
	DmTranslateMsg (see page 142)	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
DM_CENTER_ALIGN (see page 137)	Bit to indicate value is center aligned.
DM_DISABLED (see page 136)	Bit for disabled state.
DM_DRAW (see page 136)	Bit to indicate object must be redrawn.
DM_FRAME (see page 137)	Bit to indicate frame is displayed.
DM_HIDE (see page 137)	Bit to remove object from screen.
DM_RIGHT_ALIGN (see page 137)	Bit to indicate value is left aligned.
DM_UPDATE (see page 137)	Bit to indicate that only text must be redrawn.
DmDecVal (see page 141)	This macro is used to directly decrement the value.
DmGetValue (see page 140)	This macro returns the current value used for the object.
DmIncVal (see page 141)	This macro is used to directly increment the value.

Structures

Name	Description
DIGITALMETER (see page 143)	Defines the parameters required for a Digital Meter (see page 135) Object.

Description

This is file DigitalMeter.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Static text
 *****************************************************************************
 * FileName:        DigitalMeter.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 06/11/09    Version 1.0 release
 * 01/18/10    Added draw state to redraw only text.
 ****************************************************************************/
#ifndef _DIGITALMETER_H
    #define _DIGITALMETER_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define DM_DISABLED     0x0002  // Bit for disabled state.
    #define DM_RIGHT_ALIGN  0x0004  // Bit to indicate value is left aligned.
    #define DM_CENTER_ALIGN 0x0008  // Bit to indicate value is center aligned.
    #define DM_FRAME        0x0010  // Bit to indicate frame is displayed.
    #define DM_DRAW         0x4000  // Bit to indicate object must be redrawn.
    #define DM_UPDATE       0x2000  // Bit to indicate that only text must be redrawn.
    #define DM_HIDE         0x8000  // Bit to remove object from screen.

/* Indent constant for the text used in the frame. */
    #define DM_INDENT   0x02        // Text indent constant.

/* User should change this value depending on the number of digits he wants to display */
    #define DM_WIDTH    0x0A        // This value should be more than the no of digits
displayed

/*********************************************************************
* Structure:   DIGITALMETER
* Overview: Defines the parameters required for a <link Digital Meter> Object.
*********************************************************************/
typedef struct
{
    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
} DIGITALMETER;

/*********************************************************************
* Macros:  DmGetValue(pDm)
```

```
 *
 * Overview: This macro returns the current
 *           value used for the object.
 *
 * PreCondition: none
 *
 * Input: pDm - Pointer to the object.
 *
 * Output: Returns the value used.
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define DmGetValue(pDm) pDm->Cvalue

/********************************************************************
 * Function: DmSetValue(DIGITALMETER *pDm,  DWORD Value)
 *
 * Overview: This function sets the value that will be used for the object.
 *
 * PreCondition: none
 *
 * Input: pDm - The pointer to the object whose value will be modified.
 *        Value -  New value to be set for the <link Digital Meter>.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
void    DmSetValue(DIGITALMETER *pDm, DWORD Value);

/********************************************************************
 * Macros:  DmIncVal(pDm, deltaValue)
 *
 * Overview: This macro is used to directly increment the value.
 *
 * PreCondition: none
 *
 * Input: pDm - Pointer to the object.
 *        deltaValue - Number to be added to the current <link Digital Meter> value.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define DmIncVal(pDm, deltaValue)    DmSetValue(pDm, (pDm->Cvalue + deltaValue))

/********************************************************************
 * Macros:  DmDecVal(pDm, deltaValue)
 *
 * Overview: This macro is used to directly decrement the value.
 *
 * PreCondition: none
 *
 * Input: pDm - Pointer to the object.
 *        deltaValue - Number to be subtracted to the current <link Digital Meter> value.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
    #define DmDecVal(pDm, deltaValue)    DmSetValue(pDm, (pDm->Cvalue - deltaValue))

/********************************************************************
 * Function: DIGITALMETER  *DmCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT
bottom,
 *                                  WORD state , DWORD Value, BYTE NoOfDigits, BYTE DotPos,
GOL_SCHEME *pScheme)
 *
```

```
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the object.
*        top - Top most position of the object.
*        right - Right most position of the object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        Value - Sets the initial value to be displayed
*        NoOfDigits - Sets the number of digits to be displayed
*        DotPos - Sets the position of decimal point in the display
*        pScheme - Pointer to the style scheme. Set to NULL if
*                  default style scheme is used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <PRE>
*   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
*   </PRE>
*
* Side Effects: none
*
*********************************************************************/
DIGITALMETER    *DmCreate
                (
                    WORD        ID,
                    SHORT       left,
                    SHORT       top,
                    SHORT       right,
                    SHORT       bottom,
                    WORD        state,
                    DWORD       Value,
                    BYTE        NoOfDigits,
                    BYTE        DotPos,
                    GOL_SCHEME  *pScheme
                );

/*********************************************************************
* Function: WORD DmTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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 <link Digital Meter>.
*       OBJ_MSG_INVALID      Any           Any                           If the message did
not affect the object.
*   </TABLE>
*
* PreCondition: none
```

```
*
* Input: pDm   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: Returns the translated message depending on the received GOL message:
*         - DM_MSG_SELECTED – <link Digital Meter> is selected
*         - OBJ_MSG_INVALID – <link Digital Meter> is not affected
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD            DmTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD DmDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pDm - Pointer to the object to be rendered.
*
* Output: 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.
*
* Example:
*   See DmCreate() Example.
*
* Side Effects: none
*
*********************************************************************/
WORD DmDraw(void *pObj);
#endif // _DIGITALMETER_H
```

14.1.16 EditBox.c

This is file EditBox.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Edit Box
 *****************************************************************************
 * FileName:        EditBox.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, C32
 * Linker:          MPLAB LINK30, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 09/16/10     Fixed issue on focus. Modified EB_CARET behavior. Caret
 *              is now controlled by the EB_CARET state bit. If this bit
 *              is set, focus or manual user set of EB_DRAW_CARET will draw
 *              the caret. If this bit is not set, the caret will never
 *              be shown.
 * 02/24/11     Modified draw state machine to remove incorrect loops.
*****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_EDITBOX

/*********************************************************************
* Function: EDITBOX  *EbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom,
*                              WORD state , XCHAR *pText, WORD charMax, GOL_SCHEME *pScheme)
*
* Notes: Create the EDITBOX Object.
*
*********************************************************************/
EDITBOX  *EbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom,
                   WORD state , XCHAR *pText, WORD charMax, GOL_SCHEME *pScheme){

    EDITBOX *pEb = NULL;

    pEb = (EDITBOX*)GFX_malloc(sizeof(EDITBOX)+ (charMax + 1)*sizeof(XCHAR)); // ending
zero is not included into charMax
    if (pEb == NULL)
        return pEb;

    pEb->pBuffer  = (XCHAR*)((BYTE*)pEb+sizeof(EDITBOX));
    *pEb->pBuffer = 0;
    pEb->length   = 0;
    pEb->charMax  = charMax;

    if(pText != NULL)
        EbSetText(pEb, pText);

    pEb->hdr.ID             = ID;
    pEb->hdr.pNxtObj        = NULL;
    pEb->hdr.type           = OBJ_EDITBOX;
    pEb->hdr.left           = left;
    pEb->hdr.top            = top;
    pEb->hdr.right          = right;
    pEb->hdr.bottom         = bottom;
    pEb->hdr.state          = state;
```

```
    pEb->hdr.DrawObj        = EbDraw;        // draw function
    pEb->hdr.MsgObj         = EbTranslateMsg;// message function
    pEb->hdr.MsgDefaultObj  = EbMsgDefault;  // default message function
    pEb->hdr.FreeObj        = NULL;          // free function

    // Set the style scheme to be used
    if (pScheme == NULL)
        pEb->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pEb->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pEb->textHeight = GetTextHeight(pEb->hdr.pGolScheme->pFont);

    GOLAddObject((OBJ_HEADER*) pEb);
    return pEb;
}

/*********************************************************************
* Function: EbSetText(EDITBOX *pEb, XCHAR *pText)
*
* Notes: Sets a new text.
*
*********************************************************************/
void EbSetText(EDITBOX *pEb, XCHAR *pText){
WORD  ch;
WORD  length;
XCHAR* pointerFrom;
XCHAR* pointerTo;

    // Copy and count length
    pointerFrom = pText;
    pointerTo = pEb->pBuffer;
    length = 0;

    do{
        ch = *pointerFrom++;
        *pointerTo++ = ch;
        length++;
        if(length >= pEb->charMax){
            *pointerTo = 0;
            break;
        }
    }while(ch);


    pEb->length = length-1;
}

/*********************************************************************
* Function: void  EbAddChar(EDITBOX* pEb, XCHAR ch)
*
* Notes: Adds character at the end.
*
*********************************************************************/
void EbAddChar(EDITBOX* pEb, XCHAR ch){

    if(pEb->length >= pEb->charMax)
        return;

    // Add character
    pEb->pBuffer[pEb->length] = ch;
    pEb->length++;
    pEb->pBuffer[pEb->length] = 0;
}

/*********************************************************************
* Function: void  EbDeleteChar(EDITBOX* pEb)
*
* Notes: Deletes character at the end.
*
*********************************************************************/
void EbDeleteChar(EDITBOX* pEb){
```

```
    if(pEb->length == 0)
        return;

    // Delete charachter
    pEb->length--;
    pEb->pBuffer[pEb->length] = 0;

}

/*********************************************************************
* Function: WORD EbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Translates GOL message for the edit box
*
*********************************************************************/
WORD EbTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    EDITBOX *pEb;

    pEb = (EDITBOX *)pObj;

    // Evaluate if the message is for the edit box
    // Check if disabled first
    if (GetState(pEb, EB_DISABLED))
      return OBJ_MSG_INVALID;

#ifdef  USE_FOCUS
#ifdef  USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN) {
        // Check if it falls in edit box borders
        if( (pEb->hdr.left     < pMsg->param1) &&
            (pEb->hdr.right    > pMsg->param1) &&
            (pEb->hdr.top      < pMsg->param2) &&
            (pEb->hdr.bottom   > pMsg->param2) )
            return EB_MSG_TOUCHSCREEN;

        return OBJ_MSG_INVALID;
    }
#endif
#endif

#ifdef  USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD) {

        if(pMsg->uiEvent == EVENT_CHARCODE)
            return EB_MSG_CHAR;

        if(pMsg->uiEvent == EVENT_KEYSCAN)
            if(pMsg->param2 == SCAN_BS_PRESSED)
                return EB_MSG_DEL;

        return OBJ_MSG_INVALID;
    }
#endif


    return OBJ_MSG_INVALID;
}

/*********************************************************************
* Function: void EbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
*
* Notes: Changes the state of the edit box by default.
*
*********************************************************************/
void EbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg){

    EDITBOX *pEb;

    pEb = (EDITBOX *)pObj;
```

```
#ifdef  USE_FOCUS
#ifdef  USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN){
        if(!GetState(pEb,EB_FOCUSED)){
                GOLSetFocus((OBJ_HEADER*)pEb);
        }
    }
#endif
#endif

    switch(translatedMsg){

        case EB_MSG_CHAR:
            EbAddChar(pEb,(XCHAR)pMsg->param2);
            SetState(pEb, EB_DRAW);
            break;

        case EB_MSG_DEL:
            EbDeleteChar(pEb);
            SetState(pEb, EB_DRAW);
            break;

    }
}

/*********************************************************************
* Function: WORD EbDraw(void *pObj)
*
* Notes: This is the state machine to draw the button.
*
*********************************************************************/
WORD EbDraw(void *pObj)
{
typedef enum {
    EB_STATE_START,
    EB_STATE_DRAW_PANEL,
    EB_STATE_PREPARE_FOR_TEXT,
    EB_STATE_POSITION_TEXT,
    EB_STATE_TEXT,
    EB_STATE_CARET,
} EB_DRAW_STATES;

static EB_DRAW_STATES state = EB_STATE_START;
static XCHAR* pText;
SHORT temp;
SHORT width = 0;
EDITBOX *pEb;

    pEb = (EDITBOX *)pObj;

    while(!IsDeviceBusy())
    {
        switch(state){

            case EB_STATE_START:

                if(GetState(pEb, EB_HIDE)){
                    SetColor(pEb->hdr.pGolScheme->CommonBkColor);
                    if(!Bar(pEb->hdr.left,pEb->hdr.top,pEb->hdr.right,pEb->hdr.bottom))
return 0;
                    return 1;
                }

                if(GetState(pEb,EB_DISABLED)){
                    temp = pEb->hdr.pGolScheme->ColorDisabled;
                    ClrState(pEb,EB_CARET);
                    SetState(pEb,EB_DRAW_CARET);
                }else{
                    temp = pEb->hdr.pGolScheme->Color0;
                    if(GetState(pEb,EB_FOCUSED|EB_DRAW_CARET) ==
(EB_FOCUSED|EB_DRAW_CARET)){
                        SetState(pEb,EB_DRAW_CARET);
```

```
                    }
                }

                if(GetState(pEb,EB_DRAW))
                {

                    GOLPanelDraw(pEb->hdr.left,pEb->hdr.top,pEb->hdr.right,pEb->hdr.bottom,0
,
                        temp,
                        pEb->hdr.pGolScheme->EmbossDkColor,
                        pEb->hdr.pGolScheme->EmbossLtColor,
                        NULL,
                        GOL_EMBOSS_SIZE);

                    state = EB_STATE_DRAW_PANEL;
                    // no break here since it will always go to the EB_STATE_DRAW_PANEL
state
                }
                else
                {
                    state = EB_STATE_PREPARE_FOR_TEXT;
                    break;
                }

            case EB_STATE_DRAW_PANEL:
                if(!GOLPanelDrawTsk())
                    return 0;
                state = EB_STATE_PREPARE_FOR_TEXT;

                // no break here since it will always go to the EB_STATE_PREPARE_FOR_TEXT
state

            case EB_STATE_PREPARE_FOR_TEXT:

                SetClip(CLIP_ENABLE);

                SetClipRgn(pEb->hdr.left+GOL_EMBOSS_SIZE+EB_INDENT,
                           pEb->hdr.top+GOL_EMBOSS_SIZE+EB_INDENT,
                           pEb->hdr.right-GOL_EMBOSS_SIZE-EB_INDENT,
                           pEb->hdr.bottom-GOL_EMBOSS_SIZE-EB_INDENT);

                SetFont(pEb->hdr.pGolScheme->pFont);

                if(GetState(pEb,EB_DISABLED)){
                    SetColor(pEb->hdr.pGolScheme->TextColorDisabled);
                }else{
                    SetColor(pEb->hdr.pGolScheme->TextColor0);
                }

                // get the string buffer
                pText = pEb->pBuffer;
                temp = 1;

                // calculate how many lines are expected so we know where to
                // place the starting point
                while((XCHAR)*pText != (XCHAR)0){
                    if((XCHAR)*pText == (XCHAR)'\n')
                        temp++;
                    pText++;
                }

                // go back to the start of the buffer
                pText = pEb->pBuffer;
                // position the cursor
                MoveTo(GetX(),(pEb->hdr.top+pEb->hdr.bottom-temp*pEb->textHeight)>>1);

                // set state to actual string rendering loop
                state = EB_STATE_POSITION_TEXT;

                // no break here since it will always go to the EB_STATE_POSITION_TEXT state

            case EB_STATE_POSITION_TEXT:
```

```
            // get width of the string to render
            width = GetTextWidth(pText,pEb->hdr.pGolScheme->pFont);

            // place the starting point based on text alignment
            if (!GetState(pEb, EB_CENTER_ALIGN|EB_RIGHT_ALIGN)) {
                MoveTo(pEb->hdr.left+GOL_EMBOSS_SIZE+EB_INDENT, GetY());
            }else{
                if (GetState(pEb, EB_RIGHT_ALIGN)) {
                    MoveTo(pEb->hdr.right-width-EB_INDENT-GOL_EMBOSS_SIZE, GetY());
                }else{
                    MoveTo((pEb->hdr.left+pEb->hdr.right-width)>>1,GetY());
                }
            }

            state = EB_STATE_TEXT;

            // no break here since it will always go to the EB_STATE_TEXT state

        case EB_STATE_TEXT:

            // this is the actual string rendering

            if(GetState(pEb,EB_DRAW)){
                if(!OutText(pText))
                    return 0;
            }else{
                MoveRel(width, 0);
            }
            while((XCHAR)*pText>(XCHAR)15)
                pText++;

            if((XCHAR)*pText == (XCHAR)'\n')
            {
                MoveRel(0, pEb->textHeight);
            }

            // check if end of string
            if (*pText != 0)
            {
                state = EB_STATE_POSITION_TEXT;
                pText++;
                break;
            }

            state = EB_STATE_CARET;

            // no break here since it will always go to the EB_STATE_CARET state

        case EB_STATE_CARET:

            // draw the caret if required

            if(!GetState(pEb,EB_DISABLED)){

                if(GetState(pEb,EB_DRAW_CARET) && GetState(pEb,EB_CARET)){
                    SetColor(pEb->hdr.pGolScheme->TextColor0);
                }else{
                    SetColor(pEb->hdr.pGolScheme->Color0);
                }

                if(!Bar(GetX(),GetY(),GetX()+EB_CARET_WIDTH,GetY()+pEb->textHeight))
                    return 0;
            }

            SetClip(CLIP_DISABLE);

            state = EB_STATE_START;

            return 1;

    } // switch()
```

```
    }//  while(!IsDeviceBusy())

    return 0;
}

#endif // USE_EDITBOX
```

14.1.17 EditBox.h

Functions

	Name	Description
	EbAddChar (see page 150)	This function inserts a character at the end of the text used by the object.
	EbCreate (see page 147)	This function creates a EDITBOX (see page 153) 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.
	EbDeleteChar (see page 150)	This function removes a character at the end of the text used by the object.
	EbDraw (see page 148)	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.
	EbMsgDefault (see page 151)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	EbSetText (see page 149)	This function sets the text to be used for the object.
	EbTranslateMsg (see page 152)	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
EB_CARET (see page 146)	Bit to indicate the cursor will always be shown. This bit can be set when focused and reset when object is disabled. Application can also set and clear the bit through SetState (see page 289)() or ClrState (see page 289)().
EB_CENTER_ALIGN (see page 145)	Bit to indicate text is center aligned.
EB_DISABLED (see page 145)	Bit for disabled state.
EB_DRAW (see page 145)	Bit to indicate whole edit box must be redrawn.
EB_DRAW_CARET (see page 146)	Bit to indicate the cursor caret will be redrawn if the EB_CARET (see page 146) state bit is set.
EB_FOCUSED (see page 146)	Bit for focused state.
EB_HIDE (see page 146)	Bit to remove object from screen.
EB_RIGHT_ALIGN (see page 146)	Bit to indicate text is left aligned.
EbGetText (see page 149)	This macro returns the address of the current text string used for the object.

Structures

Name	Description
EDITBOX (see page 153)	Defines the parameters required for a Edit Box Object.

Description

This is file EditBox.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Edit box
 *****************************************************************************
 * FileName:        EditBox.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 09/16/10     Modified use of EB_DRAW_CARET, & EB_CARET.
 *              EB_DRAW_CARET when set will draw the caret. EB_DRAW_CARET can
 *              also be enabled by the EB_FOCUSED bit.
 *              EB_CARET when set will always draw the caret.
 *****************************************************************************/
#ifndef _EDITBOX_H
    #define _EDITBOX_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define EB_FOCUSED      0x0001  // Bit for focused state.
    #define EB_DISABLED     0x0002  // Bit for disabled state.
    #define EB_RIGHT_ALIGN  0x0004  // Bit to indicate text is left aligned.
    #define EB_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned.
    #define EB_CARET        0x0010  // Bit to indicate the cursor will always be shown.
                                    // This bit can be set when focused and reset when
object is disabled.
                                    // Application can also set and clear the bit through
SetState() or ClrState().
    #define EB_DRAW_CARET   0x2000  // Bit to indicate the cursor caret will be redrawn if
the EB_CARET state bit is set.
    #define EB_DRAW         0x4000  // Bit to indicate whole edit box must be redrawn.
    #define EB_HIDE         0x8000  // Bit to remove object from screen.
    #define EB_INDENT       0x02    // Indent for the text from the frame.
    #define EB_CARET_WIDTH  0x02    // Caret line width.
```

```
/*********************************************************************
* Overview: Defines the parameters required for a Edit Box Object.
*********************************************************************/
typedef struct
{
    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.
} EDITBOX;

/*********************************************************************
* Function: EDITBOX  *EbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom,
*                              WORD state , XCHAR *pText, WORD charMax, GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        pText - Pointer to the text to be used.
*        charMax - Defines the maximum number of characters in the edit box.
*        pScheme - Pointer to the style scheme.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   #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.
*   }
*
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
EDITBOX *EbCreate
        (
            WORD        ID,
            SHORT       left,
            SHORT       top,
            SHORT       right,
            SHORT       bottom,
            WORD        state,
            XCHAR       *pText,
            WORD        charMax,
            GOL_SCHEME  *pScheme
        );
```

```
/*********************************************************************
* Function: EbSetText(EDITBOX *pEb, XCHAR *pText)
*
* Overview: This function sets the text to be used for the object.
*
* PreCondition: none
*
* Input: pEb - The pointer to the object whose text will be modified.
*        pText - Pointer to the text that will be used.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void    EbSetText(EDITBOX *pEb, XCHAR *pText);

/*********************************************************************
* Macros:  EbGetText(pEb)
*
* Overview: This macro returns the address of the current
*           text string used for the object.
*
* PreCondition: none
*
* Input: pEb - Pointer to the object
*
* Output: Returns pointer to the text string being used.
*
* Side Effects: none
*
*********************************************************************/
    #define EbGetText(pEb)  (pEb->pBuffer)

/*********************************************************************
* Function: void  EbAddChar(EDITBOX* pEb, XCHAR ch)
*
* Overview: This function inserts a character at the end of the
*           text used by the object.
*
* PreCondition: none
*
* Input: pEb - The pointer to the object whose text will be modified.
*        ch - Character to be inserted.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void    EbAddChar(EDITBOX *pEb, XCHAR ch);

/*********************************************************************
* Function: void  EbDeleteChar(EDITBOX* pEb)
*
* Overview: This function removes a character at the end of the
*           text used by the object.
*
* PreCondition: none
*
* Input: pEb - The pointer to the object to be modified.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void    EbDeleteChar(EDITBOX *pEb);

/*********************************************************************
* Function: WORD EbTranslateMsg(void *pObj, GOL_MSG *pMsg)
```

```
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pEb   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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.
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD    EbTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: void EbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message.
*        pEb           - The pointer to the object whose state will be modified.
*        pMsg          - The pointer to the GOL message.
*
* Output: none
*
* Example:
*   Usage is similar to BtnMsgDefault() example.
*
* Side Effects: none
*
*********************************************************************/
void    EbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD EbDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pEb - Pointer to the object to be rendered.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
WORD EbDraw(void *pObj);
#endif // _EDITBOX_H
```

14.1.18 Grid.c

This is file Grid.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Grid control
 *****************************************************************************
 * FileName:        Grid.c
 * Dependencies:    string.h, Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 02/29/08    ...
 * 04/16/10    Added Grid Item as text.
 * 04/20/11    Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 ***************************************************************************/

/*
    Grid Object

TODO: Currently the grid does not support scroll bars. It must fit on the screen.

*/
#include <string.h>
#include "Graphics/Graphics.h"

#ifdef USE_GRID
    #define CELL_AT(c, r)   ((c * pGrid->numRows) + r)

/* */

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
)
{
    GRID    *pGrid = NULL;

    if((pGrid = (GRID *)GFX_malloc(sizeof(GRID))) == NULL)
    {
        return (NULL);
    }

    if((pGrid->gridObjects = GFX_malloc(sizeof(GRIDITEM) * numColumns * numRows)) == NULL)
    {
        GFX_free(pGrid);
        return (NULL);
    }

    // Initialize grid items to default.
    memset(pGrid->gridObjects, 0, sizeof(GRIDITEM) * numColumns * numRows);

    // Initialize grid
    pGrid->hdr.ID = ID;
    pGrid->hdr.pNxtObj = NULL;
    pGrid->hdr.type = OBJ_GRID;
    pGrid->hdr.state = state;
    pGrid->hdr.left = left;
    pGrid->hdr.top = top;
    pGrid->hdr.right = right;
    pGrid->hdr.bottom = bottom;
    pGrid->numColumns = numColumns;
    pGrid->numRows = numRows;
    pGrid->cellWidth = cellWidth;
    pGrid->cellHeight = cellHeight;
    pGrid->focusX = 0;
    pGrid->focusY = 0;
    pGrid->hdr.DrawObj = GridDraw;              // draw function
    pGrid->hdr.MsgObj = GridTranslateMsg;       // message function
    pGrid->hdr.MsgDefaultObj = GridMsgDefault;  // default message function
    pGrid->hdr.FreeObj = GridFreeItems;         // free function
```

```
    // Set the color scheme to be used
    if(pScheme == NULL)
    {
        pGrid->hdr.pGolScheme = _pDefaultGolScheme;
    }
    else
    {
        pGrid->hdr.pGolScheme = (GOL_SCHEME *)pScheme;
    }

    GOLAddObject((OBJ_HEADER *)pGrid);

    return (pGrid);
}

    #define CELL_LEFT       (1 + pGrid->hdr.left + (i * (pGrid->cellWidth + 1)))
    #define CELL_TOP        (1 + pGrid->hdr.top + (j * (pGrid->cellHeight + 1)))
    #define CELL_RIGHT      (CELL_LEFT + pGrid->cellWidth - 1)
    #define CELL_BOTTOM     (CELL_TOP + pGrid->cellHeight - 1)
    #define BITMAP_SCALE    1

/* */

WORD GridDraw(void *pObj)
{
    SHORT   i;
    SHORT   j;
    SHORT   xText, yText;
    GRID *pGrid;

    pGrid = (GRID *)pObj;

    if
    (
        (pGrid->hdr.state & GRID_DRAW_ITEMS) ||
        (pGrid->hdr.state & GRID_DRAW_ALL) ||
        (pGrid->hdr.state & GRID_SHOW_FOCUS)
    )
    {
        if(pGrid->hdr.state & GRID_DRAW_ALL)
        {

            // Clear the entire region.
            SetColor(pGrid->hdr.pGolScheme->CommonBkColor);
            while(!Bar(pGrid->hdr.left, pGrid->hdr.top, pGrid->hdr.right,
pGrid->hdr.bottom));

            // initialize the global cursor positions to the Grid left top position.
            _cursorX = pGrid->hdr.left;
            _cursorY = pGrid->hdr.top;

            // Draw the grid lines
            if(pGrid->hdr.state & (GRID_SHOW_LINES | GRID_SHOW_BORDER_ONLY |
GRID_SHOW_SEPARATORS_ONLY))
            {
                SetLineType(SOLID_LINE);
                SetColor(pGrid->hdr.pGolScheme->EmbossLtColor);

                // Draw the outside of the box
                // TODO This should have some 3D effects added with GOL_EMBOSS_SIZE
                if(pGrid->hdr.state & (GRID_SHOW_LINES | GRID_SHOW_BORDER_ONLY))
                {
                    while(!Line(pGrid->hdr.left, pGrid->hdr.top, pGrid->hdr.right,
pGrid->hdr.top));
                    LineTo(_cursorX, pGrid->hdr.bottom);
                    LineTo(pGrid->hdr.left, _cursorY);
                    LineTo(pGrid->hdr.left, pGrid->hdr.top);
                }

                // Draw the lines between each cell
                if(pGrid->hdr.state & (GRID_SHOW_LINES | GRID_SHOW_SEPARATORS_ONLY))
                {
```

```
                    for(i = 1; i < pGrid->numColumns; i++)
                    {
                        while
                        (
                            !Line
                                (
                                    pGrid->hdr.left + i * (pGrid->cellWidth + 1),
                                    pGrid->hdr.top,
                                    pGrid->hdr.left + i * (pGrid->cellWidth + 1),
                                    pGrid->hdr.top + pGrid->numRows * (pGrid->cellHeight +
1)
                                )
                        );
                    }

                    for(i = 1; i < pGrid->numRows; i++)
                    {
                        while
                        (
                            !Line
                                (
                                    pGrid->hdr.left,
                                    pGrid->hdr.top + i * (pGrid->cellHeight + 1),
                                    pGrid->hdr.right,
                                    pGrid->hdr.top + i * (pGrid->cellHeight + 1)
                                )
                        );
                    }
                }
            }
        }

        for(i = 0; i < pGrid->numColumns; i++)
        {
            for(j = 0; j < pGrid->numRows; j++)
            {
                if
                (
                    (pGrid->hdr.state & GRID_DRAW_ALL) ||
                    (
                        (pGrid->hdr.state & GRID_DRAW_ITEMS) &&
                        (pGrid->gridObjects[CELL_AT(i, j)].status & GRIDITEM_DRAW)
                    ) ||
                    ((pGrid->hdr.state & GRID_SHOW_FOCUS) && (i == pGrid->focusX) && (j ==
pGrid->focusY))
                )
                {

                    // Clear the cell
                    SetColor(pGrid->hdr.pGolScheme->CommonBkColor);
                    while(!Bar(CELL_LEFT, CELL_TOP, CELL_RIGHT, CELL_BOTTOM));

                    // Draw the cell
                    if((pGrid->gridObjects[CELL_AT(i, j)].status & GRID_TYPE_MASK) ==
GRIDITEM_IS_BITMAP)
                    {

                        // Draw the bitmap
                        if(pGrid->gridObjects[CELL_AT(i, j)].data)
                        {
                            while(!PutImage(CELL_LEFT, CELL_TOP,
pGrid->gridObjects[CELL_AT(i, j)].data, BITMAP_SCALE));
                        }
                    }
                    else
                    {

                        // Draw the text
                        if(pGrid->gridObjects[CELL_AT(i, j)].data)
                        {
                            SetFont(pGrid->hdr.pGolScheme->pFont);
                            SetColor(pGrid->hdr.pGolScheme->TextColor0);
```

```
                            if(GetState(pGrid, GRIDITEM_TEXTRIGHT))
                            {
                                xText =
CELL_LEFT+pGrid->cellWidth-GetTextWidth(pGrid->gridObjects[CELL_AT(i,
j)].data,pGrid->hdr.pGolScheme->pFont);
                            }
                            else if(GetState(pGrid, GRIDITEM_TEXTLEFT))
                            {
                                xText = CELL_LEFT;
                            }
                            else
                            {
                                xText =
CELL_LEFT+(pGrid->cellWidth>>1)-(GetTextWidth(pGrid->gridObjects[CELL_AT(i,
j)].data,pGrid->hdr.pGolScheme->pFont)>>1);
                            }
                            if(GetState(pGrid, GRIDITEM_TEXTBOTTOM))
                            {
                                yText =
CELL_TOP+pGrid->cellHeight-GetTextHeight(pGrid->hdr.pGolScheme->pFont);
                            }
                            else if(GetState(pGrid, GRIDITEM_TEXTTOP))
                            {
                                yText = CELL_TOP;
                            }
                            else
                            {
                                yText =
CELL_TOP+(pGrid->cellHeight>>1)-(GetTextHeight(pGrid->hdr.pGolScheme->pFont)>>1);
                            }

                            while(!OutTextXY( xText, yText, pGrid->gridObjects[CELL_AT(i,
j)].data));
                        }
                    }

                    // Draw the focus if applicable.
                    if((pGrid->hdr.state & GRID_SHOW_FOCUS) && (i == pGrid->focusX) && (j
== pGrid->focusY))
                    {
                        SetColor(pGrid->hdr.pGolScheme->EmbossLtColor);
                        SetLineType(DOTTED_LINE);
                        SetLineThickness(NORMAL_LINE);
                        while(!Rectangle(CELL_LEFT, CELL_TOP, CELL_RIGHT, CELL_BOTTOM));
                    }

                    // If the cell is selected, indicate it.
                    if(pGrid->gridObjects[CELL_AT(pGrid->focusX, pGrid->focusY)].status &
GRIDITEM_SELECTED)
                    {
                        SetColor(pGrid->hdr.pGolScheme->EmbossLtColor);
                        SetLineType(SOLID_LINE);
                        if(pGrid->hdr.state & GRID_SHOW_LINES)
                        {
                            SetLineThickness(THICK_LINE);
                        }
                        else
                        {
                            SetLineThickness(NORMAL_LINE);
                        }

                        while(!Rectangle(CELL_LEFT - 1, CELL_TOP - 1, CELL_RIGHT + 1,
CELL_BOTTOM + 1));
                    }

                    GridClearCellState(pGrid, i, j, GRIDITEM_DRAW);
                }
            }
        }

        //pGrid->state &= ~(GRID_DRAW_ITEMS || GRID_DRAW_ALL || GRID_SHOW_FOCUS);
```

```
        pGrid->hdr.state &= ~(GRID_DRAW_ITEMS || GRID_DRAW_ALL);

        // Set line state
        SetLineType(SOLID_LINE);
    }

    return (1);
}

/* */
void GridFreeItems(void *pObj)
{
    GRID *pGrid;

    pGrid = (GRID *)pObj;

    if(pGrid && pGrid->gridObjects)
    {
        GFX_free(pGrid->gridObjects);
        pGrid->gridObjects = NULL;  // Just in case...
    }
}

/* */
WORD GridSetCell(GRID *pGrid, SHORT column, SHORT row, WORD state, void *data)
{
    if((column >= pGrid->numColumns) || (row >= pGrid->numRows))
    {
        return (GRID_OUT_OF_BOUNDS);
    }

    pGrid->gridObjects[CELL_AT(column, row)].data = data;
    pGrid->gridObjects[CELL_AT(column, row)].status = state;    // This overwrites
GRIDITEM_SELECTED
    return (GRID_SUCCESS);
}

/* */
void GridClearCellState(GRID *pGrid, SHORT column, SHORT row, WORD state)
{
    if((column >= pGrid->numColumns) || (row >= pGrid->numRows))
    {
        return;
    }

    pGrid->gridObjects[CELL_AT(column, row)].status &= ~state;

    return;
}

/* */
void GridSetFocus(GRID *pGrid, SHORT column, SHORT row)
{
    if((column >= pGrid->numColumns) || (row >= pGrid->numRows))
    {
        return;
    }

    pGrid->focusX = column;
    pGrid->focusY = row;
}

/* */
void GridSetCellState(GRID *pGrid, SHORT column, SHORT row, WORD state)
{
    if((column >= pGrid->numColumns) || (row >= pGrid->numRows))
    {
        return;
    }

    pGrid->gridObjects[CELL_AT(column, row)].status |= state;
```

```
    return;
}

/* */
void *GridGetCell(GRID *pGrid, SHORT column, SHORT row, WORD *cellType)
{
    if((column >= pGrid->numColumns) || (row >= pGrid->numRows))
    {
        return (NULL);
    }

    *cellType = pGrid->gridObjects[CELL_AT(column, row)].status & GRID_TYPE_MASK;

    return (pGrid->gridObjects[CELL_AT(column, row)].data);
}

/* */
void GridMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    GRID *pGrid;

    pGrid = (GRID *)pObj;

    switch(translatedMsg)
    {
        case GRID_MSG_ITEM_SELECTED:

            // Currently, only a single item can be selected.  This can be expanded later,
            // when touchscreen support is enhanced.
            pGrid->gridObjects[CELL_AT(pGrid->focusX, pGrid->focusY)].status ^=
GRIDITEM_SELECTED;

            //            SetState( pGrid, GRID_SHOW_FOCUS | GRID_DRAW_ITEMS );
            SetState(pGrid, GRID_DRAW_ITEMS);
            break;

        case GRID_MSG_UP:
            if(pGrid->focusY > 0)
            {
                pGrid->gridObjects[CELL_AT(pGrid->focusX, pGrid->focusY)].status |=
GRIDITEM_DRAW;
                pGrid->focusY--;

                //            SetState( pGrid, GRID_SHOW_FOCUS | GRID_DRAW_ITEMS );
                SetState(pGrid, GRID_DRAW_ITEMS);
            }

            break;

        case GRID_MSG_DOWN:
            if(pGrid->focusY < (pGrid->numRows - 1))
            {
                pGrid->gridObjects[CELL_AT(pGrid->focusX, pGrid->focusY)].status |=
GRIDITEM_DRAW;
                pGrid->focusY++;

                //            SetState( pGrid, GRID_SHOW_FOCUS | GRID_DRAW_ITEMS );
                SetState(pGrid, GRID_DRAW_ITEMS);
            }

            break;

        case GRID_MSG_LEFT:
            if(pGrid->focusX > 0)
            {
                pGrid->gridObjects[CELL_AT(pGrid->focusX, pGrid->focusY)].status |=
GRIDITEM_DRAW;
                pGrid->focusX--;

                //            SetState( pGrid, GRID_SHOW_FOCUS | GRID_DRAW_ITEMS );
                SetState(pGrid, GRID_DRAW_ITEMS);
            }
```

```
            break;

        case GRID_MSG_RIGHT:
            if(pGrid->focusX < (pGrid->numColumns - 1))
            {
                pGrid->gridObjects[CELL_AT(pGrid->focusX, pGrid->focusY)].status |=
GRIDITEM_DRAW;
                pGrid->focusX++;

                //            SetState( pGrid, GRID_SHOW_FOCUS | GRID_DRAW_ITEMS );
                SetState(pGrid, GRID_DRAW_ITEMS);
            }

            break;
    }
}

/* */
WORD GridTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    GRID *pGrid;

    pGrid = (GRID *)pObj;

    // Evaluate if the message is for the check box
    // Check if disabled first
    if(GetState(pGrid, GRID_DISABLED))
    {
        return (OBJ_MSG_INVALID);
    }

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the check box borders
        if
        (
            (pGrid->hdr.left <= pMsg->param1) &&
            (pGrid->hdr.right >= pMsg->param1) &&
            (pGrid->hdr.top <= pMsg->param2) &&
            (pGrid->hdr.bottom >= pMsg->param2)
        )
        {
            return (GRID_MSG_TOUCHED);
        }

        return (OBJ_MSG_INVALID);
    }

        #endif
        #ifdef USE_KEYBOARD
    if((pMsg->uiEvent == EVENT_KEYSCAN) && (pMsg->type == TYPE_KEYBOARD) &&
((WORD)pMsg->param1 == pGrid->hdr.ID))
    {
        if((pMsg->param2 == SCAN_SPACE_PRESSED) || (pMsg->param2 == SCAN_CR_PRESSED))
        {
            return (GRID_MSG_ITEM_SELECTED);
        }
        else if(pMsg->param2 == SCAN_LEFT_PRESSED)
        {
            return (GRID_MSG_LEFT);
        }
        else if(pMsg->param2 == SCAN_RIGHT_PRESSED)
        {
            return (GRID_MSG_RIGHT);
        }
        else if(pMsg->param2 == SCAN_UP_PRESSED)
        {
            return (GRID_MSG_UP);
```

```
        }
        else if(pMsg->param2 == SCAN_DOWN_PRESSED)
        {
            return (GRID_MSG_DOWN);
        }
    }

    return (OBJ_MSG_INVALID);
        #endif
    return (OBJ_MSG_INVALID);
}

#endif // USE_GRID
```

14.1.19 Grid.h

Functions

	Name	Description
	GridClearCellState (see page 161)	This function clears the state of the cell (or Grid (see page 153) Item) specified by the column and row.
	GridCreate (see page 159)	This function creates a GRID (see page 169) 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 (see page 160)	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.
	GridFreeItems (see page 163)	This function removes all grid items for the given Grid (see page 153) and frees the memory used.
	GridGetCell (see page 163)	This function removes all grid items for the given Grid (see page 153) and frees the memory used.
	GridMsgDefault (see page 166)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	GridSetCell (see page 164)	This function sets the Grid (see page 153) Item state and data.
	GridSetCellState (see page 165)	This function sets the state of the Grid (see page 153) Item or cell.
	GridSetFocus (see page 166)	This function sets the focus of the specified Grid (see page 153) Item or cell.
	GridTranslateMsg (see page 167)	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
GRID_DISABLED (see page 155)	Bit for disabled state
GRID_DRAW_ALL (see page 156)	Bit to indicate whole edit box must be redrawn
GRID_DRAW_ITEMS (see page 156)	Bit to indicate that at least one item must be redrawn, but not the entire grid.
GRID_FOCUSED (see page 155)	Bit for focused state
GRID_HIDE (see page 156)	Bit to remove object from screen
GRID_OUT_OF_BOUNDS (see page 162)	Status of an out of bounds cell GridSetCell (see page 164)() operation.
GRID_SHOW_BORDER_ONLY (see page 156)	Draw only the outside border of the grid.

GRID_SHOW_FOCUS (see page 155)	Highlight the focused cell.
GRID_SHOW_LINES (see page 155)	Display grid lines
GRID_SHOW_SEPARATORS_ONLY (see page 156)	Draw only the lines between cells (like Tic-tac-toe)
GRID_SUCCESS (see page 163)	Status of a successful GridSetCell (see page 164)() operation.
GridGetFocusX (see page 161)	This macro returns the x position of the focused cell.
GridGetFocusY (see page 162)	This macro returns the y position of the focused cell.
GRIDITEM_DRAW (see page 159)	Draw this cell
GRIDITEM_IS_BITMAP (see page 158)	The grid item is a bitmap.
GRIDITEM_IS_TEXT (see page 157)	The grid item is a test string.
GRIDITEM_SELECTED (see page 157)	The cell is selected.
GRIDITEM_TEXTBOTTOM (see page 158)	Bit to indicate text is top aligned.
GRIDITEM_TEXTLEFT (see page 158)	Text in the cell is left aligned.
GRIDITEM_TEXTRIGHT (see page 158)	Text in the cell is right aligned.
GRIDITEM_TEXTTOP (see page 158)	Bit to indicate text is bottom aligned.

Structures

Name	Description
GRID (see page 169)	Defines the parameters required for a grid Object. Clipping is not supported in grid object.
GRIDITEM (see page 169)	Defines the grid item.

Description

This is file Grid.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Grid control
 *****************************************************************************
 * FileName:        Grid.h
 * Dependencies:    none
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 02/29/08    ...
 * 04/16/10    Added Grid Item as text.
 ***************************************************************************/
#ifndef _GRID_H_
    #define _GRID_H_

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

//***************************************************************************
// Grid Object State Definitions
//***************************************************************************
    #define GRID_FOCUSED                0x0001  // Bit for focused state
    #define GRID_DISABLED               0x0002  // Bit for disabled state
    #define GRID_SHOW_LINES             0x0004  // Display grid lines
    #define GRID_SHOW_FOCUS             0x0008  // Highlight the focused cell.
    #define GRID_SHOW_BORDER_ONLY       0x0010  // Draw only the outside border of the grid.
    #define GRID_SHOW_SEPARATORS_ONLY   0x0020  // Draw only the lines between cells (like
Tic-tac-toe)
    #define GRID_DRAW_ITEMS             0x1000  // Bit to indicate that at least one item
must be redrawn, but not the entire grid.
    #define GRID_DRAW_ALL               0x4000  // Bit to indicate whole edit box must be
redrawn
    #define GRID_HIDE                   0x8000  // Bit to remove object from screen

//***************************************************************************
// Grid Item State Definitions
//***************************************************************************
    #define GRIDITEM_SELECTED           0x0001  // The cell is selected.
    #define GRIDITEM_IS_TEXT            0x0000  // The grid item is a test string.
    #define GRIDITEM_IS_BITMAP          0x0008  // The grid item is a bitmap.
    #define GRIDITEM_TEXTRIGHT          0x0010  // Text in the cell is right aligned.
    #define GRIDITEM_TEXTLEFT           0x0020  // Text in the cell is left aligned.
    #define GRIDITEM_TEXTBOTTOM         0x0040  // Bit to indicate text is top aligned.
    #define GRIDITEM_TEXTTOP            0x0080  // Bit to indicate text is bottom aligned.
    #define GRIDITEM_DRAW               0x0100  // Draw this cell


    #define GRID_TYPE_MASK              (0x0C)
    #define GRID_SUCCESS                0x0000  // Status of a successful GridSetCell()
operation.
    #define GRID_OUT_OF_BOUNDS          0x0001  // Status of an out of bounds cell
GridSetCell() operation.

    #define GRID_MAX_COLUMNS(rw, cw)    ((rw - (GOL_EMBOSS_SIZE * 2) + 1) / (cw + 1))
    #define GRID_MAX_ROWS(rh, ch)       ((rh - (GOL_EMBOSS_SIZE * 2) + 1) / (ch + 1))
    #define GRID_WIDTH(c, cw)           ((GOL_EMBOSS_SIZE * 2) + (c * (cw + 1)) - 1)
    #define GRID_HEIGHT(r, ch)          ((GOL_EMBOSS_SIZE * 2) + (r * (ch + 1)) - 1)

/*********************************************************************
* Overview: Defines the grid item.
*
*********************************************************************/
typedef struct
{
    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
} GRIDITEM;

/*********************************************************************
* Overview: Defines the parameters required for a grid Object.
*           Clipping is not supported in grid object.
*
*********************************************************************/
typedef struct
{
    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
} GRID;

/*********************************************************************
* Function: 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)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        numColumns - Sets the number of columns for the grid.
*        numRows - Sets the number of rows for the grid.
*        cellWidth - Sets the width of each cell of the grid.
*        cellHeight - Sets the height of each cell of the grid.
*        pScheme - Pointer to the style scheme used for the object.
*                  Set to NULL if default style scheme is used.
*
* Output: Returns the pointer to the object created.
*
* Side Effects: none
*
*********************************************************************/
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
        );

/*********************************************************************
* Function: GridClearCellState(GRID *pGrid, SHORT column, SHORT row, WORD state)
*
* Overview: This function clears the state of the cell (or Grid Item) specified
*           by the column and row.
*
* PreCondition: none
*
* Input: pGrid - Pointer to the object.
*        column - column index of the cell
*        row - row index of the cell
*        atate - specifies the state to be cleared. See Grid Item State.
*
* Output: none.
*
* Side Effects: none
*
*********************************************************************/
void    GridClearCellState(GRID *pGrid, SHORT column, SHORT row, WORD state);
```

```
/*********************************************************************
* Function: WORD GridDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pGb - Pointer to the object to be rendered.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
WORD GridDraw(void *pObj);

/*********************************************************************
* Function: GridFreeItems(void *pObj)
*
* Overview: This function removes all grid items for the given Grid
*           and frees the memory used.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pGrid - The pointer to the Grid object.
*
* Output: none.
*
* Side Effects: none
*
*********************************************************************/
void    GridFreeItems(void *pObj);

/*********************************************************************
* Function: *GridGetCell(GRID *pGrid, SHORT column, SHORT row, WORD *cellType)
*
* Overview: This function removes all grid items for the given Grid
*           and frees the memory used.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pGrid - The pointer to the Grid object.
*        column - the column index of the cell
*        row - the row index of the cell
*        cellType - pointer that will receive the type of grid item
*                   or cell (GRIDITEM_IS_TEXT or GRIDITEM_IS_BITMAP).
*
* Output: Returns a pointer to the grid item or cell data.
*
* Side Effects: none
*
*********************************************************************/
void    *GridGetCell(GRID *pGrid, SHORT column, SHORT row, WORD *cellType);

/*********************************************************************
* Macros:  GridGetFocusX(pGrid)
*
* Overview: This macro returns the x position of the focused cell.
```

```
 *
 * PreCondition: none
 *
 * Input: pGrid - Pointer to the object.
 *
 * Output: Returns the x position of the focused cell.
 *
 * Side Effects: none
 *
 ********************************************************************/
#define GridGetFocusX(pGrid)    pGrid->focusX

/********************************************************************
 * Macros:  GridGetFocusY(pGrid)
 *
 * Overview: This macro returns the y position of the focused cell.
 *
 * PreCondition: none
 *
 * Input: pGrid - Pointer to the object.
 *
 * Output: Returns the y position of the focused cell.
 *
 * Side Effects: none
 *
 ********************************************************************/
#define GridGetFocusY(pGrid)    pGrid->focusY

/********************************************************************
 * Function: GridMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
 *
 * Overview: This function performs the actual state change
 *           based on the translated message given. The following state changes
 *           are supported:
 *   <TABLE>
 *       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 Item selected
will be redrawn.
 *                                               GRID_DRAW_ITEMS
 *       GRID_MSG_UP             Keyboard        Set GRIDITEM_DRAW,      Grid Item above the
currently focused item will be redrawn.
 *                                               GRID_DRAW_ITEMS
 *       GRID_MSG_DOWN           Keyboard        Set GRIDITEM_DRAW,      Grid Item below the
currently focused item will be redrawn.
 *                                               GRID_DRAW_ITEMS
 *       GRID_MSG_LEFT           Keyboard        Set GRIDITEM_DRAW,      Grid Item to the
left of the currently focused item will be redrawn.
 *                                               GRID_DRAW_ITEMS
 *       GRID_MSG_RIGHT          Keyboard        Set GRIDITEM_DRAW,      Grid Item to the
right of the currently focused item will be redrawn.
 *
GRID_DRAW_ITEMS
 *   </TABLE>
 *
 * PreCondition: none
 *
 * Input: translatedMsg - The translated message.
 *        pGrid         - The pointer to the object whose state will be modified.
 *        pMsg          - The pointer to the GOL message.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
void    GridMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/********************************************************************
 * Function: GridTranslateMsg(void *pObj, GOL_MSG *pMsg)
```

```
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pGrid - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
*
* Side Effects: none
*
*********************************************************************/
WORD    GridTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD    GridSetCell(GRID *pGrid, SHORT column, SHORT row, WORD state, void
*data)
*
* Overview: This function sets the Grid Item state and data.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pGrid - The pointer to the Grid object.
*        column - the column index of the cell
*        row - the row index of the cell
*        state - sets the state of the Grid Item specified.
*        data - pointer to the data used for the Grid Item.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
WORD    GridSetCell(GRID *pGrid, SHORT column, SHORT row, WORD state, void *data);

/*********************************************************************
* Function: GridSetCellState(GRID *pGrid, SHORT column, SHORT row, WORD state)
*
* Overview: This function sets the state of the Grid Item or cell.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pGrid - The pointer to the Grid object.
*        column - the column index of the cell
*        row - the row index of the cell
*        state - sets the state of the Grid Item specified.
*
* Output: none.
*
* Side Effects: none
*
*********************************************************************/
void    GridSetCellState(GRID *pGrid, SHORT column, SHORT row, WORD state);

/*********************************************************************
* Function: GridSetFocus(GRID *pGrid, SHORT column, SHORT row)
*
* Overview: This function sets the focus of the specified Grid Item or cell.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pGrid - The pointer to the Grid object.
*        column - the column index of the cell
*        row - the row index of the cell
*
* Output: none.
*
* Side Effects: none
*
*********************************************************************/
void    GridSetFocus(GRID *pGrid, SHORT column, SHORT row);

#endif
```

14.1.20 GroupBox.c

This is file GroupBox.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Group Box
 *****************************************************************************
 * FileName:        GroupBox.c
 * Dependencies:
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paolo A. Tamayo      11/12/07    Version 1.0 release
 ***************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_GROUPBOX

/*********************************************************************
* Function: GROUPBOX *GbCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                              SHORT bottom, WORD state, XCHAR *pText,
*                              GOL_SCHEME *pScheme)
*
* Notes: Creates a GROUPBOX object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*********************************************************************/
GROUPBOX *GbCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    GROUPBOX    *pGb = NULL;

    pGb = (GROUPBOX *)GFX_malloc(sizeof(GROUPBOX));
    if(pGb == NULL)
        return (pGb);

    pGb->hdr.ID = ID;                   // unique id assigned for referencing
    pGb->hdr.pNxtObj = NULL;
    pGb->hdr.type = OBJ_GROUPBOX;       // set object type
    pGb->hdr.left = left;               // left position
    pGb->hdr.top = top;                 // top position
    pGb->hdr.right = right;             // right position
    pGb->hdr.bottom = bottom;           // bottom position
    pGb->hdr.state = state;             // initial state
    pGb->pText = pText;                 // text label used
    pGb->hdr.DrawObj = GbDraw;          // draw function
    pGb->hdr.MsgObj = GbTranslateMsg;   // message function
    pGb->hdr.MsgDefaultObj = NULL;      // default message function
    pGb->hdr.FreeObj = NULL;            // free function

    // Set the color scheme and font to be used
    if(pScheme == NULL)
```

```
        pGb->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pGb->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pGb->textWidth = 0;
    pGb->textHeight = 0;
    if(pText != NULL)
    {

        // Set the text width & height
        pGb->textWidth = GetTextWidth(pText, pGb->hdr.pGolScheme->pFont);
        pGb->textHeight = GetTextHeight(pGb->hdr.pGolScheme->pFont);
    }

    GOLAddObject((OBJ_HEADER *)pGb);
    return (pGb);
}

/*****************************************************************
* Function: WORD GbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*****************************************************************/
WORD GbTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    GROUPBOX *pGb;

    pGb = (GROUPBOX *)pObj;

    // Evaluate if the message is for the button
    // Check if disabled first
    if(!GetState(pGb, GB_DISABLED))
    {
            #ifdef USE_TOUCHSCREEN
        if(pMsg->type == TYPE_TOUCHSCREEN)
        {

            // Check if it falls to the left or right of the center of the thumb's face
            if
            (
                (pGb->hdr.left < pMsg->param1) &&
                (pGb->hdr.right > pMsg->param1) &&
                (pGb->hdr.top < pMsg->param2) &&
                (pGb->hdr.bottom > pMsg->param2)
            )
            {
                if((pMsg->uiEvent == EVENT_PRESS) || (pMsg->uiEvent == EVENT_RELEASE))
                    return (GB_MSG_SELECTED);
            }
        }

            #endif
    }

    return (OBJ_MSG_INVALID);
}

/*****************************************************************
* Function: WORD GbDraw(void *pObj)
*
* Notes: This is the state machine to draw the button.
*
*****************************************************************/
    #if (COLOR_DEPTH == 1)
        #define THREE_D_EFFECT  0
    #else
        #define THREE_D_EFFECT  1
    #endif
```

```
/* */
WORD GbDraw(void *pObj)
{
    typedef enum
    {
        GB_STATE_IDLE,
        GB_STATE_HIDETEXT,
        GB_STATE_SETDIMENSION,
        GB_STATE_DRAWTEXT,
        GB_STATE_DRAWTOPRIGHT,
        GB_STATE_DRAWTOPLEFT,
        GB_STATE_DRAWSIDELEFT,
        GB_STATE_DRAWSIDERIGHT,
        GB_STATE_DRAWBOTTOM,
            #if (COLOR_DEPTH != 1)
        GB_STATE_2DRAWTOPRIGHT,
        GB_STATE_2DRAWTOPLEFT,
        GB_STATE_2DRAWSIDELEFT,
        GB_STATE_2DRAWSIDERIGHT,
        GB_STATE_2DRAWBOTTOM,
            #endif
    } GB_DRAW_STATES;

    static GB_DRAW_STATES state = GB_STATE_IDLE;
    static SHORT textLeft, textRight, top;                      // used to draw lines that
start/stops at text.
    GROUPBOX *pGb;

    pGb = (GROUPBOX *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case GB_STATE_IDLE:

            if(GetState(pGb, GB_HIDE))
            {                                                   // Hide the Group Box
(remove from screen)
                SetColor(pGb->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pGb->hdr.left, pGb->hdr.top, pGb->hdr.right, pGb->hdr.bottom))
                    return 0;
                return (1);
            }

            state = GB_STATE_HIDETEXT;

        case GB_STATE_HIDETEXT:                                 // hide the text first
            if(pGb->pText != NULL)
            {                                                   // needed when dynamically
changing
                SetColor(pGb->hdr.pGolScheme->CommonBkColor);   // the alignement of text
                if(!Bar(pGb->hdr.left, pGb->hdr.top, pGb->hdr.right, pGb->hdr.top +
pGb->textHeight))
                    return 0;
            }

            state = GB_STATE_SETDIMENSION;

        case GB_STATE_SETDIMENSION:
            if(IsDeviceBusy())
                return (0);

            if(GetState(pGb, GB_DISABLED))
            {   // set color to inactive color
                SetColor(pGb->hdr.pGolScheme->TextColorDisabled);
            }
            else
            {
                SetColor(pGb->hdr.pGolScheme->TextColor0);  // active color
            }
```

```
            if(pGb->pText == NULL)
            {   // there is no text, use full dimensions
                top = pGb->hdr.top;
                textLeft = pGb->hdr.left + 1;
                textRight = textLeft;
                state = GB_STATE_DRAWTOPRIGHT;  // go to drawing of right top line
                goto gb_state_draw_lines;
            }
            else
            {   // text is present, set up dimensions with text
                SetFont(pGb->hdr.pGolScheme->pFont);
                top = pGb->hdr.top + (pGb->textHeight >> 1);
// adjust lines on top
                if(pGb->hdr.state & GB_RIGHT_ALIGN)
                {

                    // do right aligned
                    textLeft = pGb->hdr.right - pGb->textWidth - 2;
                    textRight = pGb->hdr.right - 2;
                }
                else if(pGb->hdr.state & GB_CENTER_ALIGN)
                {

                    // do center aligned
                    textLeft = (pGb->hdr.left + pGb->hdr.right - pGb->textWidth) >> 1;
                    textRight = textLeft + pGb->textWidth;
                }
                else
                {

                    // do left aligned
                    textLeft = pGb->hdr.left + 2;
                    textRight = pGb->hdr.left + 2 + pGb->textWidth;
                }

                MoveTo(textLeft, pGb->hdr.top);
// move cursor to start of text
                state = GB_STATE_DRAWTEXT;
            }

        case GB_STATE_DRAWTEXT:
            if(!OutText(pGb->pText))
                return (0);
                #if (COLOR_DEPTH == 1)
            SetColor(WHITE);
                #else
            SetColor(pGb->hdr.pGolScheme->EmbossDkColor);
                #endif
            state = GB_STATE_DRAWTOPRIGHT;

    gb_state_draw_lines:

        case GB_STATE_DRAWTOPRIGHT:
            if(!Line(textRight, top + THREE_D_EFFECT, pGb->hdr.right, top + THREE_D_EFFECT))
                return 0;    // top line at right
            state = GB_STATE_DRAWTOPLEFT;

        case GB_STATE_DRAWTOPLEFT:
            if(!Line(pGb->hdr.left + THREE_D_EFFECT, top + THREE_D_EFFECT, textLeft, top +
THREE_D_EFFECT))
                return 0;                       // top line at left
            state = GB_STATE_DRAWSIDELEFT;

        case GB_STATE_DRAWSIDELEFT:
            if(!Line(pGb->hdr.left + THREE_D_EFFECT, top + THREE_D_EFFECT, pGb->hdr.left +
THREE_D_EFFECT, pGb->hdr.bottom))
                return 0;    // side line at left
            state = GB_STATE_DRAWSIDERIGHT;

        case GB_STATE_DRAWSIDERIGHT:
            if(!Line(pGb->hdr.right, top + THREE_D_EFFECT, pGb->hdr.right, pGb->hdr.bottom))
```

```
                return 0;            // side line at right
            state = GB_STATE_DRAWBOTTOM;

        case GB_STATE_DRAWBOTTOM:
            if(!Line(pGb->hdr.left + THREE_D_EFFECT, pGb->hdr.bottom, pGb->hdr.right,
pGb->hdr.bottom))
                return 0; // bottom line
                #if (COLOR_DEPTH == 1)
            state = GB_STATE_IDLE;
                #else
            state = GB_STATE_2DRAWTOPLEFT;
                #endif
                #if (COLOR_DEPTH != 1)

        case GB_STATE_2DRAWTOPLEFT:
            SetColor(pGb->hdr.pGolScheme->EmbossLtColor);                                //
2nd line top line at left
            if(!Line(pGb->hdr.left, top, textLeft, top))
                return 0;
            state = GB_STATE_2DRAWTOPRIGHT;

        case GB_STATE_2DRAWTOPRIGHT:
            if(!Line(textRight, top, pGb->hdr.right, top))
                return 0;                                   // 2nd line top line at right
            state = GB_STATE_2DRAWSIDELEFT;

        case GB_STATE_2DRAWSIDELEFT:
            if(!Line(pGb->hdr.left, top, pGb->hdr.left, pGb->hdr.bottom - 1))
                return 0;               // 2nd line left
            state = GB_STATE_2DRAWSIDERIGHT;

        case GB_STATE_2DRAWSIDERIGHT:
            if(!Line(pGb->hdr.right - 1, top + 2, pGb->hdr.right - 1, pGb->hdr.bottom - 1))
                return 0; // 2nd line right
            state = GB_STATE_2DRAWBOTTOM;

        case GB_STATE_2DRAWBOTTOM:
            if(!Line(pGb->hdr.left + 2, pGb->hdr.bottom - 1, pGb->hdr.right - 1,
pGb->hdr.bottom - 1))
                return 0;  // 2nd line bottom
            state = GB_STATE_IDLE;
                #endif
    }

    return (1);
}

#endif // USE_GROUPBOX
```

14.1.21 GroupBox.h

Functions

	Name	Description
	GbCreate (see page 172)	This function creates a GROUPBOX (see page 176) 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 (see page 174)	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 (see page 175)	This function sets the text used by passing the pointer to the static string.

	GbTranslateMsg (see page 176)	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
GB_CENTER_ALIGN (see page 171)	Bit to indicate text is center aligned
GB_DISABLED (see page 172)	Bit for disabled state
GB_DRAW (see page 172)	Bit to indicate group box must be redrawn
GB_HIDE (see page 172)	Bit to remove object from screen
GB_RIGHT_ALIGN (see page 172)	Bit to indicate text is right aligned
GbGetText (see page 174)	This macro returns the location of the text used.

Structures

Name	Description
GROUPBOX (see page 176)	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

This is file GroupBox.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Group Box
 *****************************************************************************
 * FileName:        GroupBox.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 *****************************************************************************/
#ifndef _GROUPBOX_H
```

```
    #define _GROUPBOX_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*****************************************************************
* Object States Definition:
*****************************************************************/
    #define GB_DISABLED     0x0002  // Bit for disabled state
    #define GB_RIGHT_ALIGN  0x0004  // Bit to indicate text is right aligned
    #define GB_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned

// When center and right bits are zero alignment is left
    #define GB_DRAW 0x4000          // Bit to indicate group box must be redrawn
    #define GB_HIDE 0x8000          // Bit to remove object from screen

/*****************************************************************
* 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.
*
*****************************************************************/
typedef struct
{
    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.
} GROUPBOX;

/*****************************************************************
* Macros:  GbGetText(pGb)
*
* Overview: This macro returns the location of the text used.
*
* PreCondition: none
*
* Input: pGb - Pointer to the object.
*
* Output: Returns the address of the text string used.
*
* Side Effects: none
*
*****************************************************************/
    #define GbGetText(pB)   pGb->pText

/*****************************************************************
* Function: GbSetText(GROUPBOX *pGb, XCHAR *pText)
*
* Overview: This function sets the text used by passing the pointer
*           to the static string.
*
* PreCondition: none
*
* Input: pGb - the pointer to the object whose state will be modified.
*        pText - pointer to the text that will be used.
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
void        GbSetText(GROUPBOX *pGb, XCHAR *pText);

/*****************************************************************
* Function: WORD GbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pGb   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD        GbTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: GROUPBOX *GbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom,
*                              WORD state, XCHAR *pText, GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        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.
*        pScheme - Pointer to the style scheme used for the object.
*                  Set to NULL if default style scheme is used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   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;
 *   </CODE>
 *
 * Side Effects: none
 *
 ********************************************************************/
GROUPBOX     *GbCreate
             (
                 WORD        ID,
                 SHORT       left,
                 SHORT       top,
                 SHORT       right,
                 SHORT       bottom,
                 WORD        state,
                 XCHAR       *pText,
                 GOL_SCHEME  *pScheme
             );

/********************************************************************
 * Function: WORD GbDraw(void *pObj)
 *
 * 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.
 *
 * PreCondition: Object must be created before this function is called.
 *
 * Input: pGb - Pointer to the object to be rendered.
 *
 * Output: 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.
 *
 * Example:
 *   See GbCreate() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
WORD GbDraw(void *pObj);
#endif //_GROUPBOX_H
```

14.1.22 ListBox.c

This is file ListBox.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  List Box
 *****************************************************************************
 * FileName:        ListBox.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 11/18/10     Fixed bug on icons drawn after the bottom edge of the list box.
 * 12/21/10     Fixed bug on the list count to rest back to zero when
 *              LbDelItemsList() is used to clear the list.
 * 04/20/11     Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 ****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_LISTBOX

/*********************************************************************
* Function: LISTBOX  *LbCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom,
*                              WORD state, XCHAR* pText, GOL_SCHEME *pScheme)
*
* Overview: creates the list box
*
*********************************************************************/
LISTBOX *LbCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    LISTBOX *pLb = NULL;
    XCHAR   *pointer;
    WORD    counter;

    pLb = (LISTBOX *)GFX_malloc(sizeof(LISTBOX));

    if(pLb == NULL)
        return (pLb);

    pLb->hdr.ID         = ID;
    pLb->hdr.pNxtObj    = NULL;
    pLb->hdr.type       = OBJ_LISTBOX;
    pLb->hdr.left       = left;
    pLb->hdr.top        = top;
```

```
    pLb->hdr.right      = right;
    pLb->hdr.bottom     = bottom;
    pLb->hdr.state      = state;
    pLb->pItemList      = NULL;
    pLb->scrollY        = 0;
    pLb->itemsNumber    = 0;
    pLb->hdr.DrawObj    = LbDraw;           // draw function
    pLb->hdr.MsgObj     = LbTranslateMsg;   // message function
    pLb->hdr.MsgDefaultObj = LbMsgDefault;  // default message function
    pLb->hdr.FreeObj    = LbDelItemsList;   // free function

    // Set the style scheme to be used
    if(pScheme == NULL)
        pLb->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pLb->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pLb->textHeight = GetTextHeight(pLb->hdr.pGolScheme->pFont);

    GOLAddObject((OBJ_HEADER *)pLb);

    // Add items if there's an initialization text (each line will become one item,
    // lines must be separated by '\n' character)
    if(pText != NULL)
    {
        pointer = pText;
        counter = 0;
        while(*pointer)
        {
            if(NULL == LbAddItem(pLb, NULL, pointer, NULL, 0, counter))
                break;
            while((XCHAR) *pointer++ > (XCHAR)31);
            if(*(pointer - 1) == 0)
                break;
            counter++;
        }
    }

    pLb->pFocusItem = pLb->pItemList;

    // Set focus for the object if FOCUSED state is set
        #ifdef USE_FOCUS
    if(GetState(pLb, LB_FOCUSED))
        GOLSetFocus((OBJ_HEADER *)pLb);
        #endif
    return (pLb);
}

/*********************************************************************
* Function: LISTITEM* LbAddItem(LISTBOX *pLb, LISTITEM *pPrevItem, XCHAR *pText, void*
pBitmap, WORD status, WORD data)
*
* Input: pLb - the pointer to the list box,
*        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,
*        pText -  pointer to the text,
*        pBitmap - pointer to the bitmap,
*        status - status after creation,
*        data -  some data associated with the item
*
* Output: pointer to the item created, NULL if the operation was not successful
*
* Overview: allocates memory for a new item and adds it to the list box
*
*********************************************************************/
LISTITEM *LbAddItem(LISTBOX *pLb, LISTITEM *pPrevItem, XCHAR *pText, void *pBitmap, WORD
status, WORD data)
{
    LISTITEM    *pItem;
    LISTITEM    *pCurItem;
```

```
    pItem = (LISTITEM *)GFX_malloc(sizeof(LISTITEM));

    if(pItem == NULL)
    {
        return (NULL);
    }

    pLb->itemsNumber++;

    if(pLb->pItemList == NULL)
    {
        pLb->pItemList = pItem;
        pItem->pNextItem = NULL;
        pItem->pPrevItem = NULL;
    }
    else
    {
        if(pPrevItem == NULL)
        {

            // Find last item
            pCurItem = (LISTITEM *)pLb->pItemList;
            while(pCurItem->pNextItem != NULL)
                pCurItem = (LISTITEM *)pCurItem->pNextItem;
        }
        else
        {
            pCurItem = (LISTITEM *)pPrevItem;
        }

        // Insert a new item
        pItem->pNextItem = pCurItem->pNextItem;
        pItem->pPrevItem = pCurItem;
        pCurItem->pNextItem = pItem;
    }

    pItem->pText = pText;
    pItem->pBitmap = pBitmap;
    pItem->status = status;
    pItem->data = data;

    return (pItem);
}

/*********************************************************************
* Function: void LbDelItem(LISTBOX *pLb, LISTITEM *pItem)
*
* Input: pLb - the pointer to the object,
*        pItem - the pointer to the item must be deleted
*
* Output: none
*
* Overview: removes an item from the list box and frees memory
*
*********************************************************************/
void LbDelItem(LISTBOX *pLb, LISTITEM *pItem)
{
    if(pItem->pNextItem != NULL)
    {
        ((LISTITEM *)pItem->pNextItem)->pPrevItem = pItem->pPrevItem;
        if(pItem->pPrevItem == NULL)
            pLb->pItemList = (LISTITEM *)pItem->pNextItem;
    }

    if(pItem->pPrevItem != NULL)
        ((LISTITEM *)pItem->pPrevItem)->pNextItem = pItem->pNextItem;

    GFX_free(pItem);

    pLb->itemsNumber--;

    if(pLb->itemsNumber == 0)
```

```
        pLb->pItemList = NULL;
}

/*****************************************************************
* Function: void LbDelItemsList(void *pObj)
*
* Input: pLb - the pointer to the object
*
* Output: none
*
* Overview: removes all items from list box and frees memory
*
*****************************************************************/
void LbDelItemsList(void *pObj)
{
    LISTITEM    *pCurItem;
    LISTITEM    *pItem;
    LISTBOX     *pLb;

    pLb = (LISTBOX *)pObj;

    pCurItem = pLb->pItemList;

    while(pCurItem != NULL)
    {
        pItem = pCurItem;
        pCurItem = (LISTITEM *)pCurItem->pNextItem;
        GFX_free(pItem);
    }

    pLb->pItemList = NULL;
    pLb->itemsNumber = 0;
}

/*****************************************************************
* Function: LISTITEM* LbGetSel(LISTBOX *pLb, LISTITEM *pFromItem)
*
* Input: pLb - the pointer to the object,
*        pFromItem - pointer to the item the seach must start from,
*                    if the pointer is NULL the search begins from the start of the items
list
*
* Output: pointer to the selected item, NULL if there are no items selected
*
* Overview: seaches for selected items
*
*****************************************************************/
LISTITEM *LbGetSel(LISTBOX *pLb, LISTITEM *pFromItem)
{
    if(pFromItem == NULL)
    {
        pFromItem = pLb->pItemList;
    }

    while(pFromItem != NULL)
    {
        if(pFromItem->status & LB_STS_SELECTED)
            break;
        pFromItem = (LISTITEM *)pFromItem->pNextItem;
    }

    return (pFromItem);
}

/*****************************************************************
* Function: void LbChangeSel(LISTBOX *pLb, LISTITEM *pItem)
*
* Input: pLb - the pointer to the object,
*        pItem - pointer to the item the selection status will be changed
*
* Output: none
*
```

```
* Overview: changes selection status of an item
*
*********************************************************************/
void LbChangeSel(LISTBOX *pLb, LISTITEM *pItem)
{
    LISTITEM    *pCurItem;

    if(GetState(pLb, LB_SINGLE_SEL))
    {

        // Remove selection from all items
        pCurItem = pLb->pItemList;
        while(pCurItem != NULL)
        {
            if(pCurItem->status & LB_STS_SELECTED)
            {
                pCurItem->status &= ~LB_STS_SELECTED;
                pCurItem->status |= LB_STS_REDRAW;
            }

            pCurItem = (LISTITEM *)pCurItem->pNextItem;
        }
    }

    pItem->status ^= LB_STS_SELECTED;
    pItem->status |= LB_STS_REDRAW;
}

/*********************************************************************
* Function: void LbSetFocusedItem(LISTBOX* pLb, SHORT index)
*
* Input: pLb - the pointer to the object
*        index - item number from the list beginning
*
* Output: none
*
* Overview: sets focus for the item with index defined
*
*********************************************************************/
void LbSetFocusedItem(LISTBOX *pLb, SHORT index)
{
    LISTITEM    *pCurItem;

    // Look for item to be focused
    pCurItem = pLb->pItemList;

    if(pCurItem == NULL)
        return;

    while(pCurItem->pNextItem != NULL)
    {
        if(index <= 0)
            break;
        index--;
        pCurItem = (LISTITEM *)pCurItem->pNextItem;
    }

    if(pCurItem != NULL)
    {
        if(pLb->pFocusItem != NULL)
        {
            pLb->pFocusItem->status |= LB_STS_REDRAW;
        }

        pLb->pFocusItem = pCurItem;
        pCurItem->status |= LB_STS_REDRAW;
    }
}

/*********************************************************************
* Function: SHORT LbGetFocusedItem(LISTBOX* pLb)
*
```

```
* Input: pLb - the pointer to the list box
*
* Output: index of the focused item, -1 if there's no item in focus
*
* Overview: returns focused item number from the list beginning
*
*********************************************************************/
SHORT LbGetFocusedItem(LISTBOX *pLb)
{
    LISTITEM    *pCurItem;
    SHORT       index;

    if(pLb->pFocusItem == NULL)
        return (-1);

    // Look for the focused item index
    index = 0;
    pCurItem = pLb->pItemList;
    while(pCurItem->pNextItem != NULL)
    {
        if(pCurItem == pLb->pFocusItem)
            break;
        index++;
        pCurItem = (LISTITEM *)pCurItem->pNextItem;
    }

    return (index);
}

/*********************************************************************
* Function: WORD LbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Overview: translates the GOL message for the list box
*
*********************************************************************/
WORD LbTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    LISTBOX *pLb;

    pLb = (LISTBOX *)pObj;

    // Evaluate if the message is for the list box
    // Check if disabled first
    if(GetState(pLb, LB_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in list box borders
        if
        (
            (pLb->hdr.left < pMsg->param1) &&
            (pLb->hdr.right > pMsg->param1) &&
            (pLb->hdr.top < pMsg->param2) &&
            (pLb->hdr.bottom > pMsg->param2)
        ) return (LB_MSG_TOUCHSCREEN);

        return (OBJ_MSG_INVALID);
    }

        #endif
        #ifdef USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD)
        if((WORD)pMsg->param1 == pLb->hdr.ID)
        {
            if(pMsg->uiEvent == EVENT_KEYSCAN)
            {
                if((pMsg->param2 == SCAN_UP_PRESSED) || (pMsg->param2 == SCAN_DOWN_PRESSED))
                    return (LB_MSG_MOVE);
```

```
                if((pMsg->param2 == SCAN_SPACE_PRESSED) || (pMsg->param2 ==
SCAN_CR_PRESSED))
                    return (LB_MSG_SEL);
            }

            return (OBJ_MSG_INVALID);
        }

        #endif
    return (OBJ_MSG_INVALID);
}

/*****************************************************************
* Function: void LbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
*
* Overview: changes the state of the list box by default
*
*****************************************************************/
void LbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{

    LISTBOX *pLb;

    pLb = (LISTBOX *)pObj;

        #ifdef USE_TOUCHSCREEN

    SHORT       pos;
    LISTITEM    *pItem;

    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
                #ifdef USE_FOCUS
        if(pMsg->type == TYPE_TOUCHSCREEN)
        {
            if(!GetState(pLb, LB_FOCUSED))
            {
                GOLSetFocus((OBJ_HEADER *)pLb);
            }
        }

                #endif
        if(pMsg->uiEvent == EVENT_PRESS)
        {
            pos = (pMsg->param2 - pLb->scrollY - pLb->hdr.top - LB_INDENT -
GOL_EMBOSS_SIZE) / pLb->textHeight;
            pItem = (LISTITEM *)pLb->pItemList;
            while(pos)
            {
                if(pItem == NULL)
                    break;
                if(pItem->pNextItem == NULL)
                    break;
                pItem = (LISTITEM *)pItem->pNextItem;
                pos--;
            }

            if(pLb->pFocusItem != pItem)
            {
                pItem->status |= LB_STS_REDRAW;
                pLb->pFocusItem->status |= LB_STS_REDRAW;
                pLb->pFocusItem = pItem;
                SetState(pLb, LB_DRAW_ITEMS);
            }

            LbChangeSel(pLb, pItem);
            SetState(pLb, LB_DRAW_ITEMS);
        }
    }

        #endif
        #ifdef USE_KEYBOARD
```

```
    if(pMsg->type == TYPE_KEYBOARD)
    {
        switch(translatedMsg)
        {
            case LB_MSG_SEL:
                if(pLb->pFocusItem != NULL)
                    LbChangeSel(pLb, pLb->pFocusItem);
                SetState(pLb, LB_DRAW_ITEMS);
                break;

            case LB_MSG_MOVE:
                switch(pMsg->param2)
                {
                    case SCAN_UP_PRESSED:
                        LbSetFocusedItem(pLb, LbGetFocusedItem(pLb) - 1);
                        SetState(pLb, LB_DRAW_ITEMS);
                        break;

                    case SCAN_DOWN_PRESSED:
                        LbSetFocusedItem(pLb, LbGetFocusedItem(pLb) + 1);
                        SetState(pLb, LB_DRAW_ITEMS);
                        break;
                }

                break;
        }
    }   // end of if
        #endif
}

/*****************************************************************
* Function: WORD LbDraw(void *pObj)
*
* Output: returns the status of the drawing
*         0 - not completed
*         1 - done
*
* Overview: draws edit box
*
*****************************************************************/
WORD LbDraw(void *pObj)
{
    typedef enum
    {
        LB_STATE_START,
        LB_STATE_PANEL,
        LB_STATE_ERASEITEM,
        LB_STATE_ITEMFOCUS,
        LB_STATE_BITMAP,
        LB_STATE_TEXT
    } LB_DRAW_STATES;

    static LB_DRAW_STATES state = LB_STATE_START;
    static LISTITEM *pCurItem;
    static SHORT temp;
    LISTBOX *pLb;

    pLb = (LISTBOX *)pObj;

    switch(state)
    {

        ////////////////////////////////////////////////////////////////
        // REMOVE FROM SCREEN
        ////////////////////////////////////////////////////////////////
        case LB_STATE_START:
            if(GetState(pLb, LB_HIDE))
            {
                SetColor(pLb->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pLb->hdr.left, pLb->hdr.top, pLb->hdr.right, pLb->hdr.bottom))
                    return (0);
                return (1);
```

```
            }

            if(GetState(pLb, LB_DRAW_FOCUS))
            {
                if(pLb->pFocusItem != NULL)
                    ((LISTITEM *)pLb->pFocusItem)->status |= LB_STS_REDRAW;
                SetState(pLb, LB_DRAW_ITEMS);
            }

            ////////////////////////////////////////////////////////////////////
            // DRAW PANEL
            ////////////////////////////////////////////////////////////////////
            if(GetState(pLb, LB_DRAW))
            {
                if(GetState(pLb, LB_DISABLED))
                {
                    temp = pLb->hdr.pGolScheme->ColorDisabled;
                }
                else
                {
                    temp = pLb->hdr.pGolScheme->Color0;
                }

                GOLPanelDraw
                (
                    pLb->hdr.left,
                    pLb->hdr.top,
                    pLb->hdr.right,
                    pLb->hdr.bottom,
                    0,
                    temp,
                    pLb->hdr.pGolScheme->EmbossDkColor,
                    pLb->hdr.pGolScheme->EmbossLtColor,
                    NULL,
                    GOL_EMBOSS_SIZE
                );

                state = LB_STATE_PANEL;

            case LB_STATE_PANEL:
                if(!GOLPanelDrawTsk())
                    return (0);
            }

            ////////////////////////////////////////////////////////////////////
            // DRAW ITEMS
            ////////////////////////////////////////////////////////////////////
    L_LB_DRAW:
            SetClip(CLIP_ENABLE);

            SetClipRgn
            (
                pLb->hdr.left + GOL_EMBOSS_SIZE + LB_INDENT,
                pLb->hdr.top + GOL_EMBOSS_SIZE + LB_INDENT,
                pLb->hdr.right - GOL_EMBOSS_SIZE - LB_INDENT,
                pLb->hdr.bottom - GOL_EMBOSS_SIZE - LB_INDENT
            );

            SetFont(pLb->hdr.pGolScheme->pFont);

            // Set graphics cursor
            MoveTo(pLb->hdr.left + GOL_EMBOSS_SIZE + LB_INDENT, pLb->hdr.top +
GOL_EMBOSS_SIZE + LB_INDENT + pLb->scrollY);

            pCurItem = pLb->pItemList;

            ////////////////////////////////////////////////////////////////////
            // DRAW CURRENT ITEM
            ////////////////////////////////////////////////////////////////////
    L_LB_DRAWITEM:
            if(pCurItem == NULL)
            {
```

```
                state = LB_STATE_START;
                SetClip(CLIP_DISABLE);
                return (1);
            }

            ////////////////////////////////////////////////////////////////////////
            if(GetY() < pLb->hdr.bottom - GOL_EMBOSS_SIZE - LB_INDENT)
                if((GetY() + pLb->textHeight) >= (pLb->hdr.top + GOL_EMBOSS_SIZE +
LB_INDENT))
                {
                    if(!GetState(pLb, LB_DRAW))
                        if(!(pCurItem->status & LB_STS_REDRAW))
                            goto L_LB_NEXTITEM;

                    pCurItem->status &= ~LB_STS_REDRAW;

                    state = LB_STATE_ERASEITEM;

                case LB_STATE_ERASEITEM:
                    if(IsDeviceBusy())
                        return (0);

                    if(GetState(pLb, LB_DISABLED))
                    {
                        SetColor(pLb->hdr.pGolScheme->ColorDisabled);
                    }
                    else
                    {
                        SetColor(pLb->hdr.pGolScheme->Color0);
                        if(pCurItem != NULL)
                            if(pCurItem->status & LB_STS_SELECTED)
                            {
                                SetColor(pLb->hdr.pGolScheme->Color1);
                            }
                    }

                    if
                    (
                        !Bar
                            (
                                pLb->hdr.left + GOL_EMBOSS_SIZE + LB_INDENT,
                                GetY(),
                                pLb->hdr.right - GOL_EMBOSS_SIZE - LB_INDENT,
                                GetY() + pLb->textHeight
                            )
                    ) return (0);

                    if(!GetState(pLb, LB_CENTER_ALIGN | LB_RIGHT_ALIGN))
                    {
                        MoveTo(pLb->hdr.left + GOL_EMBOSS_SIZE + LB_INDENT, GetY());
                    }
                    else
                    {
                        temp = GetTextWidth(pCurItem->pText, pLb->hdr.pGolScheme->pFont);
                        if(pCurItem->pBitmap != NULL)
                        {
                            temp += GetImageWidth(pCurItem->pBitmap) + LB_INDENT;
                        }

                        if(GetState(pLb, LB_RIGHT_ALIGN))
                        {
                            MoveTo(pLb->hdr.right - temp - LB_INDENT - GOL_EMBOSS_SIZE,
GetY());
                        }
                        else
                        {
                            MoveTo((pLb->hdr.left + pLb->hdr.right - temp) >> 1, GetY());
                        }
                    }

                    if(GetState(pLb, LB_DISABLED))
                    {
```

```
                        SetColor(pLb->hdr.pGolScheme->TextColorDisabled);
                    }
                    else
                    {
                        if(pCurItem->status & LB_STS_SELECTED)
                        {
                            SetColor(pLb->hdr.pGolScheme->TextColor1);
                        }
                        else
                        {
                            SetColor(pLb->hdr.pGolScheme->TextColor0);
                        }
                    }

                    state = LB_STATE_BITMAP;

                case LB_STATE_BITMAP:
                    if(pCurItem->pBitmap != NULL)
                    {
                        // check if the image will go beyond the list box (bottom check
only)
                        if ((GetY() + GetImageHeight(pCurItem->pBitmap)) < pLb->hdr.bottom)
                        {
                            if
                            (
                                !PutImage
                                    (
                                        GetX(),
                                        GetY() + ((pLb->textHeight -
GetImageHeight(pCurItem->pBitmap)) >> 1),
                                        pCurItem->pBitmap,
                                        1
                                    )
                            ) return (0);
                        }

                        MoveRel(GetImageWidth(pCurItem->pBitmap) + LB_INDENT, 0);
                    }

                    state = LB_STATE_TEXT;

                case LB_STATE_TEXT:
                    if(!OutText(pCurItem->pText))
                        return (0);
                }

            ////////////////////////////////////////////////////////////////////////
    L_LB_NEXTITEM:
            state = LB_STATE_ITEMFOCUS;

        case LB_STATE_ITEMFOCUS:
            if(pLb->pFocusItem == pCurItem)
            {
                if(IsDeviceBusy())
                    return (0);

                if(GetState(pLb, LB_FOCUSED))
                {
                    SetColor(pLb->hdr.pGolScheme->TextColor1);
                }
                else
                {
                    SetColor(pLb->hdr.pGolScheme->TextColor0);
                }

                SetLineType(FOCUS_LINE);
                temp = GetY();
                if
                (
                    !Rectangle
                        (
                            pLb->hdr.left + GOL_EMBOSS_SIZE + LB_INDENT,
```

```
                                GetY(),
                                pLb->hdr.right - GOL_EMBOSS_SIZE - LB_INDENT,
                                GetY() + pLb->textHeight - 1
                            )
                    ) return (0);
                    MoveTo(0, temp);
                    SetLineType(SOLID_LINE);

                    // Scroll if needed
                    if(GetY() < (pLb->hdr.top + GOL_EMBOSS_SIZE + LB_INDENT))
                    {
                        pLb->scrollY += (pLb->hdr.top + GOL_EMBOSS_SIZE + LB_INDENT) - GetY();
                        SetState(pLb, LB_DRAW);
                        goto L_LB_DRAW;
                    }

                    if((GetY() + pLb->textHeight) > (pLb->hdr.bottom - GOL_EMBOSS_SIZE -
LB_INDENT))
                    {
                        pLb->scrollY += pLb->hdr.bottom - GetY() - pLb->textHeight -
GOL_EMBOSS_SIZE - LB_INDENT;
                        SetState(pLb, LB_DRAW);
                        goto L_LB_DRAW;
                    }
                }

                pCurItem = (LISTITEM *)pCurItem->pNextItem;
                MoveRel(0, pLb->textHeight);

                goto L_LB_DRAWITEM;
        }   // end of switch

        return (1);
}

#endif // USE_LISTBOX
```

14.1.23 ListBox.h

Functions

	Name	Description
	LbAddItem (see page 184)	This function allocates memory for the LISTITEM (see page 196) and adds it to the list box. The newly created LISTITEM (see page 196) will store the location of pText, pBitmap and other parameters describing the added item.
	LbChangeSel (see page 186)	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.
	LbCreate (see page 181)	This function creates a LISTBOX (see page 195) 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.
	LbDelItem (see page 186)	This function removes an item from the list box and frees the memory used.
	LbDelItemsList (see page 193)	This function removes all items from the list box and frees the memory used.
	LbDraw (see page 183)	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.

	LbGetFocusedItem (see page 188)	This function returns the index of the focused item in the list box.
	LbGetSel (see page 188)	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.
	LbMsgDefault (see page 193)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	LbSetFocusedItem (see page 189)	This function sets the focus for the item with the given index.
	LbTranslateMsg (see page 194)	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
LB_CENTER_ALIGN (see page 179)	Bit to indicate text is center aligned
LB_DISABLED (see page 180)	Bit for disabled state
LB_DRAW (see page 180)	Bit to indicate whole edit box must be redrawn
LB_DRAW_FOCUS (see page 180)	Bit to indicate whole edit box must be redrawn
LB_DRAW_ITEMS (see page 180)	Bit to indicate whole edit box must be redrawn
LB_FOCUSED (see page 180)	Bit for focused state
LB_HIDE (see page 181)	Bit to remove object from screen
LB_RIGHT_ALIGN (see page 179)	Bit to indicate text is left aligned
LB_SINGLE_SEL (see page 179)	Bit to indicate the only item can be selected
LB_STS_REDRAW (see page 181)	Item is to be redrawn.
LB_STS_SELECTED (see page 181)	Item is selected.
LbClrtSel (see page 191)	This macro clears the selection of an item.
LbGetBitmap (see page 192)	This macro returns the location of the currently used bitmap for the item.
LbGetCount (see page 189)	This macro returns the number of items in the list box.
LbGetItemList (see page 184)	This function returns the pointer to the current item list used in the list box.
LbGetVisibleCount (see page 190)	This macro returns the number of items visible in the list box window.
LbSetBitmap (see page 191)	This macro sets the bitmap used in the item.
LbSetSel (see page 187)	This macro sets the selection status of an item to selected.

Structures

Name	Description
LISTBOX (see page 195)	Defines the parameters required for a list box Object.
LISTITEM (see page 196)	Defines the parameters required for a list item used in list box.

Description

This is file ListBox.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  List box
 *****************************************************************************
 * FileName:        ListBox.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 ***************************************************************************/
#ifndef _LISTBOX_H
    #define _LISTBOX_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define LB_FOCUSED      0x0001  // Bit for focused state
    #define LB_DISABLED     0x0002  // Bit for disabled state
    #define LB_RIGHT_ALIGN  0x0004  // Bit to indicate text is left aligned
    #define LB_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned
    #define LB_SINGLE_SEL   0x0010  // Bit to indicate the only item can be selected
    #define LB_DRAW_ITEMS   0x1000  // Bit to indicate whole edit box must be redrawn
    #define LB_DRAW_FOCUS   0x2000  // Bit to indicate whole edit box must be redrawn
    #define LB_DRAW         0x4000  // Bit to indicate whole edit box must be redrawn
    #define LB_HIDE         0x8000  // Bit to remove object from screen
    #define LB_INDENT       0x02    // Indentation constant for the text from the frame

/*********************************************************************
* Overview: Defines the parameters required for a list item used in
*           list box.
*
*********************************************************************/
typedef struct
{
    void    *pPrevItem;             // Pointer to the next item
    void    *pNextItem;             // Pointer to the next item
    WORD    status;                 // Specifies the status of the item.

    // The following values are defined for
    // the status: LB_STS_SELECTED, LB_STS_REDRAW.
    XCHAR   *pText;                 // Pointer to the text for the item
    void    *pBitmap;               // Pointer to the bitmap
    WORD    data;                   // Some data associated with the item
} LISTITEM;

/*********************************************************************
* Bit definitions for the status of an item
*********************************************************************/
    #define LB_STS_SELECTED 0x0001  // Item is selected.
    #define LB_STS_REDRAW   0x0002  // Item is to be redrawn.

/*********************************************************************
```

```
* Overview: Defines the parameters required for a list box Object.
*
*********************************************************************/
typedef struct
{
    OBJ_HEADER  hdr;            // Generic header for all Objects (see OBJ_HEADER).
    LISTITEM    *pItemList;     // 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.
} LISTBOX;

/*********************************************************************
* Macros:  LbSetBitmap(pItem, pBtmap)
*
* Overview: This macro sets the bitmap used in the item.
*
* PreCondition: none
*
* Input: pItem - Pointer to the item.
*        pBtmap - Pointer to the bitmap to be used.
*
* Output: none
*
* Example:
*   See LbAddItem() example.
*
* Side Effects: none
*
*********************************************************************/
    #define LbSetBitmap(pItem, pBtmap)  ((LISTITEM *)pItem)->pBitmap = pBtmap

/*********************************************************************
* Macros:  LbGetBitmap(pItem)
*
* Overview: This macro returns the location of the currently
*           used bitmap for the item.
*
* PreCondition: none
*
* Input: pItem - Pointer to the list item.
*
* Output: Returns the pointer to the current bitmap used.
*
* Example:
*   <CODE>
*   // Assume pLb is initialized to an existing list box
*   LISTITEM *pItem;
*   void *pBitmap;
*
*   pItem = LbGetItemList(pLb);
*   pBitmap = LbGetBitmap(pItem);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define LbGetBitmap(pItem)  ((LISTITEM *)pItem)->pBitmap

/*********************************************************************
* Function: LISTBOX  *LbCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                              SHORT bottom, WORD state, XCHAR* pText,
*                              GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
```

```
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object.
*        bottom - Bottom most position of the Object.
*        pText - Pointer to the initialization text for the items.
*        pScheme - Pointer to the style scheme.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   #define LISTBOX_ID   10
*
*   const XCHAR ItemList[] = "Line1\n" "Line2\n";
*
*   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++;
*       }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
LISTBOX * LbCreate
    (WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, XCHAR * pText,
GOL_SCHEME * pScheme);

/*********************************************************************
* Function: LISTITEM* LbAddItem(LISTBOX *pLb, LISTITEM *pPrevItem,
*                               XCHAR *pText, void* pBitmap, WORD status, WORD data)
*
* 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.
*
* PreCondition: none
*
* Input: pLb - The pointer to the list box object.
*        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.
```

```
*        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.
*        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.
*        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.
*        data -  User assigned data associated with the item.
*
* Output: Return a pointer to the item created,
*         NULL if the operation was not successful.
*
* Example:
*   <CODE>
*   const XCHAR ItemList[] = "Line1\n" "Line2\n" "Line3\n";
*
*   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);
*
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
LISTITEM    *LbAddItem(LISTBOX *pLb, LISTITEM *pPrevItem, XCHAR *pText, void *pBitmap, WORD
status, WORD data);

/*********************************************************************
* Function: void LbDelItem(LISTBOX *pLb, LISTITEM *pItem)
*
* Overview: This function removes an item from the list box
*           and frees the memory used.
*
```

```
* PreCondition: none
*
* Input: pLb - The pointer to the list box object.
*        pItem -  The pointer to the item that will be removed.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void        LbDelItem(LISTBOX *pLb, LISTITEM *pItem);

/*********************************************************************
* Function: void LbDelItemsList(void *pObj)
*
* Overview: This function removes all items from the list box
*           and frees the memory used.
*
* PreCondition: none
*
* Input: pLb - The pointer to the list box object.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void        LbDelItemsList(void *pObj);

/*********************************************************************
* Function: LISTITEM* LbGetSel(LISTBOX *pLb, LISTITEM *pFromItem)
*
* 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.
*
* PreCondition: none
*
* Input: pLb        - The pointer to the list box object.
*        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.
*
* Output: pointer to the selected item, NULL if there are no items selected
*
*
* Side Effects: none
*
*********************************************************************/
LISTITEM    *LbGetSel(LISTBOX *pLb, LISTITEM *pFromItem);

/*********************************************************************
* Function: void LbChangeSel(LISTBOX *pLb, LISTITEM *pItem)
*
* 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.
*
* PreCondition: none
*
* Input: pLb    - The pointer to the list box object.
*        pItem  - The pointer to the item the selection status
*                     will be changed.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
```

```
void        LbChangeSel(LISTBOX *pLb, LISTITEM *pItem);

/*********************************************************************
* Macro: LbSetSel(pLb, pItem)
*
* Overview: This macro sets the selection status of an item to
*           selected.
*
* PreCondition: none
*
* Input: pLb    - The pointer to the list box object.
*        pItem  - The pointer to the item the selection status
*                 will be set.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define LbSetSel(pLb, pItem)           \
    if(!(pItem->status & LB_STS_SELECTED)) \
        LbChangeSel((LISTBOX *)pLb, pItem);

/*********************************************************************
* Macro: LbClrSel(pLb, pItem)
*
* Overview: This macro clears the selection of an item.
*
* PreCondition: none
*
* Input: pLb - The pointer to the list box.
*        pItem - The pointer to the item the selection status should be cleared.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define LbClrtSel(pLb, pItem)       \
    if(pItem->status & LB_STS_SELECTED) \
        LbChangeSel((LISTBOX *)pLb, pItem);

/*********************************************************************
* Macro: LbGetCount(pLb)
*
* Overview: This macro returns the number of items in the list box.
*
* PreCondition: none
*
* Input: pLb - The pointer to the list box object.
*
* Output: The number of items the list box contains.
*
* Side Effects: none
*
*********************************************************************/
    #define LbGetCount(pLb) ((LISTBOX *)pLb)->itemsNumber

/*********************************************************************
* Macro: LbGetVisibleCount(pLb)
*
* Overview: This macro returns the number of items visible in the
*           list box window.
*
* PreCondition: none
*
* Input: pLb - The pointer to the list box object.
*
* Output: The number of items visible in the list box window.
*
* Side Effects: none
*
```

```
*****************************************************************/
    #define
LbGetVisibleCount(pLb)

\
        (

\
            (((LISTBOX *)pLb)->hdr.bottom - ((LISTBOX *)pLb)->hdr.top - 2 *
(GOL_EMBOSS_SIZE + LB_INDENT)) / \
                ((LISTBOX
*)pLb)->textHeight                                                      \
        )

/*****************************************************************
* Function: void LbSetFocusedItem(LISTBOX* pLb, SHORT index)
*
* Overview: This function sets the focus for the item with the
*           given index.
*
* PreCondition: none
*
* Input: pLb   - The pointer to the list box object.
*        index - The index number of the item to be focused.
*                First item on the list is always indexed 0.
*
* Output: none.
*
* Side Effects: none
*
*****************************************************************/
void    LbSetFocusedItem(LISTBOX *pLb, SHORT index);

/*****************************************************************
* Function: SHORT LbGetFocusedItem(LISTBOX* pLb)
*
* Overview: This function returns the index of the focused item
*           in the list box.
*
* PreCondition: none
*
* Input: pLb   - The pointer to the list box object.
*
* Output: Returns the index of the focused item in the list box.
*
* Side Effects: none
*
*****************************************************************/
SHORT   LbGetFocusedItem(LISTBOX *pLb);

/*****************************************************************
* Macro: LISTITEM LbGetItemList(LISTBOX* pLb)
*
* Overview: This function returns the pointer to the current
*           item list used in the list box.
*
* PreCondition: none
*
* Input: pLb   - The pointer to the list box object.
*
* Output: Returns the pointer to the LISTITEM used in the list box.
*
* Example:
*   See LbAddItem() example.
*
* Side Effects: none
*
*****************************************************************/
    #define LbGetItemList(pLb)  ((LISTITEM *)pLb->pItemList)

/*****************************************************************
* Function: WORD LbTranslateMsg(void *pObj, GOL_MSG *pMsg)
```

```
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pLB   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
WORD    LbTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: void LbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       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 redrawn
with selected item(s).
*       LB_MSG_MOVE          KeyBoard      Set LB_DRAW_ITEMS         List Box will redrawn
with focus on one item.
*       LB_MSG_SEL           KeyBoard      Set LB_DRAW_ITEMS         List Box will redrawn
with selection on the current item focused.
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message
*        pB            - The pointer to the object whose state will be modified.
*        pMsg          - The pointer to the GOL message.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void    LbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
```

```
 * Function: WORD LbDraw(void *pObj)
 *
 * 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.
 *
 * PreCondition: Object must be created before this function is called.
 *
 * Input: pLb - Pointer to the object to be rendered.
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 ***************************************************************************/
WORD LbDraw(void *pObj);
#endif // _LISTBOX_H
```

14.1.24 Meter.c

This is file Meter.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Meter
 *****************************************************************************
 * FileName:        Meter.c
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 Version 3.00, MPLAB C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2011 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 07/31/08     Added arc colors options
 * 08/20/08     Added accuracy option for displaying values
 * 04/01/11     Removed sineTable[], use the one in the Primitive.c instead.
 ***************************************************************************/
#include "Graphics/Graphics.h"
#include <math.h>
#include <stdio.h>

#ifdef USE_METER

/* Internal Functions */
void    MtrCalcDimensions(METER *pMtr); // used to calculate the meter dimensions

// which is dependent on meter type

/*******************************************************************
* Function: METER  *MtrCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                             SHORT bottom, WORD state, SHORT value,
*                             SHORT minValue, SHORT maxValue, XCHAR *pText,
*                             GOL_SCHEME *pScheme)
*
*
* Notes: Creates a METER object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*******************************************************************/
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
)
{
    METER   *pMtr = NULL;

    pMtr = (METER *)GFX_malloc(sizeof(METER));
    if(pMtr == NULL)
        return (NULL);

    pMtr->hdr.ID = ID;          // unique id assigned for referencing
    pMtr->hdr.pNxtObj = NULL;   // initialize pointer to NULL
    pMtr->hdr.type = OBJ_METER; // set object type
    pMtr->hdr.left = left;      // left,top coordinate
    pMtr->hdr.top = top;        //
    pMtr->hdr.right = right;    // right,bottom coordinate
    pMtr->hdr.bottom = bottom;  //
    pMtr->minValue = minValue;
    pMtr->maxValue = maxValue;
    pMtr->value = value;
    pMtr->hdr.state = state;    // state
    pMtr->pText = pText;
    pMtr->hdr.DrawObj = MtrDraw;                // draw function
    pMtr->hdr.MsgObj = MtrTranslateMsg;         // message function
    pMtr->hdr.MsgDefaultObj = MtrMsgDefault;    // default message function
```

```
    pMtr->hdr.FreeObj = NULL;                   // free function

    // set the default scale colors
    MtrSetScaleColors(pMtr, LIGHTGREEN, YELLOW, BRIGHTGREEN, BRIGHTBLUE, RED, BRIGHTRED);

    // Set the color scheme to be used
    if(pScheme == NULL)
        pMtr->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pMtr->hdr.pGolScheme = pScheme;

    // Set the Title Font to be used
    if(pTitleFont == NULL)
        pMtr->pTitleFont = (void *) &FONTDEFAULT;
    else
        pMtr->pTitleFont = pTitleFont;

    // Set the Value Font to be used
    if(pValueFont == NULL)
        pMtr->pValueFont = (void *) &FONTDEFAULT;
    else
        pMtr->pValueFont = pValueFont;

    // calculate dimensions of the meter
    MtrCalcDimensions(pMtr);

    GOLAddObject((OBJ_HEADER *)pMtr);

    return (pMtr);
}

/*********************************************************************
* Function: MtrCalcDimensions(void)
*
* Notes: Calculates the dimension of the meter. Dependent on the
*        meter type set.
*
*********************************************************************/
void MtrCalcDimensions(METER *pMtr)
{
    SHORT   tempHeight, tempWidth;
    SHORT   left, top, right, bottom;
    XCHAR   tempChar[2] = {'8',0};

    left = pMtr->hdr.left;
    right = pMtr->hdr.right;
    top = pMtr->hdr.top;
    bottom = pMtr->hdr.bottom;

    // get the text width reference. This is used to scale the meter
    if(pMtr->pText != NULL)
    {
        tempHeight = (GOL_EMBOSS_SIZE << 1) + GetTextHeight(pMtr->hdr.pGolScheme->pFont);
    }
    else
    {
        tempHeight = (GOL_EMBOSS_SIZE << 1);
    }

    tempWidth = (GOL_EMBOSS_SIZE << 1) + (GetTextWidth(tempChar,
pMtr->hdr.pGolScheme->pFont) * SCALECHARCOUNT);

    // Meter size is dependent on the width or height.
    // The radius is also adjusted to add space for the scales
        #if (METER_TYPE == MTR_WHOLE_TYPE)

    // choose the radius
    if((right - left - tempWidth) > (bottom - top - tempHeight -
GetTextHeight(pMtr->pTitleFont)))
    {
        pMtr->radius = ((bottom - top - tempHeight - GetTextHeight(pMtr->pTitleFont)) >> 1)
- ((tempHeight + bottom - top) >> 3);
```

```
    }
    else
        pMtr->radius = ((right - left) >> 1) - (tempWidth + ((right - left) >> 3));

    // center the meter on the given dimensions
    pMtr->xCenter = (left + right) >> 1;
    pMtr->yCenter = ((bottom + top) >> 1) - (tempHeight >> 1);

        #elif (METER_TYPE == MTR_HALF_TYPE)

    // choose the radius
    if((right - left) >> 1 > (bottom - top))
    {
        pMtr->radius = (bottom - top) - ((tempHeight << 1) + ((bottom - top) >> 3));
        pMtr->yCenter = ((bottom + top) >> 1) + ((pMtr->radius + ((bottom - top) >> 3)) >>
1);
    }
    else
    {
        pMtr->radius = ((right - left) >> 1) - (tempWidth + ((right - left) >> 3));
        pMtr->yCenter = ((bottom + top) >> 1) + ((pMtr->radius + ((right - left) >> 3)) >>
1);
    }

    // center the meter on the given dimensions
    pMtr->xCenter = (left + right) >> 1;

        #elif (METER_TYPE == MTR_QUARTER_TYPE)

    // choose the radius
    if
    (
        (right - left - tempWidth) >
            (bottom - top - (GetTextHeight(pMtr->pTitleFont) +
GetTextHeight(pMtr->hdr.pGolScheme->pFont))) -
                (GOL_EMBOSS_SIZE << 1)
    )
    {
        pMtr->radius = bottom - top - (GetTextHeight(pMtr->pTitleFont) +
GetTextHeight(pMtr->hdr.pGolScheme->pFont) + (GOL_EMBOSS_SIZE << 1));
    }
    else
    {
        pMtr->radius = right -
            left -
            (GetTextWidth(tempChar, pMtr->hdr.pGolScheme->pFont) * (SCALECHARCOUNT + 1)) -
            GOL_EMBOSS_SIZE;
    }

    pMtr->radius -= (((pMtr->radius) >> 2) + GOL_EMBOSS_SIZE);

    // center the meter on the given dimensions
    pMtr->xCenter = ((left + right) >> 1) - ((pMtr->radius + tempWidth + (pMtr->radius >>
2)) >> 1);
    pMtr->yCenter = ((top + bottom) >> 1) + ((pMtr->radius + (pMtr->radius >> 2)) >> 1);
        #endif
}

/*****************************************************************
* Function: MtrSetVal(METER *pMtr, SHORT newVal)
*
* Notes: Sets the value of the meter to newVal. If newVal is less
*        than 0, 0 is assigned. If newVal is greater than range,
*        range is assigned.
*
*****************************************************************/
void MtrSetVal(METER *pMtr, SHORT newVal)
{
    if((newVal < 0) || (newVal < pMtr->minValue))
    {
        pMtr->value = pMtr->minValue;
        return;
```

```
    }

    if(newVal > pMtr->maxValue)
    {
        pMtr->value = pMtr->maxValue;
        return;
    }

    pMtr->value = newVal;
}

/*********************************************************************
* Function: MtrMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Notes: This the default operation to change the state of the meter.
*        Called inside GOLMsg() when GOLMsgCallback() returns a 1.
*
*********************************************************************/
void MtrMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    METER *pMtr;

    pMtr = (METER *)pObj;

    if(translatedMsg == MTR_MSG_SET)
    {
        MtrSetVal(pMtr, pMsg->param2);      // set the value
        SetState(pMtr, MTR_DRAW_UPDATE);    // update the meter
    }
}

/*********************************************************************
* Function: WORD MtrTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*********************************************************************/
WORD MtrTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    METER *pMtr;

    pMtr = (METER *)pObj;

    // Evaluate if the message is for the meter
    // Check if disabled first
    if(GetState(pMtr, MTR_DISABLED))
        return (OBJ_MSG_INVALID);

    if(pMsg->type == TYPE_SYSTEM)
    {
        if(pMsg->param1 == pMtr->hdr.ID)
        {
            if(pMsg->uiEvent == EVENT_SET)
            {
                return (MTR_MSG_SET);
            }
        }
    }

    return (OBJ_MSG_INVALID);
}

/*********************************************************************
* Function: WORD MtrDraw(void *pObj)
*
* Notes: This is the state machine to draw the meter.
*
*********************************************************************/
WORD MtrDraw(void *pObj)
{
    typedef enum
```

```
    {
        IDLE,
        FRAME_DRAW,
        NEEDLE_DRAW,
        NEEDLE_ERASE,
        TEXT_DRAW,
        TEXT_DRAW_RUN,
        ARC_DRAW_SETUP,
        ARC_DRAW,
        SCALE_COMPUTE,
        SCALE_LABEL_DRAW,
        SCALE_DRAW,
        VALUE_ERASE,
        VALUE_DRAW,
        VALUE_DRAW_RUN,
    } MTR_DRAW_STATES;

    static MTR_DRAW_STATES state = IDLE;
    static SHORT x1, y1, x2, y2;
    static SHORT temp, j, i, angle;
    static XCHAR strVal[SCALECHARCOUNT + 1];    // add one more space here for the NULL
character
    static XCHAR tempXchar[2] = {'8',0};        // NULL is pre-defined here
    static float radian;
    static DWORD_VAL dTemp, dRes;
    METER *pMtr;

    pMtr = (METER *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case IDLE:
            if(GetState(pMtr, MTR_HIDE))
            {   // Hide the meter (remove from screen)
                SetColor(pMtr->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pMtr->hdr.left, pMtr->hdr.top, pMtr->hdr.right, pMtr->hdr.bottom))
                    return (0);
                return (1);
            }

            // Check if we need to draw the whole object
            SetLineThickness(NORMAL_LINE);
            SetLineType(SOLID_LINE);
            if(GetState(pMtr, MTR_DRAW))
            {

                // set parameters to draw the frame
                GOLPanelDraw
                    (
                        pMtr->hdr.left, pMtr->hdr.top, pMtr->hdr.right, pMtr->hdr.bottom,
0, pMtr->hdr.pGolScheme->Color0,
                            pMtr->hdr.pGolScheme->EmbossLtColor,
pMtr->hdr.pGolScheme->EmbossDkColor, NULL,
                                GOL_EMBOSS_SIZE - 1
                    );
                state = FRAME_DRAW;
            }
            else
            {
                state = NEEDLE_ERASE;
                goto mtr_needle_draw_st;
            }

        case FRAME_DRAW:
            if(!GOLPanelDrawTsk())
            {
                return (0);
            }
```

```
            state = TEXT_DRAW;

        case TEXT_DRAW:

            // draw the meter title
            SetColor(pMtr->hdr.pGolScheme->TextColor1);
            SetFont(pMtr->pTitleFont);
            temp = GetTextWidth(pMtr->pText, pMtr->pTitleFont);

            // set the start location of the meter title
                #if (METER_TYPE == MTR_WHOLE_TYPE)
                    #ifdef METER_DISPLAY_VALUES_ENABLE
            MoveTo(pMtr->xCenter - (temp >> 1), pMtr->yCenter + pMtr->radius +
GetTextHeight(pMtr->pValueFont));
                    #else
            MoveTo(pMtr->xCenter - (temp >> 1), pMtr->yCenter + pMtr->radius +
GetTextHeight(pMtr->hdr.pGolScheme->pFont));
                    #endif
                #elif (METER_TYPE == MTR_HALF_TYPE)
            MoveTo(pMtr->xCenter - (temp >> 1), pMtr->yCenter + 3);

                #elif (METER_TYPE == MTR_QUARTER_TYPE)
            MoveTo
                (
                    ((pMtr->hdr.right + pMtr->hdr.left) >> 1) - (temp >> 1),
pMtr->hdr.bottom - GOL_EMBOSS_SIZE - GetTextHeight
                        (pMtr->pTitleFont)
                );
                #endif
            state = TEXT_DRAW_RUN;

        case TEXT_DRAW_RUN:

            // render the title of the meter
            if(!OutText(pMtr->pText))
                return (0);
            state = ARC_DRAW_SETUP;

        //case ARC0_DRAW:
        case ARC_DRAW_SETUP:

            // check if we need to draw the arcs
            if(!GetState(pMtr, MTR_RING))
            {

                // if meter is not RING type, for scale label colors use
                // the three colors (normal, critical and danger)
                i = DEGREE_END;
                state = SCALE_COMPUTE;
                goto mtr_scale_compute;
            }
            else
            {

                // set the arc radii: x1 smaller radius and x2 as the larger radius
                x1 = pMtr->radius + 2;
                x2 = pMtr->radius + (pMtr->radius >> 2) + 2;
                    #if (METER_TYPE == MTR_WHOLE_TYPE)
                temp = 6;
                    #elif (METER_TYPE == MTR_HALF_TYPE)
                temp = 5;
                    #elif (METER_TYPE == MTR_QUARTER_TYPE)
                temp = 3;
                    #endif
                state = ARC_DRAW;
            }

        case ARC_DRAW:

            // draw the arcs
                #if (METER_TYPE == MTR_WHOLE_TYPE)
            while(temp)
```

```
            {
                    #elif (METER_TYPE == MTR_HALF_TYPE)
                while(temp > 1)
                {
                        #elif (METER_TYPE == MTR_QUARTER_TYPE)
                    while(temp > 1)
                    {
                            #endif

                        // decide which arc will be drawn
                        switch(temp)
                        {
                            case 6:
                                SetColor(pMtr->arcColor1);
                                if(!Arc(pMtr->xCenter, pMtr->yCenter, pMtr->xCenter,
pMtr->yCenter, x1, x2, 0x20))
                                    return (0);
                                break;

                            case 5:
                                SetColor(pMtr->arcColor2);
                                if(!Arc(pMtr->xCenter, pMtr->yCenter, pMtr->xCenter,
pMtr->yCenter, x1, x2, 0x40))
                                    return (0);
                                break;

                            case 4:
                                SetColor(pMtr->arcColor3);
                                if(!Arc(pMtr->xCenter, pMtr->yCenter, pMtr->xCenter,
pMtr->yCenter, x1, x2, 0x80))
                                    return (0);
                                break;

                            case 3:
                                SetColor(pMtr->arcColor4);
                                if(!Arc(pMtr->xCenter, pMtr->yCenter, pMtr->xCenter,
pMtr->yCenter, x1, x2, 0x01))
                                    return (0);
                                break;

                            case 2:
                                SetColor(pMtr->arcColor5);
                                if(!Arc(pMtr->xCenter, pMtr->yCenter, pMtr->xCenter,
pMtr->yCenter, x1, x2, 0x02))
                                    return (0);
                                break;

                            case 1:
                                SetColor(pMtr->arcColor6);
                                if(!Arc(pMtr->xCenter, pMtr->yCenter, pMtr->xCenter,
pMtr->yCenter, x1, x2, 0x04))
                                    return (0);
                                break;

                            default:
                                break;
                        }

                        temp--;
                    }

                    // set the color for the scale labels
                    SetColor(pMtr->hdr.pGolScheme->Color1);
                    i = DEGREE_END;
                    state = SCALE_COMPUTE;

                case SCALE_COMPUTE:
                    mtr_scale_compute : if(i >= DEGREE_START)
                    {
                        radian = i * .0175;

                        if(!GetState(pMtr, MTR_RING))
```

```
                        {
                            if(i >= ARC1_DEGREE)
                            {
                                SetColor(pMtr->arcColor1);
                            }
                            else if(i >= ARC2_DEGREE)
                            {
                                SetColor(pMtr->arcColor2);
                            }
                            else if(i >= ARC3_DEGREE)
                            {
                                SetColor(pMtr->arcColor3);
                            }
                            else if(i >= ARC4_DEGREE)
                            {
                                SetColor(pMtr->arcColor4);
                            }
                            else if(i >= ARC5_DEGREE)
                            {
                                SetColor(pMtr->arcColor5);
                            }
                            else
                            {
                                SetColor(pMtr->arcColor6);
                            }
                        }

                        // compute for the effective radius of the scales
                        if((i % 45) == 0)
                            x2 = pMtr->radius + (pMtr->radius >> 2) + 2;
                        else
                            x2 = pMtr->radius + (pMtr->radius >> 3) + 3;

                        // compute the starting x1 and y1 position of the scales
                        // x2 here is the distance from the center to the x1, y1
                        // position. Sin and cos is used here since computation speed in
initial
                        // drawing is not yet critical.
                        x1 = x2 * cos(radian);
                        y1 = (-1) * (x2 * sin(radian));

                        // using ratio and proportion we get the x2,y2 position
                        dTemp.Val = 0;
                        dTemp.w[1] = (pMtr->radius + 3);
                        dTemp.Val /= x2;

                        dRes.Val = dTemp.Val * x1;
                        x2 = dRes.w[1] + pMtr->xCenter; // adjusted to center
                        dRes.Val = dTemp.Val * y1;
                        y2 = dRes.w[1] + pMtr->yCenter; // adjusted to center
                        x1 += pMtr->xCenter;
                        y1 += pMtr->yCenter;            // adjust x1, y1 to the center
                        state = SCALE_DRAW;
                    }
                    else
                    {
                        state = NEEDLE_ERASE;
                        goto mtr_needle_draw_st;
                    }

                case SCALE_DRAW:
                    if(!Line(x1, y1, x2, y2))               // now draw the scales
                        return (0);

                    if((i % 45) == 0)
                    {

                        // draw the scale labels
                        // reusing radian, x2 and y2
                        // compute for the actual angle of needle to be shown in
screen
                        radian = (DEGREE_END - DEGREE_START) - (i - (DEGREE_START));
```

```
                        // compute the values of the label to be shown per 45 degree
                        temp = (pMtr->maxValue - pMtr->minValue) * (radian / (DEGREE_END -
DEGREE_START));

                        // adjust for the minimum or offset value
                        temp += pMtr->minValue;

                        // this implements sprintf(strVal, "%d", temp); faster
                        // note that this is just for values >= 0, while sprintf covers
negative values.
                        j = 1;

                        // get the ones value first and account for the required decimal
point if enabled
                        if(GetState(pMtr, MTR_ACCURACY))
                        {

                            // round off to nearest tens
                            dTemp.w[0] = (temp % RESOLUTION) / (RESOLUTION / 10);
                            if((dTemp.w[0]) > (RESOLUTION >> 1))
                                temp += (RESOLUTION - dTemp.w[0]);
                            temp /= RESOLUTION;
                        }

                        do
                        {
                            strVal[SCALECHARCOUNT - j] = (temp % 10) + '0';
                            if(((temp /= 10) == 0) || (j >= SCALECHARCOUNT))
                                break;
                            j++;
                        } while(j <= SCALECHARCOUNT);

                        // the (&strVal[SCALECHARCOUNT-j]) removes the leading zeros.
                        // if leading zeroes will be printed change
(&strVal[SCALECHARCOUNT-j])
                        // to simply strVal and remove the break statement above in the
do-while loop
                        x2 = GetTextWidth((&strVal[SCALECHARCOUNT - j]),
pMtr->hdr.pGolScheme->pFont);
                        y2 = GetTextHeight(pMtr->hdr.pGolScheme->pFont);

                        if(i == -45)
                        {
                            MoveTo(x1 + 3, y1);
                        }
                        else if(i == 0)
                        {
                            MoveTo(x1 + 3, y1 - (y2 >> 1));
                        }
                        else if(i == 45)
                        {
                            MoveTo(x1 + 3, y1 - (y2 >> 1) - 3);
                        }
                        else if(i == 90)
                        {
                            MoveTo(x1 - (x2 >> 1), y1 - (y2) - 3);
                        }
                        else if(i == 135)
                        {
                            MoveTo(x1 - (x2), y1 - (y2));
                        }
                        else if(i == 180)
                        {
                            MoveTo(x1 - (x2) - 3, y1 - (y2 >> 1));
                        }
                        else if(i == 225)
                        {
                            MoveTo(x1 - (x2 + 3), y1);
                        }

                        state = SCALE_LABEL_DRAW;
```

```
                    }
                    else
                    {
                        i -= DEGREECOUNT;
                        state = SCALE_COMPUTE;          //SCALE_LABEL_DRAW;
                        goto mtr_scale_compute;
                    }

                case SCALE_LABEL_DRAW:
                    SetFont(pMtr->hdr.pGolScheme->pFont);
                    if(!OutText(&strVal[SCALECHARCOUNT - j]))
                        return (0);
                    i -= DEGREECOUNT;
                    state = SCALE_COMPUTE;
                    goto mtr_scale_compute;

                case NEEDLE_ERASE:
                    mtr_needle_draw_st : if(GetState(pMtr, MTR_DRAW_UPDATE))
                    {

                        // to update the needle, redraw the old position with background
color
                        SetColor(pMtr->hdr.pGolScheme->Color0);
                        SetLineThickness(THICK_LINE);
                        if(!Line(pMtr->xCenter, pMtr->yCenter, pMtr->xPos, pMtr->yPos))
                            return (0);
                    }

                    state = NEEDLE_DRAW;

                case NEEDLE_DRAW:
                    if(IsDeviceBusy())
                        return (0);

                    // At this point, pMtr->value is assumed to contain the new value of
the meter.
                    // calculate the new angle:
                    // equation is still ratio and proportion of angle and values.
                    dTemp.Val = 0;
                    dTemp.w[1] = pMtr->value - pMtr->minValue;
                    dTemp.Val /= (pMtr->maxValue - pMtr->minValue);
                    dTemp.Val *= (DEGREE_END - DEGREE_START);

                    angle = DEGREE_END - (dTemp.w[1]);
                    temp = angle;

                    /* The method uses a lookup table of pre-calculated sine values. The
table
                is derived from calculating sine values from 0 degrees to 90 degrees.
                To save space, the entries are just a byte size. So 360/256 = 1.40625 degrees
                increments is used for each entry. (i.e entry 0 is zero, entry i is 1.40625,
                entry 2 is 2*1.40625,... entry n is n*1.40625.
                Get the sine of the entries yields a float value. Since we only use a
                byte for storage we can shift the values by 128 without overflowing the
                storage. Thus we need to shift the computed values by 8 bits. Shifting now
                permits us to work with integers which greatly reduces the execution time.
                With this in mind, the input angles must be translated to 0-90 range with the
                quadrant of the original angle stored to translate back the value from the
                table to the correct quadrant. Also the quadrant number will determine if
                the calculated x and y positions are to be swapped.
                In summary:
                 Swap x and y when calculating points in quadrant 2 and 4
                 Negate x when in quadrant 2 and 3
                 Negate y when in quadrant 1 and 2
            */

                    // translate the angle to 0-90 range
                    while(temp < 0)                     // this is needed for negative
                        temp += 90;                     // for negative values
                    while(temp > 90)
                        temp -= 90;
```

```
                    // determine which quadrant the angle is located
                    // i determines if x and y are swapped (0 no swapping, 1 swapping
needed)
                    // y2 and x2 are the multiplier to negate or not the x and y values
                    if((180 < angle) && (angle <= 270))
                    {               // quadrant 3
                        i = 0;
                        y2 = 1;
                        x2 = -1;
                    }
                    else if((90 < angle) && (angle <= 180))
                    {               // quadrant 2
                        i = 1;
                        y2 = -1;
                        x2 = -1;
                    }
                    else if((0 <= angle) && (angle <= 90))
                    {               // quadrant 1
                        i = 0;
                        y2 = -1;
                        x2 = 1;
                    }
                    else if((-90 < angle) && (angle < 0))
                    {               // quadrant 4
                        i = 1;
                        y2 = 1;
                        x2 = 1;
                    }

                    // using the calculated angle we get the sine and cosine
                    // values from the look up table, then to avoid
                    // division, the inverse is used
                    x1 = Cosine(temp) * pMtr->radius;
                    y1 = Sine(temp) * pMtr->radius;

                    // calculate new positions, check if we need to reverse x and y values
                    pMtr->xPos = ((x2) * (((i == 0) ? x1 : y1) >> 8)) + pMtr->xCenter;
                    pMtr->yPos = ((y2) * (((i == 0) ? y1 : x1) >> 8)) + pMtr->yCenter;

                    // now draw the needle with the new position
                    SetColor(BRIGHTRED);
                    SetLineThickness(THICK_LINE);
                    if(!Line(pMtr->xCenter, pMtr->yCenter, pMtr->xPos, pMtr->yPos))
                        return (0);
                    SetLineThickness(NORMAL_LINE);
                        #ifdef METER_DISPLAY_VALUES_ENABLE
                    state = VALUE_ERASE;
                        #else

                    // reset the line to normal
                    SetLineThickness(NORMAL_LINE);
                    state = IDLE;
                    return (1);
                        #endif
                        #ifdef METER_DISPLAY_VALUES_ENABLE

                case VALUE_ERASE:
                    if(IsDeviceBusy())
                        return (0);

                    // reset the line to normal
                    SetLineThickness(NORMAL_LINE);

                    // display the value
                    // erase previous value first. The temp>>1 accomodates fonts with
characters that has unequal widths
                    temp = GetTextWidth(tempXchar, pMtr->pValueFont);
                    temp = temp * SCALECHARCOUNT + (temp >> 1);

                    //temp = GetTextWidth(tempXchar, pMtr->pValueFont)*SCALECHARCOUNT;
                    SetColor(pMtr->hdr.pGolScheme->Color0);
```

```
                #if (METER_TYPE == MTR_WHOLE_TYPE)
            if
            (
                !Bar
                    (
                        pMtr->xCenter -
                        (temp >> 1), pMtr->yCenter +
                        pMtr->radius, pMtr->xCenter +
                        (temp >> 1), pMtr->yCenter +
                        pMtr->radius +
                        GetTextHeight(pMtr->pValueFont)
                    )
            ) return (0);

                #elif (METER_TYPE == MTR_HALF_TYPE)
            if
            (
                !Bar
                    (
                        pMtr->xCenter -
                        (temp >> 1), pMtr->yCenter -
                        GetTextHeight(pMtr->pValueFont), pMtr->xCenter +
                        (temp >> 1), pMtr->yCenter
                    )
            ) return (0);

                #elif (METER_TYPE == MTR_QUARTER_TYPE)
            if
            (
                !Bar
                    (
                        pMtr->xCenter -
                        1, pMtr->yCenter -
                        GetTextHeight(pMtr->pValueFont), pMtr->xCenter +
                        temp, pMtr->yCenter +
                        1
                    )
            ) return (0);
                #endif
            state = VALUE_DRAW;

        case VALUE_DRAW:
            if(IsDeviceBusy())
                return (0);

            if(angle >= ARC1_DEGREE)
            {
                SetColor(pMtr->arcColor1);
            }
            else if(angle >= ARC2_DEGREE)
            {
                SetColor(pMtr->arcColor2);
            }
            else if(angle >= ARC3_DEGREE)
            {
                SetColor(pMtr->arcColor3);
            }
            else if(angle >= ARC4_DEGREE)
            {
                SetColor(pMtr->arcColor4);
            }
            else if(angle >= ARC5_DEGREE)
            {
                SetColor(pMtr->arcColor5);
            }
            else
            {
                SetColor(pMtr->arcColor6);
            }

            // display the current value
            SetFont(pMtr->pValueFont);
```

```
                    // this implements sprintf(strVal, "%03d", pMtr->value); faster
                    // note that this is just for values >= 0, while sprintf covers
negative values.
                    i = pMtr->value;
                    j = 1;

                    // get the ones value first and account for the required decimal point
if enabled
                    if(GetState(pMtr, MTR_ACCURACY))
                    {
                        strVal[SCALECHARCOUNT - j] = (((i % RESOLUTION)) / (RESOLUTION /
10)) + '0';
                        i /= RESOLUTION;
                        j++;
                        strVal[SCALECHARCOUNT - j] = '.';
                        j++;
                    }

                    do
                    {
                        strVal[SCALECHARCOUNT - j] = (i % 10) + '0';
                        if(((i /= 10) == 0) || (j >= SCALECHARCOUNT))
                            break;
                        j++;
                    } while(j <= SCALECHARCOUNT);

                    temp = GetTextWidth(&strVal[SCALECHARCOUNT - j], pMtr->pValueFont);

                            #if (METER_TYPE == MTR_WHOLE_TYPE)
                    MoveTo(pMtr->xCenter - (temp >> 1), pMtr->yCenter + pMtr->radius);

                            #elif (METER_TYPE == MTR_HALF_TYPE)
                    MoveTo(pMtr->xCenter - (temp >> 1), pMtr->yCenter -
GetTextHeight(pMtr->pValueFont));

                            #elif (METER_TYPE == MTR_QUARTER_TYPE)
                    MoveTo(pMtr->xCenter, pMtr->yCenter - GetTextHeight(pMtr->pValueFont));
                            #endif
                    state = VALUE_DRAW_RUN;

                case VALUE_DRAW_RUN:
                    if(!OutText(&strVal[SCALECHARCOUNT - j]))
                        return (0);
                    state = IDLE;
                    return (1);
                        #endif //METER_DISPLAY_VALUES_ENABLE
                }

                return (1);
            }

            #endif // USE_METER
```

14.1.25 Meter.h

Functions

	Name	Description
	MtrCreate (see page 200)	This function creates a METER (see page 209) 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 (see page 201)	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... more (see page 201)
	MtrMsgDefault (see page 208)	This function performs the actual state change based on the translated message given. Meter (see page 196) value is set based on parameter 2 of the message given. The following state changes are supported:
	MtrSetVal (see page 202)	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.
	MtrTranslateMsg (see page 208)	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
METER_DISPLAY_VALUES_ENABLE (see page 207)	This enables the display of the values.
METER_TYPE (see page 206)	This is a compile time setting to select the type if meter shape. There are three types: • MTR_WHOLE_TYPE - Meter (see page 196) drawn with 6 octants used. • MTR_HALF_TYPE - Meter (see page 196) drawn with semi circle shape. • MTR_QUARTER_TYPE - Meter (see page 196) 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... more (see page 206)
MTR_ACCURACY (see page 207)	Sets the meter accuracy to one decimal places
MTR_DISABLED (see page 198)	Bit for disabled state.
MTR_DRAW (see page 199)	Bit to indicate object must be redrawn.
MTR_DRAW_UPDATE (see page 199)	Bit to indicate an update only.
MTR_HIDE (see page 199)	Bit to indicate object must be removed from screen.
MTR_RING (see page 199)	Bit for ring type, scales are drawn over the ring
MtrDecVal (see page 203)	This macro is used to directly decrement the value.
MtrGetVal (see page 203)	This macro returns the current value of the meter. Value is always in the minValue-maxValue range inclusive.
MtrIncVal (see page 204)	This macro is used to directly increment the value.
MtrSetScaleColors (see page 204)	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 (see page 356) 6 225 to 180 not used not used Arc (see page 356) 5 179 to 135 179 to 135 not used Arc (see page 356) 4 134 to 90 134 to 90 not used Arc (see page 356) 3 89 to 45 89 to 45 89 to 45 Arc (see page 356) 2 44 to 0 44... more (see page 204)
MtrSetTitleFont (see page 205)	This function sets the font of title.
MtrSetValueFont (see page 206)	This function sets the font of value.

RESOLUTION (see page 207)	Factor that the meter widget will divide minValue, maxValue

Structures

Name	Description
METER (see page 209)	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

This is file Meter.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Meter
 *****************************************************************************
 * FileName:        Meter.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 07/31/08     Added arc colors options
 * 08/20/08     Added accuracy option for displaying values
 * 01/18/10     Fixed MtrIncVal() and MtrDecVal() macros
 *****************************************************************************/
#ifndef _METER_H
    #define _METER_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

// Compile time Options for Meter
    #define METER_DISPLAY_VALUES_ENABLE // This enables the display of the values.

// Displaying the values will have significant
// drawing time requirement.

/*******************************************************************
```

```
* Object States Definition:
*********************************************************************/
    #define MTR_DISABLED    0x0002      // Bit for disabled state.
    #define MTR_RING        0x0004      // Bit for ring type, scales are drawn over the ring

// default is only scales drawn.
    #define MTR_ACCURACY    0x0008      // 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.
    #define MTR_DRAW_UPDATE 0x1000      // Bit to indicate an update only.
    #define MTR_DRAW        0x4000      // Bit to indicate object must be redrawn.
    #define MTR_HIDE        0x8000      // Bit to indicate object must be removed from
screen.

/*******************************************************************
* Used Constants
*********************************************************************/
    #define RADIAN      1144            // Radian definition. Equivalent to sine(1) * 2^16.
    #define PIIOVER2    102944          // The constant Pii divided by two (pii/2).

// Meter types
    #define MTR_WHOLE_TYPE      0
    #define MTR_HALF_TYPE       1
    #define MTR_QUARTER_TYPE    2

/**********************************************************************
* 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.
**********************************************************************/
    #define METER_TYPE  MTR_WHOLE_TYPE

//#define METER_TYPE                    MTR_HALF_TYPE               // Meter drawn with
semi circle shape.
//#define METER_TYPE                    MTR_QUARTER_TYPE            // Meter drawn with
quarter circle shape.
    #define ARC1_DEGREE 180             // defines one arc1 limit (used for determining
colors)
    #define ARC2_DEGREE 135             // defines one arc2 limit (used for determining
colors)
    #define ARC3_DEGREE 90              // defines one arc3 limit (used for determining
colors)
    #define ARC4_DEGREE 45              // defines one arc4 limit (used for determining
colors)
    #define ARC5_DEGREE 0               // defines one arc5 limit (used for determining
colors)

// These selects the other parameters of the meter that are dependent on the shape.
    #if (METER_TYPE == MTR_WHOLE_TYPE)
        #define DEGREE_START    - 45    // Defines the start angle to draw the meter.
        #define DEGREE_END      225     // Defines the end angle to draw the meter.
    #elif (METER_TYPE == MTR_HALF_TYPE)
        #define DEGREE_START    0       // Defines the start angle to draw the meter.
        #define DEGREE_END      180     // Defines the end angle to draw the meter.
    #elif (METER_TYPE == MTR_QUARTER_TYPE)
        #define DEGREE_START    0       // Defines the start angle to draw the meter.
        #define DEGREE_END      90      // Defines the end angle to draw the meter.
    #endif
    #define SCALECHARCOUNT  4           // Defines how many characters will be allocated
for the
```

```
// scale labels. Use this define in accordance to
// the maxValue-minValue. Example: if maxValue-minValue = 500, SCALECHARCOUNT
// should be 3. if maxValue-minValue = 90, SCALECHARCOUNT = 2
// You must include the decimal point if this
// feature is enabled (see MTR_ACCURACY state bit).
    #define DEGREECOUNT 9               // Defines how many degrees per scale, computed per
octant

// Example: for 5 division per octant 45/5 = 9.
// So every 9 degrees a scale is drawn
// for a 5 scale division per octant.
    #define RESOLUTION  10              // Factor that the meter widget will divide
minValue, maxValue

// and current value. Used only when MTR_ACCURACY state bit is set.

/*****************************************************************
* 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.
*
*****************************************************************/
typedef struct
{
    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.

    // This is computed automatically.
    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.
    WORD        arcColor6;      // Arc 6 color parameter.
    WORD        arcColor5;      // Arc 5 color parameter
    WORD        arcColor4;      // Arc 4 color parameter
    WORD        arcColor3;      // Arc 3 color parameter
    WORD        arcColor2;      // Arc 2 color parameter
    WORD        arcColor1;      // Arc 1 color parameter

    // The following three points define three fonts used in meter widget, they can be
different from the scheme font.
    // Note that the sizes of these fonts are not checked with the meter dimension. In
cases where font sizes are
    // larger than the meter dimension, some overlaps will occur.
    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
} METER;

/*****************************************************************
* Function: METER  *MtrCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                             SHORT bottom, WORD state, SHORT value,
*                             SHORT minValue, SHORT maxValue,
*                             XCHAR *pText, GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
```

```
*        left - Left most position of the object.
*        top - Top most position of the object.
*        right - Right most position of the object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        value - Initial value set to the meter.
*        minValue - The minimum value the meter will display.
*        maxValue - The maximum value the meter will display.
*        pText - Pointer to the text label of the meter.
*        pScheme - Pointer to the style scheme used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   #define ID_METER 101
*
*   extern const FONT_FLASH GOLMediumFont;      // medium font
*   extern const FONT_FLASH GOLSmallFont;       // 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*)&GOLSmallFont,   // 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);
*
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
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
        );
```

```
/*********************************************************************
* Function: WORD MtrTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pMtr  - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
WORD    MtrTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: MtrMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* 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:
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message.
*        pMtr          - The pointer to the object whose state will be modified.
*        pMsg          - The pointer to the GOL message.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void    MtrMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Macros:  MtrGetVal(pMtr)
*
* Overview: This macro returns the current value of the meter.
*           Value is always in the minValue-maxValue range inclusive.
*
* PreCondition: none
*
* Input: pMtr - Pointer to the object.
*
* Output: Returns current value of the meter.
```

```
 *
 * Side Effects: none
 *
 ***************************************************************************/
    #define MtrGetVal(pMtr) ((pMtr)->value)

/***************************************************************************
 * Function: MtrSetVal(METER *pMtr, SHORT newVal)
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: pMtr   - The pointer to the object.
 *        newVal - New value to be set for the Meter.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ***************************************************************************/
void    MtrSetVal(METER *pMtr, SHORT newVal);

/***************************************************************************
 * Macros:  MtrIncVal(pMtr, deltaValue)
 *
 * Overview: This macro is used to directly increment the value.
 *
 * PreCondition: none
 *
 * Input: pMtr - Pointer to the object.
 *        deltaValue - Number to be added to the current Meter value.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ***************************************************************************/
    #define MtrIncVal(pMtr, deltaValue)  MtrSetVal(pMtr, ((pMtr)->value + deltaValue))

/***************************************************************************
 * Macros:  MtrDecVal(pMtr, deltaValue)
 *
 * Overview: This macro is used to directly decrement the value.
 *
 * PreCondition: none
 *
 * Input: pMtr - Pointer to the object.
 *        deltaValue - Number to be subtracted to the current Meter value.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ***************************************************************************/
    #define MtrDecVal(pMtr, deltaValue)  MtrSetVal(pMtr, ((pMtr)->value - deltaValue))

/***************************************************************************
 * Macros:  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;   }
 *
 * 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.
*
* PreCondition: The object must be created (using MtrCreate()) before
*               a call to this macro is performed.
*
* Input: 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.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #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;                                         \
    }

/*********************************************************************
* Function: WORD MtrDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pMtr - Pointer to the object to be rendered.
*
* Output: 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.
*
* Example:
*   See MtrCreate() example.
*
* Side Effects: none
*
```

```
*********************************************************************/
WORD MtrDraw(void *pObj);

/*********************************************************************
* Macro: MtrSetTitleFont(pMtr, pNewFont)   (((METER*)pMtr)->pTitleFont = pNewFont)
*
* Overview: This function sets the font of title.
*
* PreCondition: Font must be created before this function is called.
*
* Input: pMtr - Pointer to the object.
*        pNewFont - Pointer to the new font used for the title.
*
* Output: N/A
*
* Side Effects: none
*
*********************************************************************/
    #define MtrSetTitleFont(pMtr, pNewFont) (((METER *)pMtr)->pTitleFont = pNewFont)

/*********************************************************************
* Macro: MtrSetValueFont(pMtr, pNewFont)   (((METER*)pMtr)->pValueFont = pNewFont)
*
* Overview: This function sets the font of value.
*
* PreCondition: Font must be created before this function is called.
*
* Input: pMtr - Pointer to the object.
*        pNewFont - Pointer to the new font used for the value.
*
* Output: N/A
*
* Side Effects: none
*
*********************************************************************/
    #define MtrSetValueFont(pMtr, pNewFont) (((METER *)pMtr)->pValueFont = pNewFont)
#endif // _METER_H
```

14.1.26 Picture.c

This is file Picture.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Picture
 *****************************************************************************
 * FileName:        Picture.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 01/19/11     Fixed bug when drawing picture from (0,0) position.
 ***************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_PICTURE

/*******************************************************************
* Function: PICTURE  *PictCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                               SHORT bottom, WORD state, char scale, void *pBitmap,
*                               GOL_SCHEME *pScheme)
*
* Overview: creates the picture control
*
*******************************************************************/
PICTURE *PictCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    char        scale,
    void        *pBitmap,
    GOL_SCHEME  *pScheme
)
{
    PICTURE *pPict = NULL;

    pPict = (PICTURE *)GFX_malloc(sizeof(PICTURE));
    if(pPict == NULL)
        return (pPict);

    pPict->hdr.ID = ID;
    pPict->hdr.pNxtObj = NULL;
    pPict->hdr.type = OBJ_PICTURE;
    pPict->hdr.left = left;
    pPict->hdr.top = top;
    pPict->hdr.right = right;
    pPict->hdr.bottom = bottom;
    pPict->pBitmap = pBitmap;
    pPict->hdr.state = state;
    pPict->scale = scale;
    pPict->hdr.DrawObj = PictDraw;          // draw function
    pPict->hdr.MsgObj = PictTranslateMsg;   // message function
    pPict->hdr.MsgDefaultObj = NULL;        // default message function
    pPict->hdr.FreeObj = NULL;              // free function

    // Set the style scheme to be used
    if(pScheme == NULL)
        pPict->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pPict->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    GOLAddObject((OBJ_HEADER *)pPict);

    return (pPict);
```

```
}

/*****************************************************************
* Function: WORD PictTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Overview: translates the GOL message for the picture control
*
*****************************************************************/
WORD PictTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    PICTURE *pPict;

    pPict = (PICTURE *)pObj;

    // Evaluate if the message is for the picture
    // Check if disabled first
    if(GetState(pPict, PICT_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the picture area
        if
        (
            (pPict->hdr.left < pMsg->param1) &&
            (pPict->hdr.right > pMsg->param1) &&
            (pPict->hdr.top < pMsg->param2) &&
            (pPict->hdr.bottom > pMsg->param2)
        )
        {
            return (PICT_MSG_SELECTED);
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*****************************************************************
* Function: WORD PictDraw(void *pObj)
*
* Output: returns the status of the drawing
*         0 - not completed
*         1 - done
*
* Overview: draws picture
*
*****************************************************************/
WORD PictDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        DRAW_IMAGE,
        DRAW_BACKGROUND1,
        DRAW_BACKGROUND2,
        DRAW_BACKGROUND3,
        DRAW_BACKGROUND4,
        DRAW_FRAME
    } PICT_DRAW_STATES;

    static PICT_DRAW_STATES state = REMOVE;
    static SHORT posleft;
    static SHORT postop;
    static SHORT posright;
    static SHORT posbottom;
    PICTURE *pPict;

    pPict = (PICTURE *)pObj;
```

```
    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(GetState(pPict, PICT_HIDE))
            {
                SetColor(pPict->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pPict->hdr.left, pPict->hdr.top, pPict->hdr.right,
pPict->hdr.bottom))
                    return (0);
                return (1);
            }

            posleft = (pPict->hdr.left + 1 + pPict->hdr.right - pPict->scale *
GetImageWidth(pPict->pBitmap)) >> 1;
            postop = (pPict->hdr.top + 1 + pPict->hdr.bottom - pPict->scale *
GetImageHeight(pPict->pBitmap)) >> 1;
            posright = (pPict->hdr.right + pPict->hdr.left + pPict->scale *
GetImageWidth(pPict->pBitmap)) >> 1;
            posbottom = (pPict->hdr.bottom + pPict->hdr.top + pPict->scale *
GetImageHeight(pPict->pBitmap)) >> 1;
            state = DRAW_IMAGE;

        case DRAW_IMAGE:
            if(pPict->pBitmap != NULL)
            {
                if(IsDeviceBusy())
                    return (0);
                if(!PutImage(posleft, postop, pPict->pBitmap, pPict->scale))
                    return (0);
            }

            SetColor(pPict->hdr.pGolScheme->CommonBkColor);
            state = DRAW_BACKGROUND1;

        case DRAW_BACKGROUND1:
            if(!Bar(pPict->hdr.left + 1, pPict->hdr.top + 1, pPict->hdr.right - 1, postop -
1))
                return (0);
            state = DRAW_BACKGROUND2;

        case DRAW_BACKGROUND2:
            if(!Bar(pPict->hdr.left + 1, posbottom, pPict->hdr.right - 1, pPict->hdr.bottom
- 1))
                return (0);
            state = DRAW_BACKGROUND3;

        case DRAW_BACKGROUND3:
            if(!Bar(pPict->hdr.left + 1, postop, posleft - 1, posbottom))
                return (0);
            state = DRAW_BACKGROUND4;

        case DRAW_BACKGROUND4:
            if(!Bar(posright, postop, pPict->hdr.right - 1, posbottom))
                return (0);
            state = DRAW_FRAME;

        case DRAW_FRAME:
            if(GetState(pPict, PICT_FRAME))
            {
                SetLineType(SOLID_LINE);
                SetColor(pPict->hdr.pGolScheme->TextColor0);
                if(!Rectangle(pPict->hdr.left, pPict->hdr.top, pPict->hdr.right,
pPict->hdr.bottom))
                    return (0);
            }

            state = REMOVE;
            return (1);
    }
```

```
    return (1);
}

#endif // USE_PICTURE
```

14.1.27 Picture.h

Functions

	Name	Description
	PictCreate (see page 212)	This function creates a PICTURE (see page 217) 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 (see page 213)	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 (see page 216)	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 (see page 217) Object.

Macros

Name	Description
PICT_DISABLED (see page 211)	Bit to indicate Picture (see page 210) is in a disabled state.
PICT_DRAW (see page 212)	Bit to indicate Picture (see page 210) will be redrawn.
PICT_FRAME (see page 212)	Bit to indicate Picture (see page 210) has a frame.
PICT_HIDE (see page 212)	Bit to indicate Picture (see page 210) must be hidden.
PictGetBitmap (see page 215)	This macro returns the pointer to the bitmap used in the object.
PictGetScale (see page 215)	This macro returns the current scale factor used to render the bitmap.
PictSetBitmap (see page 214)	This macro sets the bitmap used in the object.
PictSetScale (see page 216)	This macro sets the scale factor used to render the bitmap used in the object.

Structures

Name	Description
PICTURE (see page 217)	The structure contains data for picture control

Description

This is file Picture.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Picture control
 *****************************************************************************
 * FileName:        Picture.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 03/09/11     Removed compile warnings
 ***************************************************************************/
#ifndef _PICTURE_H
    #define _PICTURE_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define PICT_DISABLED   0x0002  // Bit to indicate Picture is in a disabled state.
    #define PICT_FRAME      0x0004  // Bit to indicate Picture has a frame.
    #define PICT_HIDE       0x8000  // Bit to indicate Picture must be hidden.
    #define PICT_DRAW       0x4000  // Bit to indicate Picture will be redrawn.

/****************************************************************************
 * Overview: The structure contains data for picture control
 ****************************************************************************/
typedef struct
{
    OBJ_HEADER  hdr;        // Generic header for all Objects (see OBJ_HEADER).
    char        scale;      // Scale factor for the bitmap
    void        *pBitmap;   // Pointer to the bitmap
} PICTURE;

/*********************************************************************
* Macros:  PictGetBitmap(pPict)
*
* Overview: This macro returns the pointer to the bitmap used in the object.
*
* PreCondition: none
*
* Input: pPict - Pointer to the object
*
* Output: Returns the pointer to the bitmap used.
*
* Side Effects: none
*
*********************************************************************/
    #define PictGetBitmap(pPict)            ((PICTURE*)pPict)->pBitmap

/*********************************************************************
* Macros:  PictSetBitmap(pPict,pBtMap)
*
* Overview: This macro sets the bitmap used in the object.
*
* PreCondition: none
*
* Input: pPict - Pointer to the object
*        pBtMap - Pointer to the bitmap to be used
```

```
 *
 * Output: none
 *
 * Side Effects: none
 *
 ***********************************************************************/
    #define PictSetBitmap(pPict, pBtmap)    ((PICTURE*)pPict)->pBitmap = pBtmap


/***********************************************************************
 * Macros:  PictSetScale(pPict,scl)
 *
 * Overview: This macro sets the scale factor used to render the
 *           bitmap used in the object.
 *
 * PreCondition: none
 *
 * Input: pPict - Pointer to the object
 *        scl   - The scale factor that will be used to display the
 *                bitmap.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ***********************************************************************/
    #define PictSetScale(pPict, scl)    pPict->scale = scl

/***********************************************************************
 * Macros:  PictGetScale(pPict,scl)
 *
 * Overview: This macro returns the current scale factor used to
 *           render the bitmap.
 *
 * PreCondition: none
 *
 * Input: pPict - Pointer to the object
 *
 * Output: Returns the current scale factor used to display the
 *         bitmap.
 *
 * Side Effects: none
 *
 ***********************************************************************/
    #define PictGetScale(pPict) pPict->scale

/***********************************************************************
 * Function: PICTURE  *PictCreate(WORD ID, SHORT left, SHORT top, SHORT right,
 *                                SHORT bottom, WORD state, char scale,
 *                                void *pBitmap, GOL_SCHEME *pScheme)
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: ID - Unique user defined ID for the object instance.
 *        left - Left most position of the Object.
 *        top - Top most position of the Object.
 *        right - Right most position of the Object.
 *        bottom - Bottom most position of the object.
 *        radius - Radius of the rounded edge.
 *        state - Sets the initial state of the object.
 *        scale - Sets the scale factor used to render the bitmap.
 *        pBitmap - Pointer to the bitmap that will be used.
 *        pScheme - Pointer to the style scheme
 *
 * Output: Returns the pointer to the object created
 *
 * Side Effects: none
 *
```

```
*********************************************************************/
PICTURE * PictCreate
    (
        WORD ID, SHORT left, SHORT top, SHORT right, SHORT bottom, WORD state, char scale,
void *pBitmap, GOL_SCHEME *
        pScheme
    );

/*********************************************************************
* Function: WORD PictTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pPict - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: Returns the translated message depending on the received GOL message:
*         - PICT_MSG_SELECTED – Picture is touched.
*         - OBJ_MSG_INVALID – Picture is not affected
*
* Side Effects: none
*
*********************************************************************/
WORD    PictTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD PictDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pPict - Pointer to the object to be rendered.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
WORD PictDraw(void *pObj);
#endif // _PICTURE_H
```

14.1.28 ProgressBar.c

This is file ProgressBar.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Progress Bar
 *****************************************************************************
 * FileName:        ProgressBar.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Anton Alkhimenok     11/12/07    Version 1.0 release
 *****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_PROGRESSBAR

/*******************************************************************
* Function: void PbWordToString(WORD value, XCHAR* buffer)
*
* Input: value  - value to be converted (from 0 - 100)
*        buffer - buffer receiving string (must be at least 5 bytes)
*
* Output: none
*
* Overview: converts SHORT into string with % at the end
*
*******************************************************************/
void PbWordToString(WORD value, XCHAR *buffer)
{
    WORD    result;
    BYTE    pos;

    if(value > 99)
```

```
    {
        buffer[0] = '1';
        buffer[1] = '0';
        buffer[2] = '0';
        buffer[3] = '%';
        buffer[4] = 0;
        return;
    }

    pos = 0;
    result = value / 10;
    if(result)
        buffer[pos++] = result + '0';
    result = value - 10 * result;

    buffer[pos++] = result + '0';
    buffer[pos++] = '%';
    buffer[pos++] = 0;
}

/*********************************************************************
* Function: PROGRESSBAR  *PbCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                                  SHORT bottom, WORD state, WORD pos, WORD range,
*                                  GOL_SCHEME *pScheme)
*
* Overview: creates the progress bar
*
*********************************************************************/
PROGRESSBAR *PbCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    WORD        pos,
    WORD        range,
    GOL_SCHEME  *pScheme
)
{
    PROGRESSBAR *pPb = NULL;

    pPb = (PROGRESSBAR *)GFX_malloc(sizeof(PROGRESSBAR));
    if(pPb == NULL)
        return (pPb);

    pPb->hdr.ID         = ID;
    pPb->hdr.pNxtObj    = NULL;
    pPb->hdr.type       = OBJ_PROGRESSBAR;
    pPb->hdr.left       = left;
    pPb->hdr.top        = top;
    pPb->hdr.right      = right;
    pPb->hdr.bottom     = bottom;
    pPb->pos            = pos;
    pPb->range          = (range == 0) ? 100 : range;
    pPb->prevPos        = 0;
    pPb->hdr.state      = state;
    pPb->hdr.DrawObj    = PbDraw;   // draw function
    pPb->hdr.MsgObj     = NULL;     // message function
    pPb->hdr.MsgDefaultObj = NULL;  // default message function
    pPb->hdr.FreeObj    = NULL;     // free function

    // Set the style scheme to be used
    if(pScheme == NULL)
        pPb->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pPb->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    GOLAddObject((OBJ_HEADER *)pPb);

    return (pPb);
```

```
}

/*********************************************************************
* Function: void PbSetPos(PROGRESSBAR *pPb, SHORT position)
*
* Overview: sets the current position of the progress bar
*
*********************************************************************/
void PbSetPos(PROGRESSBAR *pPb, WORD position)
{
    if(pPb->range < position)
        position = pPb->range;

    pPb->pos = position;
}

/*********************************************************************
* Function: void PbSetRange(PROGRESSBAR *pPb, WORD range)
*
* Overview: sets the range of the progress bar
*
*********************************************************************/
void PbSetRange(PROGRESSBAR *pPb, WORD range)
{

    // range cannot be assigned a zero value
    if(range != 0)
        pPb->range = range;
    pPb->pos = range;
    pPb->prevPos = 0;
}

/*********************************************************************
* Function: WORD PbTranslateMsg(PROGRESSBAR *pPb, GOL_MSG *pMsg)
*
* Overview: translates the GOL message for the progress bar
*
*********************************************************************/
WORD PbTranslateMsg(PROGRESSBAR *pPb, GOL_MSG *pMsg)
{

    // Evaluate if the message is for the progress bar
    // Check if disabled first
    if(GetState(pPb, PB_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the progress bar border
        if
        (
            (pPb->hdr.left < pMsg->param1) &&
            (pPb->hdr.right > pMsg->param1) &&
            (pPb->hdr.top < pMsg->param2) &&
            (pPb->hdr.bottom > pMsg->param2)
        )
        {
            return (PB_MSG_SELECTED);
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*********************************************************************
* Function: WORD PbDraw(void *pObj)
*
* Output: returns the status of the drawing
*         0 - not complete
```

```
*         1 - done
*
* Overview: draws progress bar
*
*****************************************************************/
WORD PbDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        BOX_DRAW,
        RUN_DRAW,
        BAR_DRAW,
        TEXT_DRAW1,
        TEXT_DRAW2,
        TEXT_DRAW3
    } PB_DRAW_STATES;

    static PB_DRAW_STATES state = REMOVE;
    static DWORD x1;
    static DWORD x2;
    static XCHAR text[5] = {'0','0','%',0};
    PROGRESSBAR *pPb;

    pPb = (PROGRESSBAR *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(GetState(pPb, PB_HIDE))
            {
                SetColor(pPb->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pPb->hdr.left, pPb->hdr.top, pPb->hdr.right, pPb->hdr.bottom))
                    return (0);
                return (1);
            }

            state = BOX_DRAW;

        case BOX_DRAW:
            if(GetState(pPb, PB_DRAW))
            {
                GOLPanelDraw
                (
                    pPb->hdr.left,
                    pPb->hdr.top,
                    pPb->hdr.right,
                    pPb->hdr.bottom,
                    0,
                    pPb->hdr.pGolScheme->Color0,
                    pPb->hdr.pGolScheme->EmbossDkColor,
                    pPb->hdr.pGolScheme->EmbossLtColor,
                    NULL,
                    GOL_EMBOSS_SIZE
                );

                state = RUN_DRAW;

            case RUN_DRAW:
                if(!GOLPanelDrawTsk())
                    return (0);
            }

            state = BAR_DRAW;

        case BAR_DRAW:
            if(GetState(pPb, PB_VERTICAL))
            {
                x2 = ((DWORD) (pPb->range-pPb->pos)) * (pPb->hdr.bottom - pPb->hdr.top - (2
```

```
* GOL_EMBOSS_SIZE)) / pPb->range;
                x1 = ((DWORD) (pPb->range-pPb->prevPos)) * (pPb->hdr.bottom - pPb->hdr.top
- (2 * GOL_EMBOSS_SIZE)) / pPb->range;
                x1 += (pPb->hdr.top + GOL_EMBOSS_SIZE);
                x2 += (pPb->hdr.top + GOL_EMBOSS_SIZE);
            }
            else
            {
                x1 = ((DWORD) pPb->pos) * (pPb->hdr.right - pPb->hdr.left - (2 *
GOL_EMBOSS_SIZE)) / pPb->range;
                x2 = ((DWORD) pPb->prevPos) * (pPb->hdr.right - pPb->hdr.left - (2 *
GOL_EMBOSS_SIZE)) / pPb->range;
                x1 += (pPb->hdr.left + GOL_EMBOSS_SIZE);
                x2 += (pPb->hdr.left + GOL_EMBOSS_SIZE);
            }

            if(pPb->prevPos > pPb->pos)
            {
                SetColor(pPb->hdr.pGolScheme->Color0);
                if(GetState(pPb, PB_VERTICAL))
                {
                    if(!Bar(pPb->hdr.left + GOL_EMBOSS_SIZE, x1, pPb->hdr.right -
GOL_EMBOSS_SIZE, x2))
                        return (0);
                }
                else
                {
                    if(!Bar(x1, pPb->hdr.top + GOL_EMBOSS_SIZE, x2, pPb->hdr.bottom -
GOL_EMBOSS_SIZE))
                        return (0);
                }
            }
            else
            {
                SetColor(pPb->hdr.pGolScheme->Color1);
                if(GetState(pPb, PB_VERTICAL))
                {
                    if(!Bar(pPb->hdr.left + GOL_EMBOSS_SIZE, x2, pPb->hdr.right -
GOL_EMBOSS_SIZE, x1))
                        return (0);
                }
                else
                {
                    if(!Bar(x2, pPb->hdr.top + GOL_EMBOSS_SIZE, x1, pPb->hdr.bottom -
GOL_EMBOSS_SIZE))
                        return (0);
                }
            }

            state = TEXT_DRAW1;

        case TEXT_DRAW1:
            if(GetState(pPb, PB_VERTICAL))
            {
                SetColor(pPb->hdr.pGolScheme->Color0);
                if
                (
                    !Bar
                        (
                            pPb->hdr.left + GOL_EMBOSS_SIZE,
                            (pPb->hdr.top + pPb->hdr.bottom -
GetTextHeight(pPb->hdr.pGolScheme->pFont)) >> 1,
                            pPb->hdr.right - GOL_EMBOSS_SIZE,
                            x2
                        )
                ) return (0);
            }
            else
            {
                SetColor(pPb->hdr.pGolScheme->Color1);
                if
                (
```

```
                    !Bar
                        (
                            (pPb->hdr.left + pPb->hdr.right - GetTextWidth(text,
pPb->hdr.pGolScheme->pFont)) >> 1,
                            pPb->hdr.top + GOL_EMBOSS_SIZE,
                            x1,
                            pPb->hdr.bottom - GOL_EMBOSS_SIZE
                        )
                ) return (0);
            }

            state = TEXT_DRAW2;

        case TEXT_DRAW2:
            if(GetState(pPb, PB_VERTICAL))
            {
                SetColor(pPb->hdr.pGolScheme->Color1);
                if
                (
                    !Bar
                        (
                            pPb->hdr.left + GOL_EMBOSS_SIZE,
                            x2,
                            pPb->hdr.right - GOL_EMBOSS_SIZE,
                            (pPb->hdr.top + pPb->hdr.bottom +
GetTextHeight(pPb->hdr.pGolScheme->pFont)) >> 1
                        )
                ) return (0);
            }
            else
            {
                SetColor(pPb->hdr.pGolScheme->Color0);
                if
                (
                    !Bar
                        (
                            x1,
                            pPb->hdr.top + GOL_EMBOSS_SIZE,
                            (pPb->hdr.left + pPb->hdr.right + GetTextWidth(text,
pPb->hdr.pGolScheme->pFont)) >> 1,
                            pPb->hdr.bottom - GOL_EMBOSS_SIZE
                        )
                ) return (0);
            }

            PbWordToString((DWORD) pPb->pos * 100 / pPb->range, text);
            SetColor(pPb->hdr.pGolScheme->TextColor0);

            MoveTo
            (
                (pPb->hdr.left + pPb->hdr.right - GetTextWidth(text,
pPb->hdr.pGolScheme->pFont)) >> 1,
                (pPb->hdr.top + pPb->hdr.bottom -
GetTextHeight(pPb->hdr.pGolScheme->pFont)) >> 1
            );

            SetFont(pPb->hdr.pGolScheme->pFont);
            state = TEXT_DRAW3;

        case TEXT_DRAW3:
            if(!OutText(text))
                return (0);

            pPb->prevPos = pPb->pos;
            state = REMOVE;
            return (1);
    }

    return (1);
}

#endif // USE_PROGRESSBAR
```

14.1.29 ProgressBar.h

Functions

	Name	Description
	PbCreate (see page 220)	This function creates a PROGRESSBAR (see page 226) 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 (see page 221)	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.
	PbSetPos (see page 223)	This function sets the position of the progress bar. Position should be in the given range inclusive.
	PbSetRange (see page 222)	This function sets the range of the progress bar. Calling this function also resets the position equal to the new range value.
	PbTranslateMsg (see page 225)	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
PB_DISABLED (see page 219)	Bit to indicate Progress Bar (see page 352) is in a disabled state.
PB_DRAW (see page 219)	Bit to indicate Progress Bar (see page 352) must be redrawn.
PB_DRAW_BAR (see page 220)	Bit to indicate Progress Bar (see page 352) must be redrawn.
PB_HIDE (see page 220)	Bit to indicate Progress Bar (see page 352) must be hidden.
PB_VERTICAL (see page 220)	Bit for orientation (0 - horizontal, 1 - vertical)
PbGetPos (see page 224)	This macro returns the current progress bar position.
PbGetRange (see page 223)	This macro returns the current range of the progress bar.

Structures

Name	Description
PROGRESSBAR (see page 226)	The structure contains data for the progress bar

Description

This is file ProgressBar.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Progress Bar
 *****************************************************************************
 * FileName:        ProgressBar.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 ****************************************************************************/
#ifndef _PROGRESSBAR_H
    #define _PROGRESSBAR_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define PB_DISABLED 0x0002  // Bit to indicate Progress Bar is in a disabled state.
    #define PB_VERTICAL 0x0004  // Bit for orientation  (0 - horizontal, 1 - vertical)
    #define PB_HIDE     0x8000  // Bit to indicate Progress Bar must be hidden.
    #define PB_DRAW_BAR 0x2000  // Bit to indicate Progress Bar must be redrawn.
    #define PB_DRAW     0x4000  // Bit to indicate Progress Bar must be redrawn.

/*****************************************************************************
 * Overview: The structure contains data for the progress bar
 *****************************************************************************/
typedef struct
{
    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.
} PROGRESSBAR;

/*********************************************************************
* Macros:  PbGetPos(pPb)
*
* Overview: This macro returns the current progress bar position.
*
* PreCondition: none
*
* Input: pPb - Pointer to the object
*
* Output: Returns the progress bar position.
*
* Example:
*   See PbSetPos() exmaple.
*
* Side Effects: none
*
*********************************************************************/
    #define PbGetPos(pPb)   pPb->pos

/*********************************************************************
* Function: void PbSetPos(PROGRESSBAR *pPb, WORD position)
*
```

```
* Overview: This function sets the position of the progress bar.
*           Position should be in the given range inclusive.
*
* PreCondition: none
*
* Input: pPb - Pointer to the object
*        position - New position.
*
* Output: none
*
* Example:
*   <CODE>
*   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);
*           }
*       }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
void        PbSetPos(PROGRESSBAR *pPb, WORD position);

/*********************************************************************
* Function: PROGRESSBAR  *PbCreate( WORD ID, SHORT left, SHORT top,
*                                   SHORT right, SHORT bottom, WORD state,
*                                   WORD pos, WORD range, GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object.
*        bottom - Bottom most position of the Object.
*        state - Sets the initial state of the Object.
*        pos - Defines the initial position of the progress.
*        range - This specifies the maximum value of the progress
*                bar when the progress bar is at 100% position.
*        pScheme - Pointer to the style scheme used for the object.
*                  Set to NULL if default style scheme is used.
*
* Output: Returns the pointer to the object created
*
* Example:
*   <CODE>
*   PROGRESSBAR *pPBar;
*   void CreateProgressBar(){
*       pPBar = PbCreate(ID_PROGRESSBAR1,     // ID
*                        50,90,270,140,       // dimension
*                        PB_DRAW,             // Draw the object
*                        25,                  // position
*                        50,                  // set the range
*                        NULL);               // use default GOL scheme
```

```
*       while(!PbDraw(pPBar));
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
PROGRESSBAR *PbCreate
            (
                WORD        ID,
                SHORT       left,
                SHORT       top,
                SHORT       right,
                SHORT       bottom,
                WORD        state,
                WORD        pos,
                WORD        range,
                GOL_SCHEME  *pScheme
            );

/*********************************************************************
* Function: WORD PbTranslateMsg(PROGRESSBAR *pPb, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pPb   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD        PbTranslateMsg(PROGRESSBAR *pPb, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD PbDraw(void *pObj)
*
* 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.
*
```

```
* PreCondition: Object must be created before this function is called.
*
* Input: pPb - Pointer to the object to be rendered.
*
* Output: 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.
*
* Example:
*   See PbCreate() example.
*
* Side Effects: none
*
*********************************************************************/
WORD PbDraw(void *pObj);

/*********************************************************************
* Function:  PbSetRange(PROGRESSBAR *pPb, WORD range)
*
* Overview: This function sets the range of the progress bar. Calling
*           this function also resets the position equal to the new
*           range value.
*
* PreCondition: none
*
* Input: pPb - Pointer to the object
*
* Output: none.
*
* Side Effects: Sets the position equal to the new range.
*
*********************************************************************/
void        PbSetRange(PROGRESSBAR *pPb, WORD range);

/*********************************************************************
* Macros:  PbGetRange(pPb)
*
* Overview: This macro returns the current range of the progress bar.
*
* PreCondition: none
*
* Input: pPb - Pointer to the object
*
* Output: Returns the range value.
*
* Side Effects: none
*
*********************************************************************/
    #define PbGetRange(pPb) (pPb->range)
#endif // _PROGRESSBAR_H
```

14.1.30 RoundDial.c

This is file RoundDial.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Round Dial
 *****************************************************************************
 * FileName:        RoundDial.c
 * Dependencies:    math.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 Version 3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 12/03/09     Added Object Header for Double Buffering Support
 * 11/15/10     Fixed build error when USE_KEYBOARD is not defined
 * 04/20/11     Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 ****************************************************************************/
#include "Graphics/Graphics.h"
#include <math.h>

#ifdef USE_ROUNDDIAL

/*******************************************************************
* Function: ROUNDDIAL  *RdiaCreate( WORD ID, SHORT x, SHORT y, SHORT radius,
*                       WORD state, SHORT res, SHORT value, SHORT max,
*                       GOL_SCHEME *pScheme)
*
*
* Notes: Creates a ROUNDDIAL object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*******************************************************************/
ROUNDDIAL *RdiaCreate
(
    WORD        ID,
    SHORT       x,
    SHORT       y,
    SHORT       radius,
    WORD        state,
    SHORT       res,
    SHORT       value,
    SHORT       max,
    GOL_SCHEME  *pScheme
)
{
    ROUNDDIAL   *pDia = NULL;

    pDia = (ROUNDDIAL *)GFX_malloc(sizeof(ROUNDDIAL));
    if(pDia == NULL)
        return (NULL);

    pDia->hdr.ID = ID;              // unique id assigned for referencing
    pDia->hdr.pNxtObj = NULL;       // initialize pointer to NULL
    pDia->hdr.type = OBJ_ROUNDDIAL; // set object type
```

```
    pDia->xCenter = x;              // x coordinate of center
    pDia->yCenter = y;              // y coordinate of center
    pDia->radius = radius;          // radius of dial
    pDia->res = res;
    pDia->value = value;
    pDia->max = max;
    pDia->hdr.state = state;        // state
    #ifdef USE_KEYBOARD
    pDia->vAngle = 0;               // initial position
    #endif
    pDia->curr_xPos = x + radius * 2 / 3;
    pDia->curr_yPos = y;
    pDia->hdr.DrawObj = RdiaDraw;               // draw function
    pDia->hdr.MsgObj = RdiaTranslateMsg;        // message function
    pDia->hdr.MsgDefaultObj = RdiaMsgDefault;   // default message function
    pDia->hdr.FreeObj = NULL;                   // free function

    pDia->hdr.left = x - radius;    // left position
    pDia->hdr.top = y - radius;     // top position
    pDia->hdr.right = x + radius;   // right position
    pDia->hdr.bottom = y + radius;  // bottom position

    // Set the color scheme to be used
    if(pScheme == NULL)
        pDia->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pDia->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    GOLAddObject((OBJ_HEADER *)pDia);

    return (pDia);
}

/*********************************************************************
* Function: RdiaMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
*
* Notes: This the default operation to change the state of the dial.
*        Called inside GOLMsg() when GOLMsgCallback() returns a 1.
*
*********************************************************************/
void RdiaMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    ROUNDDIAL *pDia;

    pDia = (ROUNDDIAL *)pObj;

    switch(translatedMsg)
    {
        case RD_MSG_CLOCKWISE:      SetState(pDia, RDIA_ROT_CW | RDIA_DRAW);    // set
rotate left and redraw
            break;
        case RD_MSG_CTR_CLOCKWISE:  SetState(pDia, RDIA_ROT_CCW | RDIA_DRAW);   // set
rotate right and redraw
            break;
    }
}

/*********************************************************************
* Function: SHORT RdiaCosine( SHORT v )
*
*
* Notes: Returns the cosine of the dial position.
*
*********************************************************************/
    #ifdef USE_KEYBOARD

// Dimple position table for 15 degree increments
SHORT   _cosine[RDIA_QUADRANT_POSITIONS] = { 100, 97, 87, 71, 50, 26};

/* */
SHORT RdiaCosine(SHORT v)
```

```
{
    if(v >= RDIA_QUADRANT_POSITIONS * 3)
    {
        v -= RDIA_QUADRANT_POSITIONS * 3;
        return (_cosine[RDIA_QUADRANT_POSITIONS - 1 - v]);
    }
    else if(v >= RDIA_QUADRANT_POSITIONS * 2)
    {
        v -= RDIA_QUADRANT_POSITIONS * 2;
        return (-(_cosine[v]));
    }
    else if(v >= RDIA_QUADRANT_POSITIONS)
    {
        v -= RDIA_QUADRANT_POSITIONS;
        return (-(_cosine[RDIA_QUADRANT_POSITIONS - 1 - v]));
    }
    else
    {
        return (_cosine[v]);
    }
}

/*****************************************************************
* Function: SHORT RdiaSine( SHORT v )
*
*
* Notes: Returns the sine of the dial position.
*
*****************************************************************/
SHORT RdiaSine(SHORT v)
{
    if(v >= RDIA_QUADRANT_POSITIONS * 3)
    {
        v -= RDIA_QUADRANT_POSITIONS * 3;
        return (-(_cosine[v]));
    }
    else if(v >= RDIA_QUADRANT_POSITIONS * 2)
    {
        v -= RDIA_QUADRANT_POSITIONS * 2;
        return (-(_cosine[RDIA_QUADRANT_POSITIONS - 1 - v]));
    }
    else if(v >= RDIA_QUADRANT_POSITIONS)
    {
        v -= RDIA_QUADRANT_POSITIONS;
        return (_cosine[v]);
    }
    else
    {
        return (_cosine[RDIA_QUADRANT_POSITIONS - 1 - v]);
    }
}
    #endif

/*****************************************************************
* Function: WORD RdiaTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*****************************************************************/
WORD RdiaTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    ROUNDDIAL *pDia;
    WORD      messageID = OBJ_MSG_INVALID;

    pDia = (ROUNDDIAL *)pObj;

        #ifdef USE_TOUCHSCREEN

    SHORT           touchRadius, touchX, touchY;
    static SHORT    prevX = -1, prevY = -1;
```

```
    // Evaluate if the message is for the button
    // Check if disabled first
    if(GetState(pDia, RDIA_DISABLED))
        return (OBJ_MSG_INVALID);

    if((pMsg->type == TYPE_TOUCHSCREEN) && (pMsg->uiEvent == EVENT_MOVE))
    {

        // Check if it falls in the dial's face
        // to check this the x,y position must be within the circle
        // (x - xCenter)^2 + (y - yCenter)^2 = r^2  where x and y are the points
        // to test, the distance between x,y position of touch and center must be
        // greater than or equal to the radius of the dial
        if
        (
            ((pDia->xCenter - pDia->radius) < pMsg->param1) &&
            ((pDia->xCenter + pDia->radius) > pMsg->param1) &&
            ((pDia->yCenter - pDia->radius) < pMsg->param2) &&
            ((pDia->yCenter + pDia->radius) > pMsg->param2)
        )
        {

            // first get the radius of the touch point
            touchX = pMsg->param1 - pDia->xCenter;
            touchY = pMsg->param2 - pDia->yCenter;
            touchRadius = sqrt(touchX * touchX + touchY * touchY);

            if(touchRadius <= pDia->radius)
            {

                // difference of 3 is used to remove jitter caused by noise or sensitivity
of the touchscreen
                if((abs(prevX - pMsg->param1) > 3) || (abs(prevY - pMsg->param2) > 3))
                {

                    // The first MOVE event is used to record the current position only.
The second MOVE event
                    // will be used together with the previous MOVE and determine if the
movement is in the
                    // clockwise or counter clockwise direction.
                    if((prevX == -1) || (prevY == -1))
                    {
                        prevX = pMsg->param1;
                        prevY = pMsg->param2;
                        messageID = OBJ_MSG_INVALID;
                    }
                    else
                    {

                        // this makes the sampling area a ring where the max radius is the
dial radius
                        // and min radius is 5
                        if(touchRadius > 5)
                        {
                            pDia->new_xPos = (pDia->radius * 2 / 3) * (pMsg->param1 -
pDia->xCenter) / touchRadius;
                            pDia->new_yPos = (pDia->radius * 2 / 3) * (pMsg->param2 -
pDia->yCenter) / touchRadius;

                            // check if moving in clockwise direction or counter clockwise
direction
                            if((pDia->xCenter >= pMsg->param1) && (pDia->yCenter >
pMsg->param2))
                            {
                                if((prevX < pMsg->param1) && (prevY >= pMsg->param2))
                                    messageID = RD_MSG_CLOCKWISE;
                                else if((prevX >= pMsg->param1) && (prevY < pMsg->param2))
                                    messageID = RD_MSG_CTR_CLOCKWISE;
                            }

                            if((pDia->xCenter < pMsg->param1) && (pDia->yCenter >
```

```
pMsg->param2))
                                {
                                    if((prevX < pMsg->param1) && (prevY <= pMsg->param2))
                                        messageID = RD_MSG_CLOCKWISE;
                                    else if((prevX >= pMsg->param1) && (prevY > pMsg->param2))
                                        messageID = RD_MSG_CTR_CLOCKWISE;
                                }

                                if((pDia->xCenter < pMsg->param1) && (pDia->yCenter <=
pMsg->param2))
                                {
                                    if((prevX > pMsg->param1) && (prevY <= pMsg->param2))
                                        messageID = RD_MSG_CLOCKWISE;
                                    else if((prevX <= pMsg->param1) && (prevY > pMsg->param2))
                                        messageID = RD_MSG_CTR_CLOCKWISE;
                                }

                                if((pDia->xCenter >= pMsg->param1) && (pDia->yCenter <=
pMsg->param2))
                                {
                                    if((prevX > pMsg->param1) && (prevY >= pMsg->param2))
                                        messageID = RD_MSG_CLOCKWISE;
                                    else if((prevX <= pMsg->param1) && (prevY < pMsg->param2))
                                        messageID = RD_MSG_CTR_CLOCKWISE;
                                }
                            }
                            else
                                messageID = OBJ_MSG_INVALID;

                            prevX = pMsg->param1;
                            prevY = pMsg->param2;
                        }
                    }
                    else
                        messageID = OBJ_MSG_INVALID;

                    // determine the movement clockwise or counter clockwise
                    // this is important to update the value variable
                    return (messageID);
                }
                else
                {
                    prevX = -1;
                    prevY = -1;
                }
            }
        }

            #endif
           #ifdef USE_KEYBOARD

        SHORT   newValue;

        // Evaluate if the message is for the Dial
        // Check if disabled first
        if(GetState(pDia, RDIA_DISABLED))
            return (OBJ_MSG_INVALID);

        if((pMsg->type == TYPE_KEYBOARD) && ((WORD)pMsg->param1 == pDia->hdr.ID) &&
(pMsg->uiEvent == EVENT_KEYSCAN))
        {
            if(pMsg->param2 == SCAN_RIGHT_PRESSED)
            {
                newValue = pDia->value + pDia->res;
                if(newValue > pDia->max)
                {
                    newValue -= (pDia->max + 1);
                }

                pDia->vAngle += 1;
                messageID = RD_MSG_CLOCKWISE;
            }
```

```
        if(pMsg->param2 == SCAN_LEFT_PRESSED)
        {
            newValue = pDia->value - pDia->res;
            if(newValue < 0)
            {
                newValue += (pDia->max + 1);
            }

            pDia->vAngle -= 1;
            messageID = RD_MSG_CTR_CLOCKWISE;
        }

        if (pDia->vAngle > (RDIA_MAX_POSITIONS - 1))
            pDia->vAngle = 0;
        else if (pDia->vAngle < 0)
            pDia->vAngle = (RDIA_MAX_POSITIONS - 1);

        pDia->new_xPos = pDia->radius * 2 * RdiaCosine(pDia->vAngle) / 100 / 3;
        pDia->new_yPos = pDia->radius * 2 * RdiaSine(pDia->vAngle) / 100 / 3;
    }

        #endif
    return (messageID);
}

/*********************************************************************
* Function: WORD RdiaDraw(void *pObj)
*
*
* Notes: This is the state machine to draw the dial.
*
*********************************************************************/
WORD RdiaDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        RND_PANEL_DRAW,
        RND_PANEL_TASK,
        ERASE_POSITION,
        DRAW_POSITION
    } RDIA_DRAW_STATES;

    static RDIA_DRAW_STATES state = REMOVE;
    static SHORT dimpleRadius;
    WORD faceClr;
    ROUNDDIAL *pDia;

    pDia = (ROUNDDIAL *)pObj;

    switch(state)
    {
        case REMOVE:
            if(IsDeviceBusy())
                return (0);

            if(GetState(pDia, RDIA_HIDE))
            {   // Hide the dial (remove from screen)
                SetColor(pDia->hdr.pGolScheme->CommonBkColor);
                if
                (
                    !Bar
                        (
                            pDia->xCenter - pDia->radius,
                            pDia->yCenter - pDia->radius,
                            pDia->xCenter + pDia->radius,
                            pDia->yCenter + pDia->radius
                        )
                ) return (0);
                return (1);
            }
```

```
        dimpleRadius = (pDia->radius >> 3) + 1;

        if(GetState(pDia, RDIA_ROT_CCW | RDIA_ROT_CW))
        {
            state = ERASE_POSITION;
            goto erase_current_pos;
        }

        state = RND_PANEL_DRAW;

    case RND_PANEL_DRAW:
        if(!GetState(pDia, RDIA_DISABLED))
        {
            faceClr = pDia->hdr.pGolScheme->Color0;
        }
        else
        {
            faceClr = pDia->hdr.pGolScheme->ColorDisabled;
        }

        SetLineThickness(NORMAL_LINE);
        SetLineType(SOLID_LINE);
        GOLPanelDraw
        (
            pDia->xCenter,
            pDia->yCenter,
            pDia->xCenter,
            pDia->yCenter,
            pDia->radius,
            faceClr,
            pDia->hdr.pGolScheme->EmbossLtColor,
            pDia->hdr.pGolScheme->EmbossDkColor,
            NULL,
            GOL_EMBOSS_SIZE
        );
        state = RND_PANEL_TASK;

    case RND_PANEL_TASK:
        if(!GOLPanelDrawTsk())
        {
            return (0);
        }

        state = DRAW_POSITION;
        goto draw_current_pos;

    case ERASE_POSITION:
        erase_current_pos : SetColor(pDia->hdr.pGolScheme->Color0);
        if
        (
            !Bar
                (
                    pDia->curr_xPos - dimpleRadius,
                    pDia->curr_yPos - dimpleRadius,
                    pDia->curr_xPos + dimpleRadius,
                    pDia->curr_yPos + dimpleRadius
                )
        ) return (0);

        // determine if the value will increment or decrement
        if(GetState(pDia, RDIA_ROT_CW))
        {
            pDia->value = pDia->value + pDia->res;
            if(pDia->value > pDia->max)
            {
                pDia->value -= (pDia->max + 1);
            }
        }
        else if(GetState(pDia, RDIA_ROT_CCW))
        {
            pDia->value = pDia->value - pDia->res;
```

```
                if(pDia->value < 0)
                {
                    pDia->value += (pDia->max + 1);
                }
            }

            // else do not update counter yet
            // locate the new position of the dimple
            pDia->curr_xPos = pDia->xCenter + pDia->new_xPos;
            pDia->curr_yPos = pDia->yCenter + pDia->new_yPos;

            ClrState(pDia, RDIA_ROT_CW | RDIA_ROT_CCW); // make sure this is cleared to
avoid

            // unwanted redraw
            state = DRAW_POSITION;

        case DRAW_POSITION:
            draw_current_pos : if(IsDeviceBusy()) return (0);

            SetColor(pDia->hdr.pGolScheme->EmbossLtColor);
            SetLineThickness(NORMAL_LINE);
            SetLineType(SOLID_LINE);
            if(!Circle(pDia->curr_xPos, pDia->curr_yPos, dimpleRadius))
                return (0);
            SetColor(pDia->hdr.pGolScheme->EmbossDkColor);
            if(!FillCircle(pDia->curr_xPos, pDia->curr_yPos, dimpleRadius - 1))
                return (0);

            state = REMOVE;
            return (1);
    }

    return (1);
}

#endif // USE_ROUNDDIAL
```

14.1.31 RoundDial.h

Functions

	Name	Description
	RdiaCreate (see page 127)	This function creates a ROUNDDIAL (see page 134) 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 (see page 128)	This function renders the object on the screen using the current parameter settings. Location of the object is determined by the center (x,y) postion 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 (see page 132)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	RdiaTranslateMsg (see page 132)	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
RDIA_DISABLED (see page 126)	Bit for disabled state.
RDIA_DRAW (see page 126)	Bit to indicate object must be redrawn.
RDIA_HIDE (see page 126)	Bit to indicate object must be removed from screen.

RDIA_ROT_CCW (see page 127)	Bit for rotate counter clockwise state.
RDIA_ROT_CW (see page 127)	Bit for rotate clockwise state.
RdiaDecVal (see page 130)	Used to directly decrement the value. The delta change used is the resolution setting (res).
RdiaGetVal (see page 130)	Returns the current dial value. Value is always in the 0-max range inclusive.
RdiaIncVal (see page 129)	Used to directly increment the value. The delta change used is the resolution setting (res).
RdiaSetVal (see page 131)	Sets the value to the given new value. Value set must be in 0-max range inclusive.

Structures

Name	Description
ROUNDDIAL (see page 134)	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

This is file RoundDial.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Round Dial
 *****************************************************************************
 * FileName:        RoundDial.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07         Version 1.0 release
 * 03/12/09         Added Object Header for Double Buffering Support
 * 11/15/10         Added new structure member for KEYBOARD support
 ******************************************************************************/
#ifndef _ROUNDDIAL_H
    #define _ROUNDDIAL_H

    #include "GenericTypeDefs.h"
```

```
    #include "GraphicsConfig.h"
    #include "Graphics/GOL.h"
/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define RDIA_DISABLED   0x0002  // Bit for disabled state.
    #define RDIA_ROT_CW     0x0004  // Bit for rotate clockwise state.
    #define RDIA_ROT_CCW    0x0008  // Bit for rotate counter clockwise state.
    #define RDIA_DRAW       0x4000  // Bit to indicate object must be redrawn.
    #define RDIA_HIDE       0x8000  // Bit to indicate object must be removed from screen.
#ifdef USE_KEYBOARD
    #define RDIA_QUADRANT_POSITIONS 6
    #define RDIA_MAX_POSITIONS      24
    extern SHORT    _cosine[RDIA_QUADRANT_POSITIONS];
    SHORT           RdiaCosine(SHORT v);
    SHORT           RdiaSine(SHORT v);
#endif

/*********************************************************************
* 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.
*
*********************************************************************/
typedef struct
{
    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).

    // Minimum is always zero.
    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.
    #ifdef USE_KEYBOARD
    SHORT           vAngle;
    #endif
} ROUNDDIAL;

/*********************************************************************
* Function: ROUNDDIAL  *RdiaCreate( WORD ID, SHORT x, SHORT y, SHORT radius,
*                       WORD state, SHORT res, SHORT value, SHORT max,
*                       GOL_SCHEME *pScheme);
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        x - Location of the center of the dial in the x coordinate.
*        y - Location of the center of the dial in the y coordinate.
*        radius - Defines the radius of the dial.
*        state - Sets the initial state of the object.
*        res - Sets the resolution of the dial when rotating clockwise or
*              counter clockwise.
*        value - Sets the initial value of the dial.
*        max - Sets the maximum value of the dial.
*        pScheme - Pointer to the style scheme used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   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;
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
ROUNDDIAL   *RdiaCreate
            (
                WORD        ID,
                SHORT       x,
                SHORT       y,
                SHORT       radius,
                WORD        state,
                SHORT       res,
                SHORT       value,
                SHORT       max,
                GOL_SCHEME  *pScheme
            );

/*********************************************************************
* Function: RdiaTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pDia  - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   <CODE>
*   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_ROUNDIAL:
*                       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;
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD        RdiaTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: RdiaMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       Translated Message      Input Source    Set/Clear State Bit     Description
*       ##################      ############    ######                  ###########
*       RD_MSG_CLOCKWISE        Touch Screen    Set RDIA_ROT_CW,        Dial will be
redrawn with clockwise update.
*                                                Set RDIA_DRAW
*       RD_MSG_CTR_CLOCKWISE    Touch Screen    Set RDIA_ROT_CCW,       Dial will be
redrawn with counter clockwise update.
*                                                Set RDIA_DRAW
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message
*        pDia          - The pointer to the object whose state will be modified
*        pMsg          - The pointer to the GOL message
*
* Output: none
*
* Example:
*   See RdiaTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
void        RdiaMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Macros:  RdiaGetVal(pDia)
*
* Overview: Returns the current dial value. Value is always
*           in the 0-max range inclusive.
*
* PreCondition: none
*
* Input: pDia - Pointer to the object.
*
* Output: Returns the current value of the dial.
*
```

```
* Example:
*   <CODE>
*   WORD currVal;
*   ROUNDDIAL *pDia;
*
*       // assuming pDia is initialized to an existing dial Object
*       currVal = RdiaGetVal(pDia);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define RdiaGetVal(pDia)    (pDia)->value

/*********************************************************************
* Macros:  RdiaSetVal(pDia, newVal)
*
* Overview: Sets the value to the given new value. Value set must be in 0-max
*           range inclusive.
*
* PreCondition: none
*
* Input: pDia - Pointer to the object.
*        newVal - New dial value.
*
* Output: none
*
* Example:
*   <CODE>
*   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);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define RdiaSetVal(pDia, newVal)    (pDia)->value = newVal

/*********************************************************************
* Macros:  RdiaIncVal(pDia)
*
* Overview: Used to directly increment the value. The delta
*           change used is the resolution setting (res).
*
* PreCondition: none
*
* Input: pDia - Pointer to the object.
*
* Output: none
*
* Example:
*   <CODE>
*   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);
*   </CODE>
*
* Side Effects: none
*
```

```
*********************************************************************/
    #define RdiaIncVal(pDia)    RdiaSetVal(pDia, (pDia->val + pDia->res))

/*********************************************************************
* Macros:  RdiaDecVal(pDia)
*
* Overview: Used to directly decrement the value. The delta
*           change used is the resolution setting (res).
*
* PreCondition: none
*
* Input: pDia - Pointer to the object.
*
* Output: none
*
* Example:
*   Refer to RdiaIncVal() example.
*
* Side Effects: none
*
*********************************************************************/
    #define RdiaDecVal(pDia)    RdiaSetVal(pDia, (pDia->pos - pDia->res))

/*********************************************************************
* Function: WORD RdiaDraw(void *pObj)
*
* 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) postion 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pDia - Pointer to the object
*
* Output: Returns the status of the drawing
*         - 1 - If the rendering was completed and
*         - 0 - If the rendering is not yet finished.
*
* Side Effects: none
*
*********************************************************************/
WORD RdiaDraw(void *pObj);
#endif // _ROUNDDIAL_H
```

14.1.32 RadioButton.c

This is file RadioButton.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Radio Button
 *****************************************************************************
 * FileName:        RadioButton.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 04/20/11     Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 ************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_RADIOBUTTON

// This pointer is used to create linked list of radio buttons for the group
RADIOBUTTON *_pListButtons = NULL;

/*******************************************************************
* Function: RADIOBUTTON  *RbCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                                  SHORT bottom, WORD state, XCHAR *pText, GOL_SCHEME *pScheme)
*
* Overview: creates the radio button
*
*******************************************************************/
RADIOBUTTON *RbCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    RADIOBUTTON *pRb = NULL;
    RADIOBUTTON *pointer;

    pRb = (RADIOBUTTON *)GFX_malloc(sizeof(RADIOBUTTON));
    if(pRb == NULL)
        return (pRb);

    pRb->hdr.ID = ID;
    pRb->hdr.pNxtObj = NULL;
    pRb->hdr.type = OBJ_RADIOBUTTON;
    pRb->hdr.left = left;
    pRb->hdr.top = top;
    pRb->hdr.right = right;
    pRb->hdr.bottom = bottom;
    pRb->pText = pText;
    pRb->pNext = NULL;                      // last radio button in the list
    pRb->hdr.state = state;
```

```
    pRb->hdr.DrawObj = RbDraw;              // draw function
    pRb->hdr.MsgObj = RbTranslateMsg;       // message function
    pRb->hdr.MsgDefaultObj = RbMsgDefault;  // default message function
    pRb->hdr.FreeObj = NULL;                // free function

    if(GetState(pRb, RB_GROUP))
    {

        // If it's first button in the group start new button's list
        _pListButtons = pRb;

        // Attach the button to the list
        pRb->pHead = (OBJ_HEADER *)_pListButtons;
    }
    else
    {

        // Attach the button to the list
        pRb->pHead = (OBJ_HEADER *)_pListButtons;
        pointer = _pListButtons;
        while(pointer->pNext != NULL)
        {
            pointer = (RADIOBUTTON *)pointer->pNext;
        }

        pointer->pNext = (OBJ_HEADER *)pRb;
    }

    // Set the style scheme to be used
    if(pScheme == NULL)
        pRb->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pRb->hdr.pGolScheme = pScheme;

    // Set the text height
    pRb->textHeight = 0;
    if(pText != NULL)
    {
        pRb->textHeight = GetTextHeight(pRb->hdr.pGolScheme->pFont);
    }

    GOLAddObject((OBJ_HEADER *)pRb);

    return (pRb);
}

/*********************************************************************
* Function: void RbSetCheck(RADIOBUTTON *pRb, WORD ID)
*
* Input: pRb - the pointer to any radio button in the group
*        ID - ID of button to be checked
*
* Output: none
*
* Overview: sets radio button to checked state, and marks states of group
*           radio buttons to be redrawn
*
*********************************************************************/
void RbSetCheck(RADIOBUTTON *pRb, WORD ID)
{
    RADIOBUTTON *pointer;

    pointer = (RADIOBUTTON *)pRb->pHead;

    while(pointer != NULL)
    {
        if(pointer->hdr.ID == ID)
        {
            SetState(pointer, RB_CHECKED | RB_DRAW_CHECK);  // set check and redraw
        }
        else
        {
```

```
            ClrState(pointer, RB_CHECKED);                  // reset checked
            SetState(pointer, RB_DRAW_CHECK);               // redraw
        }

        pointer = (RADIOBUTTON *)pointer->pNext;
    }
}

/*****************************************************************
* Function: WORD RbGetCheck(RADIOBUTTON *pRb)
*
* Input: pRb - the pointer to any radio button in the group
*
* Output: ID of checked button, -1 if there are no checked buttons
*
* Overview: gets ID of checked radio button
*
*****************************************************************/
WORD RbGetCheck(RADIOBUTTON *pRb)
{
    RADIOBUTTON *pointer;

    pointer = (RADIOBUTTON *)pRb->pHead;

    while(pointer != NULL)
    {
        if(GetState(pointer, RB_CHECKED))
        {
            return (pointer->hdr.ID);
        }

        pointer = (RADIOBUTTON *)pointer->pNext;
    }

    return (-1);
}

/*****************************************************************
* Function: RbSetText(RADIOBUTTON *pRb, XCHAR *pText)
*
* Input: pRb - the pointer to the radio button
*        pText - pointer to the text
*
* Output: none
*
* Overview: sets text
*
*****************************************************************/
void RbSetText(RADIOBUTTON *pRb, XCHAR *pText)
{
    pRb->pText = pText;
    pRb->textHeight = GetTextHeight((BYTE *)pRb->hdr.pGolScheme->pFont);
}

/*****************************************************************
* Function: RbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: changes the state of the radio button by default
*
*****************************************************************/
void RbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    RADIOBUTTON *pointer;
    RADIOBUTTON *pRb;

    pRb = (RADIOBUTTON *)pObj;

        #ifdef USE_FOCUS
            #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
        if(!GetState(pRb, RB_FOCUSED))
```

```
        {
            GOLSetFocus((OBJ_HEADER *)pRb);
        }
    }

            #endif
        #endif
    if(translatedMsg == RB_MSG_CHECKED)
    {

        // Uncheck radio buttons in the group
        pointer = (RADIOBUTTON *)pRb->pHead;

        while(pointer != NULL)
        {
            if(GetState(pointer, RB_CHECKED))
            {
                ClrState(pointer, RB_CHECKED);      // reset check
                SetState(pointer, RB_DRAW_CHECK);   // redraw
            }

            pointer = (RADIOBUTTON *)pointer->pNext;
        }

        SetState(pRb, RB_CHECKED | RB_DRAW_CHECK);  // set check and redraw
    }
}

/*********************************************************************
* Function: WORD RbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Overview: translates the GOL message for the radio button
*
*********************************************************************/
WORD RbTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    RADIOBUTTON *pRb;

    pRb = (RADIOBUTTON *)pObj;

    // Evaluate if the message is for the radio button
    // Check if disabled first
    if(GetState(pRb, RB_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
        if(pMsg->uiEvent == EVENT_PRESS)
        {

            // Check if it falls in the radio button borders
            if
            (
                (pRb->hdr.left < pMsg->param1) &&
                (pRb->hdr.right > pMsg->param1) &&
                (pRb->hdr.top < pMsg->param2) &&
                (pRb->hdr.bottom > pMsg->param2)
            )
            {
                if(!GetState(pRb, RB_CHECKED))
                    return (RB_MSG_CHECKED);
            }
        }

        return (OBJ_MSG_INVALID);
    }

        #endif
        #ifdef USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD)
```

```
    {
        if((WORD)pMsg->param1 == pRb->hdr.ID)
        {
            if(pMsg->uiEvent == EVENT_KEYSCAN)
            {
                if((pMsg->param2 == SCAN_SPACE_PRESSED) || (pMsg->param2 ==
SCAN_CR_PRESSED))
                {
                    if(!GetState(pRb, RB_CHECKED))
                        return (RB_MSG_CHECKED);
                }
            }
        }

        return (OBJ_MSG_INVALID);
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*****************************************************************
* Function: WORD RbDraw(void *pObj)
*
* Output: returns the status of the drawing
*         0 - not completed
*         1 - done
*
* Overview: draws radio button
*
*****************************************************************/
WORD RbDraw(void *pObj)
{
    typedef enum
    {
        REMOVE,
        DRAW_BUTTON0,
        DRAW_BUTTON1,
        DRAW_TEXT,
        DRAW_TEXT_RUN,
        DRAW_CHECK,
        DRAW_CHECK_RUN,
        DRAW_FOC
    } RB_DRAW_STATES;

        #define SIN45   46341

    static RB_DRAW_STATES   state = REMOVE;
    SHORT                   checkIndent;
    static SHORT            radius;
    static SHORT            x, y;
    static DWORD_VAL        temp;

    WORD                    faceClr;
    RADIOBUTTON *pRb;

    pRb = (RADIOBUTTON *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case REMOVE:
            if(GetState(pRb, (RB_HIDE | RB_DRAW)))
            {
                SetColor(pRb->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pRb->hdr.left, pRb->hdr.top, pRb->hdr.right, pRb->hdr.bottom))
                    return (0);
            }

            if(GetState(pRb, RB_HIDE))
```

```
            return (1);

        radius = ((pRb->hdr.bottom - pRb->hdr.top) >> 1) - RB_INDENT;
        x = pRb->hdr.left + ((pRb->hdr.bottom - pRb->hdr.top) >> 1) + RB_INDENT;
        y = (pRb->hdr.bottom + pRb->hdr.top) >> 1;
        temp.Val = SIN45;

        if(GetState(pRb, RB_DRAW))
        {
            state = DRAW_BUTTON0;
        }
        else
        {
            state = DRAW_CHECK;
            goto rb_draw_check;
        }

    case DRAW_BUTTON0:
        if(!GetState(pRb, RB_DISABLED))
        {
            faceClr = pRb->hdr.pGolScheme->Color0;
        }
        else
        {
            faceClr = pRb->hdr.pGolScheme->ColorDisabled;
        }

        GOLPanelDraw
        (
            x,
            y,
            x,
            y,
            radius,
            faceClr,
            pRb->hdr.pGolScheme->EmbossDkColor,
            pRb->hdr.pGolScheme->EmbossLtColor,
            NULL,
            GOL_EMBOSS_SIZE
        );
        state = DRAW_BUTTON1;

    case DRAW_BUTTON1:
        if(!GOLPanelDrawTsk())
        {
            return (0);
        }

        state = DRAW_TEXT;

    case DRAW_TEXT:
        if(pRb->pText != NULL)
        {
            SetFont(pRb->hdr.pGolScheme->pFont);

            if(GetState(pRb, RB_DISABLED))
            {
                SetColor(pRb->hdr.pGolScheme->TextColorDisabled);
            }
            else
            {
                SetColor(pRb->hdr.pGolScheme->TextColor0);
            }

            MoveTo
            (
                pRb->hdr.left + pRb->hdr.bottom - pRb->hdr.top + RB_INDENT,
                (pRb->hdr.bottom + pRb->hdr.top - pRb->textHeight) >> 1
            );

            state = DRAW_TEXT_RUN;
```

```
            case DRAW_TEXT_RUN:
                if(!OutText(pRb->pText))
                    return (0);
            }

            state = DRAW_CHECK;

    rb_draw_check:

        case DRAW_CHECK:
            if(GetState(pRb, RB_CHECKED))
            {
                if(GetState(pRb, RB_DISABLED))
                {
                    SetColor(pRb->hdr.pGolScheme->TextColorDisabled);
                }
                else
                {
                        #if (COLOR_DEPTH == 1)
                    SetColor(BLACK);
                        #else
                    SetColor(pRb->hdr.pGolScheme->TextColor0);
                        #endif
                }
            }
            else
            {
                if(GetState(pRb, RB_DISABLED))
                {
                    SetColor(pRb->hdr.pGolScheme->ColorDisabled);
                }
                else
                {
                        #if (COLOR_DEPTH == 1)
                    SetColor(WHITE);
                        #else
                    SetColor(pRb->hdr.pGolScheme->Color0);
                        #endif
                }
            }

            state = DRAW_CHECK_RUN;

        case DRAW_CHECK_RUN:
            checkIndent = (pRb->hdr.bottom - pRb->hdr.top) >> 2;
            if(!FillCircle(x, y, radius - checkIndent))
                return (0);

            state = DRAW_FOC;

        case DRAW_FOC:
            if(GetState(pRb, RB_DRAW | RB_DRAW_FOCUS))
            {
                if(IsDeviceBusy())
                    return (0);

                if(GetState(pRb, RB_FOCUSED))
                {
                    SetColor(pRb->hdr.pGolScheme->TextColor0);
                }
                else
                {
                    SetColor(pRb->hdr.pGolScheme->CommonBkColor);
                }

                SetLineType(FOCUS_LINE);
                if(!Rectangle(pRb->hdr.left, pRb->hdr.top, pRb->hdr.right, pRb->hdr.bottom))
                    return (0);
                SetLineType(SOLID_LINE);
            }

            state = REMOVE;
```

```
            return (1);
        }

        return (1);
    }

    #endif // USE_RADIOBUTTON
```

14.1.33 RadioButton.h

Functions

	Name	Description
	RbCreate (see page 230)	This function creates a RADIOBUTTON (see page 237) 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 (see page 231)	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 (see page 232)	This function returns the ID of the currently checked Radio Button (see page 65) in the group.
	RbMsgDefault (see page 235)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	RbSetCheck (see page 233)	This function sets the Radio Button (see page 65) with the given ID to its checked state.
	RbSetText (see page 234)	This function sets the string used for the object.
	RbTranslateMsg (see page 236)	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
RB_CHECKED (see page 228)	Bit to indicate Radio Button (see page 65) is checked.
RB_DISABLED (see page 228)	Bit for disabled state.
RB_DRAW (see page 228)	Bit to indicate whole Radio Button (see page 65) must be redrawn.
RB_DRAW_CHECK (see page 229)	Bit to indicate check mark should be redrawn.
RB_DRAW_FOCUS (see page 229)	Bit to indicate focus must be redrawn.
RB_FOCUSED (see page 229)	Bit for focused state.
RB_GROUP (see page 229)	Bit to indicate the first Radio Button (see page 65) in the group.
RB_HIDE (see page 229)	Bit to indicate that button must be removed from screen.
RbGetText (see page 233)	This macro returns the address of the current text string used for the object.

Structures

Name	Description
RADIOBUTTON (see page 237)	the structure contains data for the Radio Button (see page 65)

Description

This is file RadioButton.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
```

```
 *  GOL Layer
 *  Radio Button
 ****************************************************************************
 * FileName:        RadioButton.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 ****************************************************************************/
#ifndef _RADIOBUTTON_H
    #define _RADIOBUTTON_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

    #define RB_INDENT   2           // Indent for the text from title bar border

/*******************************************************************
* Object States Definition:
*******************************************************************/
    #define RB_FOCUSED      0x0001  // Bit for focused state.
    #define RB_DISABLED     0x0002  // Bit for disabled state.
    #define RB_CHECKED      0x0004  // Bit to indicate Radio Button is checked.
    #define RB_GROUP        0x0008  // Bit to indicate the first Radio Button in the group.
    #define RB_HIDE         0x8000  // Bit to indicate that button must be removed from
screen.
    #define RB_DRAW_FOCUS   0x2000  // Bit to indicate focus must be redrawn.
    #define RB_DRAW_CHECK   0x1000  // Bit to indicate check mark should be redrawn.
    #define RB_DRAW         0x4000  // Bit to indicate whole Radio Button must be redrawn.

/*****************************************************************************
* Overview: the structure contains data for the Radio Button
*****************************************************************************/
typedef struct
{
    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
} RADIOBUTTON;

/*******************************************************************
```

```
* Macros:  RbGetText(pRb)
*
* Overview: This macro returns the address of the current
*           text string used for the object.
*
* PreCondition: none
*
* Input: pRb - Pointer to the object
*
* Output: Returns pointer to the text string being used.
*
* Side Effects: none
*
*********************************************************************/
    #define RbGetText(pRb)  pRb->pText

/*********************************************************************
* Function: RbSetText(RADIOBUTTON *pRb, XCHAR *pText)
*
* Overview: This function sets the string used for the object.
*
* PreCondition: none
*
* Input: pRb - The pointer to the object whose text will be modified
*        pText - Pointer to the text that will be used
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void        RbSetText(RADIOBUTTON *pRb, XCHAR *pText);

/*********************************************************************
* Function: void RbSetCheck(RADIOBUTTON *pRb, WORD ID)
*
* Overview: This function sets the Radio Button with the given ID
*           to its checked state.
*
* PreCondition: none
*
* Input: pRb - Pointer to the Radio Button in the group.
*        ID - ID of the object to be checked.
*
* Output: none
*
* Example:
*   See RbGetCheck() example.
*
* Side Effects: none
*
*********************************************************************/
void        RbSetCheck(RADIOBUTTON *pRb, WORD ID);

/*********************************************************************
* Function: WORD RbGetCheck(RADIOBUTTON *pRb)
*
* Overview: This function returns the ID of the currently
*           checked Radio Button in the group.
*
* PreCondition: none
*
* Input: pRb - Pointer to the Radio Button in the group.
*
* Output: Returns the ID of the selected button in the group.
*         It returns -1 if there is no object checked.
*
* Example:
*   <CODE>
*   GOL_SCHEME *pScheme;
*   RADIOBUTTON *pRb[3];
*   SHORT ID;
```

```
*
*       pScheme = GOLCreateScheme();
*       pRb[0] = RbCreate(ID_RADIOBUTTON1,              // ID
*                         255,40,310,80,                // dimension
*                         RB_DRAW|RB_GROUP|RB_CHECKED,  // will be dislayed and
*                                                       // focused after creation
*                                                       // first button in the group
*                         "RB1",                        // text
*                         pScheme);                     // scheme used
*
*       pRb[1] = RbCreate(ID_RADIOBUTTON2,              // ID
*                         255,85,310,125,               // dimension
*                         RB_DRAW,                      // will be dislayed and
*                                                       // checked after creation
*                         "RB2",                        // text
*                         pScheme);                     // scheme used
*
*       pRb[2] = RbCreate(ID_RADIOBUTTON3,              // ID
*                         255,130,310,170,              // dimension
*                         RB_DRAW,                      // will be dislayed and
*                                                       // disabled after creation
*                         "RB3",                        // text
*                         pScheme);                     // 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);
*       }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WORD        RbGetCheck(RADIOBUTTON *pRb);

/*********************************************************************
* Function: RADIOBUTTON  *RbCreate( WORD ID, SHORT left, SHORT top,
*                                   SHORT right, SHORT bottom, WORD state,
*                                   XCHAR *pText, GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object
*        bottom - Bottom most position of the object
*        state - Sets the initial state of the object
*        pText – The pointer to the text used for the Radio Button.
*        pScheme - Pointer to the style scheme used.
*
* Output: Returns the pointer to the object created
*
* Example:
*   <CODE>
*   GOL_SCHEME *pScheme;
*   RADIOBUTTON *pRb[3];
*
```

```
*       pScheme = GOLCreateScheme();
*       pRb[0] = RbCreate(ID_RADIOBUTTON1,              // ID
*                         255,40,310,80,                // dimension
*                         RB_DRAW|RB_GROUP|RB_CHECKED,  // will be dislayed and
*                                                       // focused after creation
*                                                       // first button in the group
*                         "RB1",                        // text
*                         pScheme);                     // scheme used
*
*       pRb[1] = RbCreate(ID_RADIOBUTTON2,              // ID
*                         255,85,310,125,               // dimension
*                         RB_DRAW,                      // will be dislayed and
*                                                       // checked after creation
*                         "RB2",                        // text
*                         pScheme);                     // scheme used
*
*       pRb[2] = RbCreate(ID_RADIOBUTTON3,              // ID
*                         255,130,310,170,              // dimension
*                         RB_DRAW,                      // will be dislayed and
*                                                       // disabled after creation
*                         "RB3",                        // text
*                         pScheme);                     // scheme used
*
*       while(!RbDraw(pRb[0]));                         // draw the objects
*       while(!RbDraw(pRb[1]));
*       while(!RbDraw(pRb[2]));
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
RADIOBUTTON *RbCreate
            (
                WORD        ID,
                SHORT       left,
                SHORT       top,
                SHORT       right,
                SHORT       bottom,
                WORD        state,
                XCHAR       *pText,
                GOL_SCHEME  *pScheme
            );

/*********************************************************************
* Function: WORD RbTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pRb   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD        RbTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: RbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       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
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message
*        pRb           - The pointer to the object whose state will be modified
*        pMsg          - The pointer to the GOL message
*
* Output: none
*
* Example:
*   See BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
void        RbMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD RbDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pB - Pointer to the object to be rendered.
*
* Output: 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.
*
* Example:
*   See RbCreate() example.
*
* Side Effects: none
*
*********************************************************************/
WORD RbDraw(void *pObj);
```

```
#endif // _RADIOBUTTON_H
```

14.1.34 Slider.c

This is file Slider.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Slider
 *****************************************************************************
 * FileName:        Slider.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 04/29/10     - Added OBJ_MSG_PASSIVE message to detect event on the slider
 *                but with no action.
 *              - fixed the swapped message translation in the
 *                SldTranslateMsg() when keyboard event is detected.
 * 02/23/11     Removed references to BLACK and WHITE colors. Replaced
 *              with emboss dark and light colors. These are used
 *              in rendering the thumb path.
 * 04/20/11     Fixed KEYBOARD bug on object ID and GOL_MSG param1 comparison.
 *****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_SLIDER

/* Internal Functions */
SHORT   SldSetThumbSize(SLIDER *pSld, SHORT high, SHORT low);
void    SldGetMinMaxPos(SLIDER *pSld, WORD *minPos, WORD *maxPos);
WORD    SldGetWidth(SLIDER *pSld);
WORD    SldGetHeight(SLIDER *pSld);

/*****************************************************************
* Function: SLIDER *SldCreate(WORD ID, SHORT left, SHORT top, SHORT right,
```

```
*                               SHORT bottom, WORD state, SHORT range,
*                               SHORT page, SHORT pos, GOL_SCHEME *pScheme)
*
* Notes: Creates a SLIDER object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
*        is returned. If not successful, NULL is returned.
*
*******************************************************************/
SLIDER *SldCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    WORD        range,
    WORD        page,
    WORD        pos,
    GOL_SCHEME  *pScheme
)
{
    SLIDER  *pSld = NULL;

    pSld = (SLIDER *)GFX_malloc(sizeof(SLIDER));
    if(pSld == NULL)
        return (pSld);

    pSld->hdr.ID        = ID;                           // unique id assigned for
referencing
    pSld->hdr.pNxtObj   = NULL;
    pSld->hdr.type      = OBJ_SLIDER;                   // set object type
    pSld->hdr.left      = left;                         // left and right should be equal
when oriented vertically
    pSld->hdr.top       = top;                          // top and bottom should be equal
when oriented horizontally
    pSld->hdr.right     = right;
    pSld->hdr.bottom    = bottom;
    pSld->hdr.state     = state;
    pSld->hdr.DrawObj      = SldDraw;                   // draw function
    pSld->hdr.MsgObj         = SldTranslateMsg;         // message function
    pSld->hdr.MsgDefaultObj  = SldMsgDefault;           // default message function
    pSld->hdr.FreeObj  = NULL;                          // default free function

    // Parameters in the user defined range system (pos, page and range)
    pSld->range = range;                             // range of the slider movement (always
measured from 0 to range)

    // 0 refers to pSld->minPos and
    // range refers to pSld->maxpos where: minPos and maxPos are
    // the coordinate equivalent of 0 and range value
    pSld->page = page;                               // set the resolution
    pSld->pos = pos;                                 // set the initial position

    // calculate the thumb width and height
    pSld->thWidth = SldGetWidth(pSld);
    pSld->thHeight = SldGetHeight(pSld);

    // Set the color scheme to be used
    if(pScheme == NULL)
        pSld->hdr.pGolScheme = _pDefaultGolScheme;  // use default scheme
    else
        pSld->hdr.pGolScheme = (GOL_SCHEME *)pScheme;   // user defined scheme
    GOLAddObject((OBJ_HEADER *)pSld);                   // add the new object to the
current list
    return (pSld);
}

/*******************************************************************
* Function: SHORT SldSetThumbSize(SLIDER *pSld, SHORT high, SHORT low)
*
* Notes: An INTERNAL function used to compute for the width or
```

```
*        height of the thumb. This function is created to save
*        code size. This function is called only to dynamically
*        compute for the thumb size. Used only when slider is
*        type Scrollbar. Parameter are defined as:
*        pSld - pointer to the object
*        high - higher value to be used
*        low  - lower value to be used
*
*********************************************************************/
SHORT SldSetThumbSize(SLIDER *pSld, SHORT high, SHORT low)
{
    WORD    temp;

    temp = (pSld->range / pSld->page);
    temp = (high - low) / temp;

    // when size is less than half of emboss size, set the
    // size to half the emboss size. This is to make sure
    // thumb will always have a size.
    if(temp < (GOL_EMBOSS_SIZE << 1))
        temp = (GOL_EMBOSS_SIZE << 1);

    return (SHORT) temp;
}

/*********************************************************************
* Function: WORD SldGetWidth(SLIDER *pSld)
*
* Notes: An INTERNAL function that computes for the width
*        of the thumb. This function is created to save
*        code size. This function is called only to dynamically
*        compute for the thumb size.
*
*********************************************************************/
WORD SldGetWidth(SLIDER *pSld)
{
    WORD    temp;

    /*
        Calculating the width is dependent on the mode type.
        If type Scrollbar, width is dependent on the ratio of the
        page/range = width/max-min (see SetThumbSize())
        if type is Slider, width is dependent on height*3/8

        When horizontal width is dynamic, height is contant.

    */
    if(GetState(pSld, SLD_VERTICAL))
    {
        temp = pSld->hdr.right - pSld->hdr.left;
        if(GetState(pSld, SLD_SCROLLBAR))
        {
            temp = temp - (GOL_EMBOSS_SIZE << 1);
        }
        else
        {
            temp = temp - (GOL_EMBOSS_SIZE << 1) - 2;
        }
    }
    else
    {
        if(GetState(pSld, SLD_SCROLLBAR))
        {
            temp = SldSetThumbSize(pSld, pSld->hdr.right, pSld->hdr.left);
        }
        else
        {
            temp = (((pSld->hdr.bottom - pSld->hdr.top) - (GOL_EMBOSS_SIZE << 1) - 2) * 3)
>> 3;
        }
    }
```

```
    // to avoid calculations of dividing by two, we store half the width value
    return (temp >> 1);
}

/*********************************************************************
* Function: WORD SldGetHeight(SLIDER *pSld)
*
* Notes: An INTERNAL function that computes for the height
*        of the thumb. This function is created to save
*        code size. This function is called only to dynamically
*        compute for the thumb size.
*
*********************************************************************/
WORD SldGetHeight(SLIDER *pSld)
{
    WORD    temp;

    /*
        Calculating the height is dependent on the mode type.
        If type Scrollbar, width is dependent on the ratio of the
        page/range = width/max-min (see SetThumbSize())
        if type is Slider, width is dependent on width*3/8

        When vertical height is dynamic, width is contant.
    */
    if(GetState(pSld, SLD_VERTICAL))
    {
        if(GetState(pSld, SLD_SCROLLBAR))
        {
            temp = SldSetThumbSize(pSld, pSld->hdr.bottom, pSld->hdr.top);
        }
        else
        {
            temp = (((pSld->hdr.right - pSld->hdr.left) - (GOL_EMBOSS_SIZE << 1) - 2) * 3)
>> 3;
        }
    }
    else
    {
        temp = pSld->hdr.bottom - pSld->hdr.top;
        if(GetState(pSld, SLD_SCROLLBAR))
        {
            temp = temp - (GOL_EMBOSS_SIZE << 1);
        }
        else
        {
            temp = temp - (GOL_EMBOSS_SIZE << 1) - 2;
        }
    }

    // to avoid calculations of dividing by two, we store half the height value
    return (temp >> 1);
}

/*********************************************************************
* Function: void SldGetMinMaxPos(SLIDER *pSld, WORD *min, WORD *max)
*
* Notes:  An INTERNAL function that computes for the minimum
*         and maximum pixel position in the screen. This function is
*         created to save code size. Used to define the minimum
*         & maximum position of the thumb when sliding. Parameters
*         used are defined as:
*         pSld - pointer to the object
*         min  - pointer to the minimum variable
*         max  - pointer to the maximum variable
*
*********************************************************************/
void SldGetMinMaxPos(SLIDER *pSld, WORD *min, WORD *max)
{
    WORD    temp;

    // calculate maximum and minimum position
```

```
    if(GetState(pSld, SLD_VERTICAL))
    {
        temp = pSld->thHeight + GOL_EMBOSS_SIZE;
        *min = pSld->hdr.top + temp;
        *max = pSld->hdr.bottom - temp;
    }
    else
    {
        temp = pSld->thWidth + GOL_EMBOSS_SIZE;
        *min = pSld->hdr.left + temp;
        *max = pSld->hdr.right - temp;
    }

    // for aestetics.
    if(!GetState(pSld, SLD_SCROLLBAR))
    {
        *min = *min + 2;
        *max = *max - 2;
    }
}

/*********************************************************************
* Function: void SldSetRange(SLIDER *pSld, SHORT newRange)
*
* Notes:  Sets the new range value of the slider or scrollbar.
*         Object must be redrawn after this function is called to
*         reflect the changes to the object.
*
*********************************************************************/
void SldSetRange(SLIDER *pSld, SHORT newRange)
{
    WORD        newPos;
    DWORD_VAL   dTemp;

    // this checks for limits of the range (minimum is 2)
    if(newRange <= 2)
        newRange = 2;

    if((WORD) newRange > (WORD) 0x7FFF)
        newRange = 0x7FFF;

    dTemp.Val = newRange * pSld->pos;
    dTemp.Val = dTemp.Val / pSld->range;

    // get new range
    newPos = dTemp.w[0];

    // set the new range
    pSld->range = newRange;

    // now check the page, adjust when necessary
    // page maximum limit is range/2, minimum is 1
    if(pSld->page > ((pSld->range) >> 1))
    {
        if(!((pSld->range) >> 1))
            pSld->page = 1;
        else
            pSld->page = (pSld->range) >> 1;
    }

    // calculate new thumb width and height
    pSld->thWidth = SldGetWidth(pSld);
    pSld->thHeight = SldGetHeight(pSld);
    SldSetPos(pSld, newPos);
}

/*********************************************************************
* Function: void SldSetPage(SLIDER *pSld, WORD newPage)
*
* Notes: Sets the new page value of the slider or scrollbar.
*        The page maximum limit is range/2, minimum is 1
*
```

```
*********************************************************************/
void SldSetPage(SLIDER *pSld, WORD newPage)
{
    if(newPage < 1)
        newPage = 1;
    else if(newPage > ((pSld->range) >> 1))
        newPage = (pSld->range) >> 1;
    pSld->page = newPage;

    // calculate new thumb width and height
    pSld->thWidth = SldGetWidth(pSld);
    pSld->thHeight = SldGetHeight(pSld);
}

/*********************************************************************
* Function: SldSetPos(SLIDER *pSld, SHORT newPos)
*
* Notes: Sets the thumb to the new position. Checking is first
*        preformed if the new position is within the range (0 to range)
*        of the slider. Object must be redrawn after this function is called to
*        reflect the changes to the object.
*
*********************************************************************/
void SldSetPos(SLIDER *pSld, SHORT newPos)
{
    WORD        minPos, maxPos, relPos;
    DWORD_VAL   dTemp;

    // get minimum and maximum positions
    SldGetMinMaxPos(pSld, &minPos, &maxPos);
    dTemp.Val = 0;

        #ifndef SLD_INVERT_VERTICAL

    // check if the new value is still in range
    if(newPos <= 0)
    {
        pSld->pos = 0;                                  // set to zero in range domain
        if(GetState(pSld, SLD_VERTICAL))
        {                                               // min and max in vertical is
inverted
            pSld->currPos = maxPos;                     // minimum position is the bottom
position in
        }                                               // coordinate domain
        else
            pSld->currPos = minPos;                     // minimum is left most position in
coordinate domain
    }
    else if(newPos >= pSld->range)
    {
        pSld->pos = pSld->range;                        // set to maximum value in range
domain
        if(GetState(pSld, SLD_VERTICAL))
        {                                               // min and max in vertical is
inverted
            pSld->currPos = minPos;                     // maximum position is the top
position in
        }                                               // coordinate domain
        else
            pSld->currPos = maxPos;                     // maximum is right most position
in coordinate domain
    }
    else
    {
        pSld->pos = newPos;                             // get new position in range domain
        dTemp.w[1] = newPos;
        dTemp.Val = dTemp.Val / pSld->range;
        dTemp.Val = (maxPos - minPos) * dTemp.Val;

        // set current position in coordinate domain
        relPos = dTemp.w[1] + minPos;
```

```
        if(GetState(pSld, SLD_VERTICAL))
        {                                                   // test if we need to transform min
and max position
            pSld->currPos = maxPos - (relPos - minPos); // min and max position is swapped
in coordinate domain
        }
        else
            pSld->currPos = relPos;                         // use position
    }

        #else

    // check if the new value is still in range
    if(newPos <= 0)
    {
        pSld->pos = 0;              // set to zero in range domain
        pSld->currPos = minPos;     // set to minimum in coordinate domain
    }
    else if(newPos >= pSld->range)
    {
        pSld->pos = pSld->range;    // set to maximum value in range domain
        pSld->currPos = maxPos;     // set to minimum in coordinate domain
    }
    else
    {
        pSld->pos = newPos;         // get new position in range domain
        dTemp.w[1] = newPos;
        dTemp.Val = dTemp.Val / pSld->range;
        dTemp.Val = (maxPos - minPos) * dTemp.Val;

        // set current position in coordinate domain
        pSld->currPos = dTemp.w[1] + minPos;
    }

        #endif // ifndef SLD_INVERT_VERTICAL
}

/*********************************************************************
* Function: void SldMsgDefault(WORD translatedMsg, void *pObj,
*                              GOL_MSG* pMsg)
*
* Notes: This the default operation to change the state of the button.
*        Called inside GOLMsg() when GOLMsgCallback() returns a 1.
*
*********************************************************************/
void SldMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    SLIDER *pSld;

    pSld = (SLIDER *)pObj;

        #ifdef USE_TOUCHSCREEN

    WORD        newPos, minPos, maxPos;
    DWORD_VAL   dTemp;

            #ifdef USE_FOCUS
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
        if(!GetState(pSld, SLD_FOCUSED))
        {
            GOLSetFocus((OBJ_HEADER *)pSld);
        }
    }

            #endif // USE_FOCUS

    // if message was passive do not do anything
    if (translatedMsg == OBJ_MSG_PASSIVE)
        return;

    // get the min and max positions
```

```
    SldGetMinMaxPos(pSld, &minPos, &maxPos);

    if(pMsg->type == TYPE_TOUCHSCREEN)
    {
        if((translatedMsg == SLD_MSG_DEC) || (translatedMsg == SLD_MSG_INC))
        {

            // newPos in this context is used in the coordinate domain
            if(!GetState(pSld, SLD_VERTICAL))
            {                               // check if Horizontal or Vertical orientation
                if(pMsg->param1 <= minPos)
                {                           // Horizontal orientation: test x position
                    newPos = minPos;        // beyond minimum, use min position
                }
                else if(pMsg->param1 >= maxPos)
                {
                    newPos = maxPos;        // beyond maximum, use max position
                }
                else
                {
                    newPos = pMsg->param1;  // within range: use x position given
                }
            }
            else
            {
                if(pMsg->param2 <= minPos)
                {                           // Vertical orientation: test y position
                    newPos = minPos;        // beyond minimum, use min position
                }
                else if(pMsg->param2 >= maxPos)
                {
                    newPos = maxPos;        // beyond maximum, use max position
                }
                else
                {
                    newPos = pMsg->param2;  // within range: use y position given
                }
            }

            if(newPos != pSld->currPos)
            {                               // check if we need to redraw thumb
                // yes redraw is needed, translate newPos into range domain
                // first get new position in range domain
                dTemp.Val = (DWORD) (newPos - minPos) * (DWORD) pSld->range;
                dTemp.Val = dTemp.Val / (maxPos - minPos);
                newPos = dTemp.w[0];

                        #ifndef SLD_INVERT_VERTICAL
                if(GetState(pSld, SLD_VERTICAL))
                {                           // check if we need to swap min and max in
vertical
                    newPos = pSld->range - newPos;  // min and max is swapped in vertical
orientation
                }

                        #endif
                SldSetPos(pSld, newPos);            // set to new position
                SetState(pSld, SLD_DRAW_THUMB);     // redraw the thumb only
            }
            else
                return;
        }
        else
            return;
    }

        #endif // USE_TOUCHSCREEN
        #ifdef USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD)
    {                           // for keyboard
        if(translatedMsg == SLD_MSG_INC)
        {
```

```
            SldIncPos(pSld);    // increment is requested
        }
        else
        {
            SldDecPos(pSld);    // decrement is requested
        }

        SetState(pSld, SLD_DRAW_THUMB); // redraw the thumb only
    }

        #endif // USE_KEYBOARD
}

/*********************************************************************
* Function: WORD SldTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*********************************************************************/
WORD SldTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    SLIDER *pSld;

    pSld = (SLIDER *)pObj;

    // Evaluate if the message is for the slider
    // Check if disabled first
    if(GetState(pSld, SLD_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls to the left or right of the center of the thumb's face
        if((pMsg->uiEvent == EVENT_PRESS) || (pMsg->uiEvent == EVENT_MOVE))
        {
            if
            (
                (pSld->hdr.left < pMsg->param1) &&
                (pSld->hdr.right > pMsg->param1) &&
                (pSld->hdr.top < pMsg->param2) &&
                (pSld->hdr.bottom > pMsg->param2)
            )
            {
                if(GetState(pSld, SLD_VERTICAL))
                {
                    if(pSld->currPos < pMsg->param2)
                        return (SLD_MSG_INC);
                    else
                        return (SLD_MSG_DEC);
                }
                else
                {
                    if(pSld->currPos < pMsg->param1)
                        return (SLD_MSG_INC);
                    else
                        return (SLD_MSG_DEC);
                }
            }
        }   // end of if((pMsg->uiEvent == EVENT_PRESS) || (pMsg->uiEvent == EVENT_MOVE))

        // when the event is release emit OBJ_MSG_PASSIVE this can be used to
        // detect that the release event happened on the slider.
        if(pMsg->uiEvent == EVENT_RELEASE)
            return OBJ_MSG_PASSIVE;

        return (OBJ_MSG_INVALID);
    }       // end of if(pMsg->type == TYPE_TOUCHSCREEN
        #endif
```

```
        #ifdef USE_KEYBOARD
    if(pMsg->type == TYPE_KEYBOARD)
    {
        if((WORD)pMsg->param1 == pSld->hdr.ID)
        {
            if(pMsg->uiEvent == EVENT_KEYSCAN)
            {
                if((pMsg->param2 == SCAN_RIGHT_PRESSED) || (pMsg->param2 ==
SCAN_UP_PRESSED))
                {
                    return (SLD_MSG_INC);
                }

                if((pMsg->param2 == SCAN_LEFT_PRESSED) || (pMsg->param2 ==
SCAN_DOWN_PRESSED))
                {
                    return (SLD_MSG_DEC);
                }
            }
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*****************************************************************
* Function: WORD SldDraw(void *pObj)
*
* Notes: This is the state machine to draw the slider or scrollbar.
*
*****************************************************************/
WORD SldDraw(void *pObj)
{
    typedef enum
    {
        SLD_STATE_IDLE,
        SLD_STATE_PANEL,
        SLD_STATE_THUMBPATH1,
        SLD_STATE_THUMBPATH2,
        SLD_STATE_CLEARTHUMB,
        SLD_STATE_REDRAWPATH1,
        SLD_STATE_REDRAWPATH2,
        SLD_STATE_THUMB,
        SLD_STATE_THUMBPANEL,
        SLD_STATE_FOCUS
    } SLD_DRAW_STATES;

    static WORD colorTemp = 0;

    static SLD_DRAW_STATES state = SLD_STATE_IDLE;
    static WORD left, top, right, bottom;
    static WORD midPoint, thWidth, thHeight;
    static WORD minPos, maxPos;
    SLIDER *pSld;

    pSld = (SLIDER *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case SLD_STATE_IDLE:
            if(GetState(pSld, SLD_HIDE))
            {
                SetColor(pSld->hdr.pGolScheme->CommonBkColor);      // set to common BK
Color
                if(!Bar(pSld->hdr.left, pSld->hdr.top, pSld->hdr.right, pSld->hdr.bottom))
                    return (0);
                return (1);
            }
```

```
            if(!GetState(pSld, SLD_DISABLED))
            {
                colorTemp = pSld->hdr.pGolScheme->Color0;           // select enabled color
            }
            else
            {
                colorTemp = pSld->hdr.pGolScheme->ColorDisabled;    // select disabled color
            }

            SldGetMinMaxPos(pSld, &minPos, &maxPos);

            midPoint = GetState(pSld, SLD_VERTICAL) ? (pSld->hdr.left + pSld->hdr.right) >>
1 : (pSld->hdr.top + pSld->hdr.bottom) >> 1;

            // calculate the thumb width and height Actually gets the half value
            // (see calculation of width and height) SldGetWidth() and SldGetHeight()
            thWidth = pSld->thWidth;                                // gets half the width
            thHeight = pSld->thHeight;                              // gets half the height
            SetLineThickness(NORMAL_LINE);
            SetLineType(SOLID_LINE);
            if(GetState(pSld, SLD_DRAW))
            {   // draw the panel for the slider
                // modify the color setting if scroll bar mode or slider mode
                GOLPanelDraw
                (
                    pSld->hdr.left,
                    pSld->hdr.top,
                    pSld->hdr.right,
                    pSld->hdr.bottom,
                    0,
                    colorTemp,
                    (GetState(pSld, SLD_SCROLLBAR)) ? pSld->hdr.pGolScheme->EmbossDkColor :
pSld->hdr.pGolScheme->EmbossLtColor,
                    (GetState(pSld, SLD_SCROLLBAR)) ? pSld->hdr.pGolScheme->EmbossLtColor :
pSld->hdr.pGolScheme->EmbossDkColor,
                    NULL,
                    GOL_EMBOSS_SIZE
                );

                // initialize current and previous position
                SldSetPos(pSld, pSld->pos);
                pSld->prevPos = pSld->currPos;

                state = SLD_STATE_PANEL;
            }
            else
            {   // we do not need to draw the whole object
                state = SLD_STATE_CLEARTHUMB;   // go to thumb drawing
                goto sld_state_clearthumb;
            }

        case SLD_STATE_PANEL:
            if(!GOLPanelDrawTsk())              // draw the panel of the slider
                return (0);
            if(GetState(pSld, SLD_SCROLLBAR))
            {                               // check if slider or scroll bar
                state = SLD_STATE_THUMB;    // scrollbar: go directly to thumb drawing
                goto sld_state_thumb;       // thumb path is not drawn in scrollbar
            }
            else
            {
                state = SLD_STATE_THUMBPATH1;   // slider: draw thumb path next
            }

        case SLD_STATE_THUMBPATH1:
            SetColor(pSld->hdr.pGolScheme->EmbossDkColor);
            if(!GetState(pSld, SLD_VERTICAL))
            {
                if(!Line(minPos, midPoint, maxPos, midPoint))
                    return (0);
            }
```

```
            else
            {
                if(!Line(midPoint, minPos, midPoint, maxPos))
                    return (0);
            }

            state = SLD_STATE_THUMBPATH2;

        case SLD_STATE_THUMBPATH2:
            SetColor(pSld->hdr.pGolScheme->EmbossLtColor);
            if(!GetState(pSld, SLD_VERTICAL))
            {
                if(!Line(minPos, midPoint + 1, maxPos, midPoint + 1))
                    return (0);
            }
            else
            {
                if(!Line(midPoint + 1, minPos, midPoint + 1, maxPos))
                    return (0);
            }

            if(GetState(pSld, SLD_DRAW))
            {                               // if drawing the whole slider
                state = SLD_STATE_THUMB;    // go straight to drawing the thumb
                goto sld_state_thumb;
            }
            else
                // if just drawing the thumb
                state = SLD_STATE_CLEARTHUMB;   // go to state to remove current position

        case SLD_STATE_CLEARTHUMB:              // this removes the current thumb
            sld_state_clearthumb : if(IsDeviceBusy()) return (0);

            if(!GetState(pSld, SLD_DRAW_THUMB))
            {                               // SLD_DRAW_THUMB is only set when
                state = SLD_STATE_FOCUS;    // object type is SLIDER
                goto sld_state_focus;
            }

            SetColor(colorTemp);

            // Remove the current thumb by drawing a bar with background color
            if(!GetState(pSld, SLD_VERTICAL))
            {
                if(!Bar(pSld->prevPos - thWidth, midPoint - thHeight, pSld->prevPos +
thWidth, midPoint + thHeight))
                    return (0);
            }
            else
            {
                if(!Bar(midPoint - thWidth, pSld->prevPos - thHeight, midPoint + thWidth,
pSld->prevPos + thHeight))
                    return (0);
            }

            if(!GetState(pSld, SLD_SCROLLBAR))
            {                               // check if slider or scroll bar
                state = SLD_STATE_REDRAWPATH1;
            }
            else
            {
                state = SLD_STATE_THUMB;    // go directly to thumb drawing
                goto sld_state_thumb;       // thumb path is not drawn in scrollbar
            }

        case SLD_STATE_REDRAWPATH1:         // redraws the lines that it covered
            SetColor(pSld->hdr.pGolScheme->EmbossDkColor);

            // Check if the redraw area exceeds the actual dimension. This will
            // adjust the redrawing area to just within the parameters
            if(!GetState(pSld, SLD_VERTICAL))
            {
```

```
            if(minPos + thWidth > pSld->prevPos)
                left = minPos;
            else
                left = pSld->prevPos - thWidth;

            if(maxPos - thWidth < pSld->prevPos)
                right = maxPos;
            else
                right = pSld->prevPos + thWidth;

            if(!Line(left, midPoint, right, midPoint))
                return (0);
        }
        else
        {
            if(minPos + thHeight > pSld->prevPos)
                top = minPos;
            else
                top = pSld->prevPos - thHeight;

            if(maxPos - thHeight < pSld->prevPos)
                bottom = maxPos;
            else
                bottom = pSld->prevPos + thHeight;

            if(!Line(midPoint, top, midPoint, bottom))
                return (0);
        }

        state = SLD_STATE_REDRAWPATH2;

    case SLD_STATE_REDRAWPATH2:
        SetColor(pSld->hdr.pGolScheme->EmbossLtColor);
        if(!GetState(pSld, SLD_VERTICAL))
        {
            if(!Line(left, midPoint + 1, right, midPoint + 1))
                return (0);
        }
        else
        {
            if(!Line(midPoint + 1, top, midPoint + 1, bottom))
                return (0);
        }

        state = SLD_STATE_THUMB;

    case SLD_STATE_THUMB:
        sld_state_thumb : if(IsDeviceBusy()) return (0);
        if(!GetState(pSld, SLD_VERTICAL))
        {                                   // Draw the slider thumb based on the
            // current position
            left = pSld->currPos - thWidth;
            top = midPoint - thHeight;
            right = pSld->currPos + thWidth;
            bottom = midPoint + thHeight;
        }
        else
        {
            left = midPoint - thWidth;
            top = pSld->currPos - thHeight;
            right = midPoint + thWidth;
            bottom = pSld->currPos + thHeight;
        }

        GOLPanelDraw
        (
            left,
            top,
            right,
            bottom,
            0,                              // set the parameters of the thumb
            colorTemp,
```

```
                pSld->hdr.pGolScheme->EmbossLtColor,
                pSld->hdr.pGolScheme->EmbossDkColor,
                NULL,
                (GOL_EMBOSS_SIZE - 1) ? GOL_EMBOSS_SIZE - 1 : 1
            );

            state = SLD_STATE_THUMBPANEL;

        case SLD_STATE_THUMBPANEL:
            if(!GOLPanelDrawTsk())          // draw the panel of the thumb
                return (0);

            pSld->prevPos = pSld->currPos;  // record the current position as previous
            if(GetState(pSld, SLD_SCROLLBAR))
            {                               // check if scroll bar focus is not used
                state = SLD_STATE_IDLE;     // go back to idle state
                return (1);
            }

            if(!GetState(pSld, SLD_DRAW_FOCUS))
            {
                state = SLD_STATE_IDLE;
                return (1);
            }

            state = SLD_STATE_FOCUS;

        case SLD_STATE_FOCUS:
            sld_state_focus : if(!GetState(pSld, SLD_SCROLLBAR))
            {                               // do not draw focus when in scroll bar mode
                SetLineType(FOCUS_LINE);
                if(GetState(pSld, SLD_FOCUSED))
                {
                    SetColor(pSld->hdr.pGolScheme->TextColor0); // draw the focus box
                }
                else
                {
                    SetColor(colorTemp);                        // remove the focus box,
colorTemp
                }

                if
                (
                    !Rectangle
                        (
                            pSld->hdr.left + GOL_EMBOSS_SIZE,
                            pSld->hdr.top + GOL_EMBOSS_SIZE,
                            pSld->hdr.right - GOL_EMBOSS_SIZE,
                            pSld->hdr.bottom - GOL_EMBOSS_SIZE
                        )
                ) return (0);

                SetLineType(SOLID_LINE);                        // reset line type
            }

            state = SLD_STATE_IDLE;                             // set state to idle
            return (1); // return as done
    }

    return (1);
}

#endif // USE_SLIDER
```

14.1.35 Slider.h

Functions

	Name	Description
	SldCreate (see page 241)	This function creates a SLIDER (see page 252) object with the parameters given. Depending on the SLD_SCROLLBAR (see page 241) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (see page 241) 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 (see page 243)	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.
	SldMsgDefault (see page 250)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	SldSetPage (see page 244)	This sets the page size of the object. Page size defines the delta change of the thumb position when incremented via SldIncPos (see page 248)() or decremented via SldDecPos (see page 249)(). Page size minimum value is 1. Maximum value is range/2.
	SldSetPos (see page 245)	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 (see page 247)	This sets the range of the thumb. If this field is changed Object must be completely redrawn to reflect the change.
	SldTranslateMsg (see page 251)	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
SLD_DISABLED (see page 239)	Bit for disabled state
SLD_DRAW (see page 240)	Bit to indicate whole slider must be redrawn
SLD_DRAW_FOCUS (see page 240)	Bit to indicate that only the focus will be redrawn
SLD_DRAW_THUMB (see page 240)	Bit to indicate that only thumb area must be redrawn
SLD_FOCUSED (see page 240)	Bit for focus state
SLD_HIDE (see page 240)	Bit to remove object from screen
SLD_SCROLLBAR (see page 241)	Bit for type usage (0 - Slider (see page 237), 1 - ScrollBar)
SLD_VERTICAL (see page 241)	Bit for orientation (0 - horizontal, 1 - vertical)
SldDecPos (see page 249)	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.
SldGetPage (see page 244)	Returns returns the current slider thumb position. Value is always in the 1 to range/2 inclusive.
SldGetPos (see page 246)	Returns returns the current position of the slider thumb.
SldGetRange (see page 247)	Returns the current slider page. Value is always in the 1-range/2 inclusive for both slider and scrollbar type.
SldIncPos (see page 248)	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.

Structures

Name	Description
SLIDER (see page 252)	Defines the parameters required for a slider/scrollbar Object. Depending on the SLD_SCROLLBAR (see page 241) state bit slider or scrollbar mode is set. If SLD_SCROLLBAR (see page 241) is set, mode is scrollbar; if not set mode is slider. For scrollbar mode, focus rectangle is not drawn.

Description

This is file Slider.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Slider
 *****************************************************************************
 * FileName:        Slider.h
 * Dependencies:    none
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07    Version 1.0 release
 * 08/12/10    Added casting to pointers in macros.
 *****************************************************************************/
#ifndef _SLIDER_H
    #define _SLIDER_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define SLD_FOCUSED     0x0001  // Bit for focus state
    #define SLD_DISABLED    0x0002  // Bit for disabled state
    #define SLD_VERTICAL    0x0004  // Bit for orientation  (0 - horizontal, 1 - vertical)
    #define SLD_SCROLLBAR   0x0010  // Bit for type usage (0 - Slider, 1 - ScrollBar)
    #define SLD_DRAW_THUMB  0x1000  // Bit to indicate that only thumb area must be redrawn
    #define SLD_DRAW_FOCUS  0x2000  // Bit to indicate that only the focus will be redrawn
    #define SLD_DRAW        0x4000  // Bit to indicate whole slider must be redrawn
```

```
    #define SLD_HIDE        0x8000  // Bit to remove object from screen

/*********************************************************************
* 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.
*
*********************************************************************/
typedef struct
{
    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

    // in range domain.
    WORD        thWidth;    // Thumb width. This is computed internally.

    // User must not change this value.
    WORD        thHeight;   // Thumb width. This is computed internally.
    // User must not change this value.
} SLIDER;

/*********************************************************************
* Function:  SldSetRange(SLIDER *pSld, SHORT newRange)
*
* Overview: This sets the range of the thumb. If this field is changed
*           Object must be completely redrawn to reflect the change.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*        newRange - Value of the new range used.
*
* Output: None.
*
* Example:
*   <CODE>
*   SLIDER *pSld;
*
*       SldSetRange(pSld, 100);
*       // to completely redraw the object when GOLDraw() is executed.
*       SetState(pSld, SLD_DRAW);
*
*   </CODE>
*
* Side Effects: Position of the thumb may change when redrawn.
*
*********************************************************************/
void    SldSetRange(SLIDER *pSld, SHORT newRange);

/*********************************************************************
* Function: void SldSetPage(SLIDER *pSld, WORD newPage)
*
* 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.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*        newPage - Value of the new page used.
*
* Output: None.
*
* Example:
```

```
*   See SldIncPos() example.
*
* Side Effects: Position of the thumb may change when redrawn.
*
*********************************************************************/
void    SldSetPage(SLIDER *pSld, WORD newPage);

/*********************************************************************
* Macros:  SldGetRange(pSld)
*
* Overview: Returns the current slider page. Value is always
*           in the 1-range/2 inclusive for both slider and
*           scrollbar type.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*
* Output: Returns the current value of the slider range.
*
* Example:
*   <CODE>
*   WORD range;
*   SLIDER *pSld;
*
*       range = SldGetRange(pSld);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define SldGetRange(pSld)   (((SLIDER*)pSld)->range)

/*********************************************************************
* Macros:  SldGetPage(pSld)
*
* Overview: Returns returns the current slider thumb position.
*           Value is always in the 1 to range/2 inclusive.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*
* Output: Returns the current value of the slider page.
*
* Example:
*   <CODE>
*   WORD page;
*   SLIDER *pSld;
*
*       page = SldGetPage(pSld);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define SldGetPage(pSld)    (((SLIDER*)pSld)->page)

/*********************************************************************
* Macros:  SldGetPos(pSld)
*
* Overview: Returns returns the current position of the slider thumb.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*
* Output: Returns the current position of the slider's thumb.
*
* Example:
*   <CODE>
*   #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"
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define SldGetPos(pSld) (((SLIDER*)pSld)->pos)

/*********************************************************************
* Function:  SldSetPos(SLIDER *pSld, SHORT newPos)
*
* 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.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*        newPos - The new position of the slider's thumb.
*
* Output: none
*
* Example:
*   <CODE>
*   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";
*       }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
void    SldSetPos(SLIDER *pSld, SHORT newPos);

/*********************************************************************
* Macros:  SldIncPos(pSld)
*
* 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.
*
* PreCondition: none
```

```
*
* Input: pSld - Pointer to the object.
*
* Output: none
*
* Example:
*   <CODE>
*   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
*       }
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #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    \
    )

/*********************************************************************
* Macros:  SldDecPos(pSld)
*
* 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.
*
* PreCondition: none
*
* Input: pSld - Pointer to the object.
*
* Output: none
*
* Example:
*   See SldIncPos() example.
*
* Side Effects: none
*
*********************************************************************/
    #define
SldDecPos(pSld)                                                                    \
    SldSetPos

\
    (

\
        pSld,

\
        (((LONG) ((SLIDER*)pSld)->pos - (LONG) ((SLIDER*)pSld)->page) >= 0)
```

```
?                       \
                         (((SLIDER*)pSld)->pos - ((SLIDER*)pSld)->page) :
0                          \
    )

/*****************************************************************
* Function: SLIDER *SldCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                             SHORT bottom, WORD state, SHORT range,
*                             SHORT page, SHORT pos, GOL_SCHEME *pScheme);
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
*        right - Right most position of the Object.
*        bottom - Bottom most position of the Object.
*        state - This defines the initial state of the Object.
*        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.
*        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.
*        pos - This defines the initial position of the thumb.
*        pScheme - The pointer to the style scheme used for the Object.
*                  Set to NULL if default style scheme is used.
*
* Output: Returns the pointer to the object created.
*
* Example:
*   <CODE>
*   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;
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
SLIDER  *SldCreate
        (
            WORD        ID,
            SHORT       left,
            SHORT       top,
            SHORT       right,
            SHORT       bottom,
            WORD        state,
            WORD        range,
            WORD        page,
            WORD        pos,
            GOL_SCHEME  *pScheme
        );

/*********************************************************************
* Function: void SldMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       Translated Message   Input Source     Set/Clear State Bit   Description
*       ##################   #############    ######                ############
*       SLD_MSG_INC          Touch Screen,    Set SLD_DRAW_THUMB    Slider will be redrawn
with thumb in the incremented position.
*                             Keyboard
*       SLD_MSG_DEC          Touch Screen,    Set SLD_DRAW_THUMB    Slider will be redrawn
with thumb in the decremented position.
*                             Keyboard
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message.
*        pSld          - The pointer to the object whose state will be modified.
*        pMsg          - The pointer to the GOL message.
*
* Output: none
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
void    SldMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
```

```
* Function: WORD SldTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pSld  - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD    SldTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD SldDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pSld - Pointer to the object to be rendered.
*
* Output: 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.
```

```
 *
 * Example:
 *   See SldCreate() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
WORD SldDraw(void *pObj);
#endif //_SLIDER_H
```

14.1.36 StaticText.c

This is file StaticText.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Static Text
 *****************************************************************************
 * FileName:        StaticText.c
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * PAT                  11/12/07    Version 1.0 release
 * PAT                  11/12/07    Fixed clipping enabling location
 *****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_STATICTEXT

/*******************************************************************
* Function: STATICTEXT  *StCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT
bottom,
*                                 WORD state , XCHAR *pText, GOL_SCHEME *pScheme)
*
* Notes: Creates a STATICTEXT object and adds it to the current active list.
*        If the creation is successful, the pointer to the created Object
```

```
*        is returned. If not successful, NULL is returned.
*
*********************************************************************/
STATICTEXT *StCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    STATICTEXT  *pSt = NULL;

    pSt = (STATICTEXT *)GFX_malloc(sizeof(STATICTEXT));
    if(pSt == NULL)
        return (pSt);

    pSt->hdr.ID = ID;               // unique id assigned for referencing
    pSt->hdr.pNxtObj = NULL;        // initialize pointer to NULL
    pSt->hdr.type = OBJ_STATICTEXT; // set object type
    pSt->hdr.left = left;           // left,top corner
    pSt->hdr.top = top;
    pSt->hdr.right = right;         // right buttom corner
    pSt->hdr.bottom = bottom;
    pSt->pText = pText;             // location of the text
    pSt->hdr.state = state;
    pSt->hdr.DrawObj = StDraw;          // draw function
    pSt->hdr.MsgObj = StTranslateMsg;   // message function
    pSt->hdr.MsgDefaultObj = NULL;      // default message function
    pSt->hdr.FreeObj = NULL;            // free function

    // Set the color scheme to be used
    if(pScheme == NULL)
        pSt->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pSt->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    pSt->textHeight = 0;
    if(pSt->pText != NULL)
    {

        // Set the text height
        pSt->textHeight = GetTextHeight(pSt->hdr.pGolScheme->pFont);
    }

    GOLAddObject((OBJ_HEADER *)pSt);
    return (pSt);
}

/*********************************************************************
* Function: StSetText(STATICTEXT *pSt, XCHAR *pText)
*
* Notes: Sets the string that will be used.
*
*********************************************************************/
void StSetText(STATICTEXT *pSt, XCHAR *pText)
{
    pSt->pText = pText;
    pSt->textHeight = GetTextHeight(pSt->hdr.pGolScheme->pFont);
}

/*********************************************************************
* Function: WORD StTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Evaluates the message if the object will be affected by the
*        message or not.
*
*********************************************************************/
```

```
WORD StTranslateMsg(void *pObj, GOL_MSG *pMsg)
{

    STATICTEXT *pSt;

    pSt = (STATICTEXT *)pObj;


    // Evaluate if the message is for the static text
    // Check if disabled first
    if(GetState(pSt, ST_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in static text control borders
        if
        (
            (pSt->hdr.left < pMsg->param1) &&
            (pSt->hdr.right > pMsg->param1) &&
            (pSt->hdr.top < pMsg->param2) &&
            (pSt->hdr.bottom > pMsg->param2)
        )
        {
            return (ST_MSG_SELECTED);
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*********************************************************************
* Function: WORD StDraw(void *pObj)
*
* Notes: This is the state machine to draw the static text.
*
*********************************************************************/
WORD StDraw(void *pObj)
{
    typedef enum
    {
        ST_STATE_IDLE,
        ST_STATE_CLEANAREA,
        ST_STATE_INIT,
        ST_STATE_SETALIGN,
        ST_STATE_DRAWTEXT
    } ST_DRAW_STATES;

    static ST_DRAW_STATES state = ST_STATE_IDLE;
    static SHORT charCtr = 0, lineCtr = 0;
    static XCHAR *pCurLine = NULL;
    SHORT textWidth;
    XCHAR ch = 0;
    STATICTEXT *pSt;

    pSt = (STATICTEXT *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case ST_STATE_IDLE:
            SetClip(CLIP_DISABLE);

            if(GetState(pSt, ST_HIDE))
            {
                SetColor(pSt->hdr.pGolScheme->CommonBkColor);
                if(!Bar(pSt->hdr.left, pSt->hdr.top, pSt->hdr.right, pSt->hdr.bottom))
```

```
                    return (0);

                // State is still IDLE STATE so no need to set
                return (1);
            }

            if(GetState(pSt, ST_DRAW))
            {
                // show frame if specified to be shown
                SetLineType(SOLID_LINE);
                SetLineThickness(NORMAL_LINE);

                if(GetState(pSt, ST_FRAME))
                {

                    if(!GetState(pSt, ST_DISABLED))
                    {

                        // show enabled color
                        SetColor(pSt->hdr.pGolScheme->Color1);
                        if(!Rectangle(pSt->hdr.left, pSt->hdr.top, pSt->hdr.right,
pSt->hdr.bottom))
                            return (0);
                    }
                    else
                    {

                        // show disabled color
                        SetColor(pSt->hdr.pGolScheme->ColorDisabled);
                        if(!Rectangle(pSt->hdr.left, pSt->hdr.top, pSt->hdr.right,
pSt->hdr.bottom))
                            return (0);
                    }
                }
                else
                {
                    // show enabled color
                    SetColor(pSt->hdr.pGolScheme->CommonBkColor);
                    if(!Rectangle(pSt->hdr.left, pSt->hdr.top, pSt->hdr.right,
pSt->hdr.bottom))
                        return (0);

                }
            }

            state = ST_STATE_CLEANAREA;

        case ST_STATE_CLEANAREA:

            // clean area where text will be placed.
            SetColor(pSt->hdr.pGolScheme->CommonBkColor);
            if(!Bar(pSt->hdr.left + 1, pSt->hdr.top + 1, pSt->hdr.right - 1,
pSt->hdr.bottom - 1))
                return (0);

            // set clipping area, text will only appear inside the static text area.
            SetClip(CLIP_ENABLE);
            SetClipRgn(pSt->hdr.left + ST_INDENT, pSt->hdr.top, pSt->hdr.right - ST_INDENT,
pSt->hdr.bottom);
            state = ST_STATE_INIT;

        case ST_STATE_INIT:
            if(IsDeviceBusy())
                return (0);

            // set the text color
            if(!GetState(pSt, ST_DISABLED))
            {
                SetColor(pSt->hdr.pGolScheme->TextColor0);
            }
            else
            {
```

```
                SetColor(pSt->hdr.pGolScheme->TextColorDisabled);
            }

            // use the font specified in the object
            SetFont(pSt->hdr.pGolScheme->pFont);
            pCurLine = pSt->pText;                  // get first line of text
            state = ST_STATE_SETALIGN;              // go to drawing of text

        case ST_STATE_SETALIGN:
            st_state_alignment : if(!charCtr)
            {

                // set position of the next character (based on alignment and next
character)
                textWidth = GetTextWidth(pCurLine, pSt->hdr.pGolScheme->pFont);

                // Display text with center alignment
                if(GetState(pSt, (ST_CENTER_ALIGN)))
                {
                    MoveTo((pSt->hdr.left + pSt->hdr.right - textWidth) >> 1, pSt->hdr.top
+ (lineCtr * pSt->textHeight));
                }

                // Display text with right alignment
                else if(GetState(pSt, (ST_RIGHT_ALIGN)))
                {
                    MoveTo((pSt->hdr.right - textWidth - ST_INDENT), pSt->hdr.top +
(lineCtr * pSt->textHeight));
                }

                // Display text with left alignment
                else
                {
                    MoveTo(pSt->hdr.left + ST_INDENT, pSt->hdr.top + (lineCtr *
pSt->textHeight));
                }
            }

            state = ST_STATE_DRAWTEXT;

        case ST_STATE_DRAWTEXT:
            ch = *(pCurLine + charCtr);

            // output one character at time until a newline character or a NULL character
is sampled
            while((0x0000 != ch) && (0x000A != ch))
            {
                if(!OutChar(ch))
                    return (0);                     // render the character
                charCtr++;                          // update to next character
                ch = *(pCurLine + charCtr);
            }

            // pCurText is updated for the next line
            if(ch == 0x000A)
            {                                       // new line character
                pCurLine = pCurLine + charCtr + 1;  // go to first char of next line
                lineCtr++;                          // update line counter
                charCtr = 0;                        // reset char counter
                goto st_state_alignment;            // continue to next line
            }

            // end of text string is reached no more lines to display
            else
            {
                pCurLine = NULL;                    // reset static variables
                lineCtr = 0;
                charCtr = 0;
                SetClip(CLIP_DISABLE);              // remove clipping
                state = ST_STATE_IDLE;              // go back to IDLE state
                return (1);
            }
```

```
    }
    return (1);
}
#endif // USE_STATICTEXT
```

14.1.37 StaticText.h

Functions

	Name	Description
	StCreate (see page 255)	This function creates a STATICTEXT (see page 259) 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 (see page 256)	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 (see page 258)	This function sets the string that will be used for the object.
	StTranslateMsg (see page 258)	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
ST_CENTER_ALIGN (see page 254)	Bit to indicate text is center aligned.
ST_DISABLED (see page 254)	Bit for disabled state.
ST_DRAW (see page 254)	Bit to indicate static text must be redrawn.
ST_FRAME (see page 255)	Bit to indicate frame is displayed.
ST_HIDE (see page 255)	Bit to remove object from screen.
ST_RIGHT_ALIGN (see page 255)	Bit to indicate text is left aligned.
ST_UPDATE (see page 255)	Bit to indicate that text area only is redrawn.
StGetText (see page 257)	This macro returns the address of the current text string used for the object.

Structures

Name	Description
STATICTEXT (see page 259)	Defines the parameters required for a Static Text Object.

Description

This is file StaticText.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Static text
 *****************************************************************************
 * FileName:        StaticText.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 ****************************************************************************/
#ifndef _STATICTEXT_H
    #define _STATICTEXT_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define ST_DISABLED     0x0002  // Bit for disabled state.
    #define ST_RIGHT_ALIGN  0x0004  // Bit to indicate text is left aligned.
    #define ST_CENTER_ALIGN 0x0008  // Bit to indicate text is center aligned.
    #define ST_FRAME        0x0010  // Bit to indicate frame is displayed.
    #define ST_UPDATE       0x2000  // Bit to indicate that text area only is redrawn.
    #define ST_DRAW         0x4000  // Bit to indicate static text must be redrawn.
    #define ST_HIDE         0x8000  // Bit to remove object from screen.

/* Indent constant for the text used in the frame. */
    #define ST_INDENT   0x02        // Text indent constant.

/*********************************************************************
* Overview: Defines the parameters required for a Static Text Object.
*
*********************************************************************/
typedef struct
{
    OBJ_HEADER  hdr;        // Generic header for all Objects (see OBJ_HEADER).
    SHORT       textHeight; // Pre-computed text height.
    XCHAR       *pText;     // The pointer to text used.
} STATICTEXT;

/*********************************************************************
* Macros:  StGetText(pSt)
*
* Overview: This macro returns the address of the current
*           text string used for the object.
*
* PreCondition: none
*
* Input: pSt - Pointer to the object.
*
* Output: Returns the pointer to the text string used.
*
* Side Effects: none
```

```
 *
 ********************************************************************/
    #define StGetText(pSt)  pSt->pText

 /********************************************************************
 * Function: StSetText(STATICTEXT *pSt, XCHAR *pText)
 *
 * Overview: This function sets the string that will be used for the object.
 *
 * PreCondition: none
 *
 * Input: pSt - The pointer to the object whose text string will be modified.
 *        pText - The pointer to the string that will be used.
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
 void        StSetText(STATICTEXT *pSt, XCHAR *pText);

 /********************************************************************
 * Function: STATICTEXT  *StCreate(WORD ID, SHORT left, SHORT top, SHORT right, SHORT
 bottom,
 *                                 WORD state , XCHAR *pText, GOL_SCHEME *pScheme)
 *
 * 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.
 *
 * PreCondition: none
 *
 * Input: ID - Unique user defined ID for the object instance.
 *        left - Left most position of the object.
 *        top - Top most position of the object.
 *        right - Right most position of the object.
 *        bottom - Bottom most position of the object.
 *        state - Sets the initial state of the object.
 *        pText - Pointer to the text used in the static text.
 *        pScheme - Pointer to the style scheme. Set to NULL if
 *                  default style scheme is used.
 *
 * Output: Returns the pointer to the object created.
 *
 * Example:
 *   <CODE>
 *   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 Text\n Example", // text
 *                pScheme);                 // use given scheme
 *
 *       while(!StDraw(pSt));               // draw the object
 *   </CODE>
 *
 * Side Effects: none
 *
 ********************************************************************/
 STATICTEXT  *StCreate
             (
                 WORD        ID,
                 SHORT       left,
                 SHORT       top,
                 SHORT       right,
                 SHORT       bottom,
                 WORD        state,
```

```
                XCHAR       *pText,
                GOL_SCHEME  *pScheme
            );

/*****************************************************************
* Function: WORD StTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pSt   - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*****************************************************************/
WORD        StTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*****************************************************************
* Function: WORD StDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
*
* Input: pSt - Pointer to the object to be rendered.
*
* Output: 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.
*
* Example:
*   See StCreate() Example.
*
* Side Effects: none
*
*****************************************************************/
WORD StDraw(void *pObj);
#endif // _STATICTEXT_H
```

14.1.38 TextEntry.c

This is file TextEntry.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  TextEntry
 *****************************************************************************
 * FileName:        Textentry.c
 * Dependencies:    Textentry.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 Version 3.00, C32
 * Linker:          MPLAB LINK30, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Harold Serrano       10/24/08    ...
 * PAT                  07/01/09    Updated for 2D accelerated primitive support.
 * PAT                  04/15/10    Corrected TeSetBuffer() issue on string max size.
 *****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_TEXTENTRY

/*******************************************************************
* Function: 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)
*
*
* Notes:
*
*******************************************************************/
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
)
{
    TEXTENTRY   *pTe = NULL;                    //Text entry
    pTe = (TEXTENTRY *)GFX_malloc(sizeof(TEXTENTRY));
    if(pTe == NULL)
        return (NULL);

    pTe->hdr.ID = ID;
    pTe->hdr.pNxtObj = NULL;
    pTe->hdr.type = OBJ_TEXTENTRY;              // set object type
    pTe->hdr.left = left;                       // left parameter of the text-entry
    pTe->hdr.top = top;                         // top parameter of the text-entry
    pTe->hdr.right = right;                     // right parameter of the text-entry
    pTe->hdr.bottom = bottom;                   // bottom parameter of the text-entry
    pTe->hdr.state = state;                     // State of the Text-Entry
    pTe->horizontalKeys = horizontalKeys;       // number of horizontal keys
    pTe->verticalKeys = verticalKeys;           // number of vertical keys
    pTe->CurrentLength = 0;                     // current length of text
    pTe->pHeadOfList = NULL;
    TeSetBuffer(pTe, pBuffer, bufferLength);    // set the text to be displayed buffer
length is also initialized in this call
    pTe->pActiveKey = NULL;
    pTe->hdr.DrawObj = TeDraw;                  // draw function
    pTe->hdr.MsgObj = TeTranslateMsg;           // message function
    pTe->hdr.MsgDefaultObj = TeMsgDefault;      // default message function
    pTe->hdr.FreeObj = TeDelKeyMembers;         // free function

    // Set the color scheme to be used
    if(pScheme == NULL)
        pTe->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pTe->hdr.pGolScheme = (GOL_SCHEME *)pScheme;

    // Set the font to be used
    if(pDisplayFont == NULL)
        pTe->pDisplayFont = (void *) &GOLFontDefault;
    else
        pTe->pDisplayFont = pDisplayFont;

    //check if either values of horizontal keys and vertical keys are equal to zero
    if((pTe->horizontalKeys != 0) || (pTe->verticalKeys != 0))
    {

        //create the key members, return null if not successful
        if(TeCreateKeyMembers(pTe, pText) == 0)
        {
            GFX_free(pTe);
            return (NULL);
        }
    }

    //Add this new widget object to the GOL list
    GOLAddObject((OBJ_HEADER *)pTe);
    return (pTe);
}   //end TeCreate()

/*********************************************************************
* Function: WORD TeDraw(void *pObj)
*
* Notes: This function draws the keys with their appropriate text
*
```

```
*****************************************************************/
WORD TeDraw(void *pObj)
{
    static WORD         faceClr, embossLtClr, embossDkClr, xText, yText;
    static XCHAR        XcharTmp;
    static KEYMEMBER    *pKeyTemp = NULL;

    static WORD         CountOfKeys = 0;
    static WORD         counter = 0;
    static XCHAR        hideChar[2] = {0x2A, 0x00};

    typedef enum
    {
        TE_START,
        TE_HIDE_WIDGET,
        TE_DRAW_PANEL,
        TE_INIT_DRAW_EDITBOX,
        TE_DRAW_EDITBOX,
        TE_DRAW_KEY_INIT,
        TE_DRAW_KEY_SET_PANEL,
        TE_DRAW_KEY_DRAW_PANEL,
        TE_DRAW_KEY_TEXT,
        TE_DRAW_KEY_UPDATE,
        TE_UPDATE_STRING_INIT,
        TE_UPDATE_STRING,
        TE_WAIT_ERASE_EBOX_AREA,
        TE_UPDATE_CHARACTERS,
    } TE_DRAW_STATES;

    static TE_DRAW_STATES state = TE_START;
    TEXTENTRY *pTe;

    pTe = (TEXTENTRY *)pObj;

    if(IsDeviceBusy())
        return (0);

    switch(state)
    {
        case TE_START:
            if(IsDeviceBusy())
                return (0);

            if(GetState(pTe, TE_HIDE))
            {
                SetColor(pTe->hdr.pGolScheme->CommonBkColor);
                Bar(pTe->hdr.left, pTe->hdr.top, pTe->hdr.right, pTe->hdr.bottom);
                state = TE_HIDE_WIDGET;
                goto te_hide_widget;
            }
            else
            {
                if(GetState(pTe, TE_DRAW))
                {

                    /************DRAW THE WIDGET PANEL****************************/
                    GOLPanelDraw
                    (
                        pTe->hdr.left,
                        pTe->hdr.top,
                        pTe->hdr.right,
                        pTe->hdr.bottom,
                        0,
                        pTe->hdr.pGolScheme->Color0,        //face color of panel
                        pTe->hdr.pGolScheme->EmbossDkColor, //emboss dark color
                        pTe->hdr.pGolScheme->EmbossLtColor, //emboss light color
                        NULL,
                        GOL_EMBOSS_SIZE
                    );
                    state = TE_DRAW_PANEL;
                    goto te_draw_panel;
                }
```

```
            // update the keys (if TE_UPDATE_TEXT is also set it will also be redrawn)
            // at the states after the keys are updated
            else if(GetState(pTe, TE_UPDATE_KEY))
            {
                state = TE_DRAW_KEY_INIT;
                goto te_draw_key_init_st;
            }

            // check if updating only the text displayed
            else if(GetState(pTe, TE_UPDATE_TEXT))
            {
                state = TE_UPDATE_STRING_INIT;
                goto te_update_string_init_st;
            }
        }

    /*hide the widget*/
    case TE_HIDE_WIDGET:
        te_hide_widget : if(IsDeviceBusy()) return (0);
        state = TE_START;
        return (1);

    /*Draw the widget of the Text-Entry*/
    case TE_DRAW_PANEL:
        te_draw_panel : if(!GOLPanelDrawTsk()) return (0);
        state = TE_INIT_DRAW_EDITBOX;

    case TE_INIT_DRAW_EDITBOX:

        //Draw the editbox
        GOLPanelDraw
        (
        pTe->hdr.left,
        pTe->hdr.top,
        pTe->hdr.right,
        pTe->hdr.top + GetTextHeight(pTe->pDisplayFont) + (GOL_EMBOSS_SIZE << 1),
        0,
        pTe->hdr.pGolScheme->Color1; ,
    pTe->hdr.pGolScheme->EmbossDkColor,
    pTe->hdr.pGolScheme->EmbossLtColor,
    NULL,
    GOL_EMBOSS_SIZE
        );

        state = TE_DRAW_EDITBOX;

    case TE_DRAW_EDITBOX:
        if(!GOLPanelDrawTsk())
            return (0);
        state = TE_DRAW_KEY_INIT;

    /* ******************************************************************* */

    /*                  Update the keys                                    */

    /* ******************************************************************* */
    case TE_DRAW_KEY_INIT:
        te_draw_key_init_st : embossLtClr = pTe->hdr.pGolScheme->EmbossLtColor;
        embossDkClr = pTe->hdr.pGolScheme->EmbossDkColor;
        faceClr = pTe->hdr.pGolScheme->Color0;

        // if the active key update flag is set, only one needs to be redrawn
        if((GetState(pTe, TE_DRAW) != TE_DRAW) && (pTe->pActiveKey->update == TRUE))
        {
            CountOfKeys = (pTe->horizontalKeys * pTe->verticalKeys) -
            1;
            pKeyTemp = pTe->pActiveKey;
        }
        else
        {
            CountOfKeys = 0;
```

```
                pKeyTemp = pTe->pHeadOfList;
            }

            state = TE_DRAW_KEY_SET_PANEL;

        case TE_DRAW_KEY_SET_PANEL:
            te_draw_key_set_panel_st : if(CountOfKeys < (pTe->horizontalKeys *
pTe->verticalKeys))
            {

                // check if we need to draw the panel
                if(GetState(pTe, TE_DRAW) != TE_DRAW)
                {
                    if(pKeyTemp->update == TRUE)
                    {

                        // set the colors needed
                        if(GetState(pTe, TE_KEY_PRESSED))
                        {
                            embossLtClr = pTe->hdr.pGolScheme->EmbossDkColor;
                            embossDkClr = pTe->hdr.pGolScheme->EmbossLtColor;
                            faceClr = pTe->hdr.pGolScheme->Color1;
                        }
                        else
                        {
                            embossLtClr = pTe->hdr.pGolScheme->EmbossLtColor;
                            embossDkClr = pTe->hdr.pGolScheme->EmbossDkColor;
                            faceClr = pTe->hdr.pGolScheme->Color0;
                        }
                    }
                    else
                    {
                        state = TE_DRAW_KEY_UPDATE;
                        goto te_draw_key_update_st;
                    }
                }

                if(GetState(pTe, TE_DISABLED) == TE_DISABLED)
                {
                    faceClr = SetColor(pTe->hdr.pGolScheme->ColorDisabled);
                }

                // set up the panel
                GOLPanelDraw
                (
                    pKeyTemp->left,
                    pKeyTemp->top,
                    pKeyTemp->right,
                    pKeyTemp->bottom,
                    0,
                    faceClr,
                    embossLtClr,
                    embossDkClr,
                    NULL,
                    GOL_EMBOSS_SIZE
                );

                state = TE_DRAW_KEY_DRAW_PANEL;
            }
            else
            {
                state = TE_UPDATE_STRING_INIT;
                goto te_update_string_init_st;
            }

        case TE_DRAW_KEY_DRAW_PANEL:
            if(!GOLPanelDrawTsk())
                return (0);

            // reset the update flag since the key panel is already redrawn
            pKeyTemp->update = FALSE;
```

```
            //set the text coordinates of the drawn key
            xText = ((pKeyTemp->left) + (pKeyTemp->right) - (pKeyTemp->textWidth)) >>
            1;
            yText = ((pKeyTemp->bottom) + (pKeyTemp->top) - (pKeyTemp->textHeight)) >>
            1;

            //set color of text
            // if the object is disabled, draw the disabled colors
            if(GetState(pTe, TE_DISABLED) == TE_DISABLED)
            {
                SetColor(pTe->hdr.pGolScheme->TextColorDisabled);
            }
            else
            {
                if((GetState(pTe, TE_DRAW) != TE_DRAW) && (GetState(pTe, TE_KEY_PRESSED))
 == TE_KEY_PRESSED)
                {
                    SetColor(pTe->hdr.pGolScheme->TextColor1);
                }
                else
                {
                    SetColor(pTe->hdr.pGolScheme->TextColor0);
                }
            }

            //output the text
            MoveTo(xText, yText);

            // set the font to be used
            SetFont(pTe->hdr.pGolScheme->pFont);

            state = TE_DRAW_KEY_TEXT;

        case TE_DRAW_KEY_TEXT:
            if(!OutText(pKeyTemp->pKeyName))
                return (0);
            state = TE_DRAW_KEY_UPDATE;

        case TE_DRAW_KEY_UPDATE:
            te_draw_key_update_st :

            // update loop variables
            CountOfKeys++;
            pKeyTemp = pKeyTemp->pNextKey;

            state = TE_DRAW_KEY_SET_PANEL;
            goto te_draw_key_set_panel_st;

        /* ******************************************************************* */

        /*                  Update the displayed string                        */

        /* ******************************************************************* */
        case TE_UPDATE_STRING_INIT:
            te_update_string_init_st :

            // check if there are characters to remove
            if(pTe->pActiveKey != NULL)
            {
                if(pTe->pActiveKey->command == TE_DELETE_COM)
                {
                    if(pTe->CurrentLength == 0)
                    {
                        state = TE_START;
                        return (1);
                    }
                }
            }
            else
            {

                // check if text indeed needs to be updated
```

```
                if(pTe->CurrentLength == pTe->outputLenMax)
                {
                    state = TE_START;
                    return (1);
                }
            }

            //set the clipping region
            SetClipRgn
            (
                pTe->hdr.left + GOL_EMBOSS_SIZE,
                pTe->hdr.top + GOL_EMBOSS_SIZE,
                pTe->hdr.right - GOL_EMBOSS_SIZE,
                pTe->hdr.top + GOL_EMBOSS_SIZE + GetTextHeight(pTe->pDisplayFont)
            );

            SetClip(1);     //set the clipping
            if(GetState(pTe, TE_DRAW))
            {

                // update only the displayed text
                // position the string rendering on the right position
                if(GetState(pTe, TE_ECHO_HIDE))
                {

                    // fill the area with '*' character so we use the width of this
character
                    MoveTo
                    (
                        pTe->hdr.right - 4 - GOL_EMBOSS_SIZE - (GetTextWidth(hideChar,
pTe->pDisplayFont) * pTe->CurrentLength),
                        pTe->hdr.top + GOL_EMBOSS_SIZE
                    );
                }
                else
                {
                    MoveTo
                    (
                        pTe->hdr.right - 4 - GOL_EMBOSS_SIZE - GetTextWidth(pTe->pTeOutput,
pTe->pDisplayFont),
                        pTe->hdr.top + GOL_EMBOSS_SIZE
                    );
                }
            }
            else if(GetState(pTe, TE_UPDATE_TEXT))
            {

                // erase the current text by drawing a bar over the edit box area
                SetColor(pTe->hdr.pGolScheme->Color1);
                Bar
                (
                    pTe->hdr.left + GOL_EMBOSS_SIZE,
                    pTe->hdr.top + GOL_EMBOSS_SIZE,
                    pTe->hdr.right - GOL_EMBOSS_SIZE,
                    pTe->hdr.top + GOL_EMBOSS_SIZE + GetTextHeight(pTe->pDisplayFont)
                );

                // we have to make sure we finish the Bar() first before we continue.
                state = TE_WAIT_ERASE_EBOX_AREA;
                goto te_wait_erase_ebox_area;
            }
            else
            {
                SetClip(0); //reset the clipping
                state = TE_START;
                return (1);
            }

            counter = 0;
            state = TE_UPDATE_STRING;
            goto te_update_string;
```

```
        case TE_WAIT_ERASE_EBOX_AREA:
            te_wait_erase_ebox_area : if(IsDeviceBusy()) return (0);

            // check if the command given is delete a character
            if(pTe->pActiveKey->command == TE_DELETE_COM)
            {
                *(pTe->pTeOutput + (--pTe->CurrentLength)) = 0;
            }

            // position the cursor to the start of string rendering
            // notice that we need to remove the characters first before we position the
cursor when
            // deleting characters
            if(GetState(pTe, TE_ECHO_HIDE))
            {

                // fill the area with '*' character so we use the width of this character
                MoveTo
                (
                    pTe->hdr.right - 4 - GOL_EMBOSS_SIZE - (GetTextWidth(hideChar,
pTe->pDisplayFont) * (pTe->CurrentLength)),
                    pTe->hdr.top + GOL_EMBOSS_SIZE
                );
            }
            else
            {
                MoveTo
                (
                    pTe->hdr.right - 4 - GOL_EMBOSS_SIZE - GetTextWidth(pTe->pTeOutput,
pTe->pDisplayFont),
                    pTe->hdr.top + GOL_EMBOSS_SIZE
                );
            }

            counter = 0;
            state = TE_UPDATE_STRING;

        case TE_UPDATE_STRING:
            te_update_string :

            //output the text
            SetColor(pTe->hdr.pGolScheme->TextColor1);
            SetFont(pTe->pDisplayFont);

            // this is manually doing the OutText() function but with the capability to
replace the
            // characters to the * character when hide echo is
enabled.
            XcharTmp = *((pTe->pTeOutput) + counter);
            if(XcharTmp < (XCHAR)15)
            {

                // update is done time to return to start and exit with success
                SetClip(0); //reset the clipping
                state = TE_START;
                return (1);
            }
            else
            {
                if(GetState(pTe, TE_ECHO_HIDE))
                    OutChar(0x2A);
                else
                    OutChar(XcharTmp);
                state = TE_UPDATE_CHARACTERS;
            }

        case TE_UPDATE_CHARACTERS:
            if(IsDeviceBusy()) return (0);
            counter++;
            state = TE_UPDATE_STRING;
            goto te_update_string;
    }                       //end switch
```

```
    return (1);
}                           //end TeDraw()

/*****************************************************************
* Function: TeTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Notes: Function to check which key was pressed/released
*
*****************************************************************/
WORD TeTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    SHORT       NumberOfKeys, param1, param2;
    KEYMEMBER   *pKeyTemp = NULL;
    TEXTENTRY *pTe;

    pTe = (TEXTENTRY *)pObj;

    // Check if disabled first
    if(GetState(pTe, TE_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN

    //find the total number of keys
    NumberOfKeys = (pTe->horizontalKeys) *
    (pTe->verticalKeys);
    param1 = pMsg->param1;
    param2 = pMsg->param2;

    if((pMsg->type == TYPE_TOUCHSCREEN))
    {

        // Check if it falls in the panel of the TextEntry
        if
        (
            (pTe->hdr.left < pMsg->param1) &&
            (pTe->hdr.right > pMsg->param1) &&
            (pTe->hdr.top + (GetTextHeight(pTe->pDisplayFont) + (GOL_EMBOSS_SIZE << 1)) <
pMsg->param2) &&
            (pTe->hdr.bottom > pMsg->param2)
        )
        {

            /* If it fell inside the TextEntry panel, go through the link list and check
which one was pressed
               At this point the touch screen event is either EVENT_MOVE or EVENT_PRESS.
            */

            //point to the head of the link list
            pKeyTemp = pTe->pHeadOfList;

            while(pKeyTemp != NULL)
            {
                if
                (
                    (pKeyTemp->left < param1) &&
                    (pKeyTemp->right > param1) &&
                    (pKeyTemp->top < param2) &&
                    (pKeyTemp->bottom > param2)
                )
                {
                    if(pMsg->uiEvent == EVENT_RELEASE)
                    {
                        pTe->pActiveKey = pKeyTemp;
                        pKeyTemp->update = TRUE;

                        if(pTe->pActiveKey->state == TE_KEY_PRESSED)
                        {
                            if(pKeyTemp->command == 0)
                                return (TE_MSG_ADD_CHAR);
```

```
                        //command for a TE_DELETE_COM key
                        if(pKeyTemp->command == TE_DELETE_COM)
                            return (TE_MSG_DELETE);

                        //command for a TE_SPACE_COM key 0x20
                        if(pKeyTemp->command == TE_SPACE_COM)
                            return (TE_MSG_SPACE);

                        //command for a TE_ENTER_COM key
                        if(pKeyTemp->command == TE_ENTER_COM)
                            return (TE_MSG_ENTER);
                    }

                    // this is a catch all backup
                    return (TE_MSG_RELEASED);
                }
                else
                {

                    // to shift the press to another key make sure that there are no
other
                    // keys currently pressed. If there is one it must be released
first.
                    // check if there are previously pressed keys
                    if(GetState(pTe, TE_KEY_PRESSED))
                    {

                        // there is a key being pressed.
                        if(pKeyTemp->index != pTe->pActiveKey->index)
                        {

                            // release the currently pressed key first
                            pTe->pActiveKey->update = TRUE;
                            return (TE_MSG_RELEASED);
                        }
                    }
                    else
                    {

                        // check if the active key is not pressed
                        // if not, set to press since the current touch event
                        // is either move or press
                        // check if there is an active key already set
                        // if none, set the current key as active and return a pressed
mesage
                        if(pTe->pActiveKey == NULL)
                        {
                            pTe->pActiveKey = pKeyTemp;
                            pKeyTemp->update = TRUE;
                            return (TE_MSG_PRESSED);
                        }

                        if(pTe->pActiveKey->state != TE_KEY_PRESSED)
                        {
                            pTe->pActiveKey = pKeyTemp;
                            pKeyTemp->update = TRUE;
                            return (TE_MSG_PRESSED);
                        }
                        else
                        {
                            return (OBJ_MSG_INVALID);
                        }
                    }
                }
            }
            else
            {

                // if the key is in the pressed state and current touch is not here
                // then it has to be redrawn
                if(pKeyTemp->state == TE_KEY_PRESSED)
                {
```

```
                        pTe->pActiveKey = pKeyTemp;
                        pKeyTemp->update = TRUE;
                        return (TE_MSG_RELEASED);
                    }
                }

                //access the next link list
                pKeyTemp = pKeyTemp->pNextKey;
            }   //end while
        }
        else
        {
            if((pMsg->uiEvent == EVENT_MOVE) && (GetState(pTe, TE_KEY_PRESSED)))
            {
                pTe->pActiveKey->update = TRUE;
                return (TE_MSG_RELEASED);
            }
        }
    }

    return (OBJ_MSG_INVALID);
        #endif // USE_TOUCHSCREEN
}               //end TeTranslateMsg()

/*********************************************************************
* Function: TeMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
*
* Notes: This the default operation to change the state of the key.
*        Called inside GOLMsg() when GOLMsgCallback() returns a 1.
*
*********************************************************************/
void TeMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg)
{
    TEXTENTRY *pTe;

    pTe = (TEXTENTRY *)pObj;

    switch(translatedMsg)
    {
        case TE_MSG_DELETE:
            SetState(pTe, TE_UPDATE_KEY | TE_UPDATE_TEXT);
            break;

        case TE_MSG_SPACE:
            TeSpaceChar(pTe);
            SetState(pTe, TE_UPDATE_KEY | TE_UPDATE_TEXT);
            break;

        case TE_MSG_ENTER:
            SetState(pTe, TE_UPDATE_KEY);
            break;

        case TE_MSG_ADD_CHAR:
            TeAddChar(pTe);
            SetState(pTe, TE_UPDATE_KEY | TE_UPDATE_TEXT);
            break;

        case TE_MSG_PRESSED:
            (pTe->pActiveKey)->state = TE_KEY_PRESSED;
            SetState(pTe, TE_KEY_PRESSED | TE_UPDATE_KEY);
            return;

        case TE_MSG_RELEASED:
            (pTe->pActiveKey)->state = 0;
            ClrState(pTe, TE_KEY_PRESSED);  // reset pressed
            SetState(pTe, TE_UPDATE_KEY);   // redraw
            return;
    }

    if(pTe->pActiveKey != NULL)
        (pTe->pActiveKey)->state = 0;
```

```
    ClrState(pTe, TE_KEY_PRESSED);
}

/*********************************************************************
* Function: void TeClearBuffer(TEXTENTRY *pTe)
*
* Notes: This function will clear the edibox and the buffer.
*        You must set the drawing state bit TE_UPDATE_TEXT
*        to update the TEXTENTRY on the screen.
*
*********************************************************************/
void TeClearBuffer(TEXTENTRY *pTe)
{
    WORD    i;

    //clear the buffer
    for(i = 0; i < (pTe->outputLenMax); i++)
    {
        *(pTe->pTeOutput + i) = 0;
    }

    pTe->CurrentLength = 0;
}

/*********************************************************************
* Function: void TeSetBuffer(TEXTENTRY *pTe, XCHAR *pText, WORD size)
*
* Notes: This function will replace the currently used buffer.
*        MaxSize defines the length of the buffer. Buffer must be
*        a NULL terminated string.
*
*********************************************************************/
void TeSetBuffer(TEXTENTRY *pTe, XCHAR *pText, WORD MaxSize)
{
    WORD    count = 0;
    XCHAR   *pTemp;

    pTemp = pText;

    while(*pTemp != 0)
    {
        if(count >= MaxSize)
            break;
        *pTemp++;
        count++;
    }

    // terminate the string
    *pTemp = 0;

    pTe->CurrentLength = count;
    pTe->outputLenMax = MaxSize-1;
    pTe->pTeOutput = pText;
}

/*********************************************************************
* Function: BOOL TeIsKeyPressed(TEXTENTRY *pTe,WORD index)
*
* Notes: This function will check if the key was pressed. If no
*        key was pressed it will return FALSE.
*
*********************************************************************/
BOOL TeIsKeyPressed(TEXTENTRY *pTe, WORD index)
{
    KEYMEMBER   *pTemp;

    pTemp = pTe->pHeadOfList;

    //search the key using the given index
    while(index != pTemp->index)
    {
```

```
        // catch all check
        if(pTemp == NULL)
            return (FALSE);
        pTemp = pTemp->pNextKey;
    }

    if(pTemp->state == TE_KEY_PRESSED)
    {
        return (TRUE);
    }
    else
    {
        return (FALSE);
    }
}

/*********************************************************************
* Function: BOOL TeSetKeyCommand(TEXTENTRY *pTe,WORD index,WORD command)
*
* Notes: This function will assign a command to a particular key.
*        Returns TRUE if sucessful and FALSE if not.
*
*********************************************************************/
BOOL TeSetKeyCommand(TEXTENTRY *pTe, WORD index, WORD command)
{
    KEYMEMBER   *pTemp;

    pTemp = pTe->pHeadOfList;

    //search the key using the given index
    while(index != pTemp->index)
    {

        // catch all check
        if(pTemp == NULL)
            return (FALSE);
        pTemp = pTemp->pNextKey;
    }

    pTemp->command = command;
    return (TRUE);
}

/*********************************************************************
* Function: TeGetKeyCommand(pTe, index)
*
* Notes: This function will return the currently used command by a key
*        with the given index.
*
*********************************************************************/
WORD TeGetKeyCommand(TEXTENTRY *pTe, WORD index)
{
    KEYMEMBER   *pTemp;

    pTemp = pTe->pHeadOfList;

    //search the key using the given index
    while(index != pTemp->index)
    {

        // catch all check
        if(pTemp == NULL)
            return (0);
        pTemp = pTemp->pNextKey;
    }

    return (pTemp->command);
}

/*********************************************************************
* Function: BOOL TeSetKeyText(TEXTENTRY *pTe,WORD index, XCHAR *pText)
*
```

```
* Notes: This function will set the string associated with the key
*        with the new string pText. The key to be modified is determined
*        by the index. Returns TRUE if sucessful and FALSE if not.
*
*********************************************************************/
BOOL TeSetKeyText(TEXTENTRY *pTe, WORD index, XCHAR *pText)
{
    KEYMEMBER   *pTemp;

    pTemp = pTe->pHeadOfList;

    //search the key using the given index
    while(index != pTemp->index)
    {
        // catch all check
        if(pTemp == NULL)
            return (FALSE);
        pTemp = pTemp->pNextKey;
    }

    // Set the the text
    pTemp->pKeyName = pText;

    return (TRUE);
}


/*********************************************************************
* Function: KEYMEMBER *TeCreateKeyMembers(TEXTENTRY *pTe,XCHAR *pText[])
*
* Notes: This function will create the members of the list
*
*********************************************************************/
KEYMEMBER *TeCreateKeyMembers(TEXTENTRY *pTe, XCHAR *pText[])
{
    SHORT       NumberOfKeys, width, height;
    SHORT       keyTop, keyLeft;
    WORD        rowcount, colcount;
    WORD        index = 0;

    KEYMEMBER   *pKl = NULL;    //link list
    KEYMEMBER   *pTail = NULL;

    // determine starting positions of the keys
    keyTop = pTe->hdr.top +
    GetTextHeight(pTe->pDisplayFont) +
    (GOL_EMBOSS_SIZE << 1);
    keyLeft = pTe->hdr.left;    // + GOL_EMBOSS_SIZE;

    //calculate the total number of keys, and width and height of each key
    NumberOfKeys = pTe->horizontalKeys *
    pTe->verticalKeys;
    width = (pTe->hdr.right - keyLeft + 1) / pTe->horizontalKeys;
    height = (pTe->hdr.bottom - keyTop + 1) / pTe->verticalKeys;

    /*create the list and calculate the coordinates of each bottom, and the
textwidth/textheight of each font*/

    //Add a list for each key
    for(colcount = 0; colcount < pTe->verticalKeys; colcount++)
    {
        for(rowcount = 0; rowcount < pTe->horizontalKeys; rowcount++)
        {

            //get storage for new entry
            pKl = (KEYMEMBER *)GFX_malloc(sizeof(KEYMEMBER));
            if(pKl == NULL)
                return (NULL);
            if(pTe->pHeadOfList == NULL)
                pTe->pHeadOfList = pKl;
            if(pTail == NULL)
            {
```

```
                pTail = pKl;
            }
            else
            {
                pTail->pNextKey = pKl;
                pTail = pTail->pNextKey;
            }

            //set the index for the new list
            pKl->index = index;

            // set update flag to off
            pKl->update = FALSE;

            //calculate the x-y coordinate for each key
            pKl->left   = keyLeft + (rowcount * width);
            pKl->top    = keyTop  + (colcount * height);
            pKl->right  = keyLeft + ((rowcount + 1) * width);
            pKl->bottom = keyTop  + ((colcount + 1) * height);

            //Add the text to the list and increase the index
            pKl->pKeyName = pText[index++];

            //set the COMMAND to NULL for all keys
            pKl->command = 0;

            //calculate the textwidth, textheight
            pKl->textWidth = 0;
            pKl->textHeight = 0;
            if(pKl->pKeyName != NULL)
            {

                // Calculate the text width & height
                pKl->textWidth = GetTextWidth(pKl->pKeyName, pTe->hdr.pGolScheme->pFont);
                pKl->textHeight = GetTextHeight(pTe->hdr.pGolScheme->pFont);
            }                   //end if
        }                       //end for
    }                           //end for

    pTail->pNextKey = NULL;

    return (pKl);
}

/*********************************************************************
* Function: void TeDelKeyMembers(void *pObj)
*
* Notes: This function will delete the members of the list
*********************************************************************/
void TeDelKeyMembers(void *pObj)
{
    KEYMEMBER   *pCurItem;
    KEYMEMBER   *pItem;
    TEXTENTRY *pTe;

    pTe = (TEXTENTRY *)pObj;

    pCurItem = pTe->pHeadOfList;

    while(pCurItem != NULL)
    {
        pItem = pCurItem;
        pCurItem = pCurItem->pNextKey;
        GFX_free(pItem);
    }

    pTe->pHeadOfList = NULL;
}

/*********************************************************************
* Function: void TeSpaceChar(TEXTENTRY *pTe)
*
```

```
* Notes: This function will add a space to the buffer/editbox
*********************************************************************/
void TeSpaceChar(TEXTENTRY *pTe)
{

    //first determine if the array has not overflown
    if((pTe->CurrentLength) < pTe->outputLenMax)
    {
        *(pTe->pTeOutput + (pTe->CurrentLength)) = 0x20;
        *(pTe->pTeOutput + (pTe->CurrentLength) + 1) = 0;
    }   //end if
    (pTe->CurrentLength)++;
}

/*********************************************************************
* Function: void TeAddChar(TEXTENTRY *pTe)
*
* Notes: This function will add a character to the buffer/editbox
*********************************************************************/
void TeAddChar(TEXTENTRY *pTe)
{
    XCHAR   *pPoint;

    //first determine if the array has not overflown
    if((pTe->CurrentLength) < pTe->outputLenMax)
    {
        pPoint = (pTe->pActiveKey)->pKeyName;
        while(*(pPoint) != 0)
        {
            *(pTe->pTeOutput + (pTe->CurrentLength)) = *(pPoint)++;
        }
    }   //end if
    else
    {

        // it is full ignore the added key
        return;
    }

    (pTe->CurrentLength)++;
    // add the string terminator
    *(pTe->pTeOutput + pTe->CurrentLength) = 0;
}

#endif // USE_TEXTENTRY
```

14.1.39 TextEntry.h

Functions

	Name	Description
	TeAddChar (see page 270)	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 (see page 264) or TE_DRAW (see page 263) must be set to see the effect of this insertion.
	TeClearBuffer (see page 268)	This function will clear the data in the display. You must set the drawing state bit TE_UPDATE_TEXT (see page 264) to update the TEXTENTRY (see page 276) on the screen.
	TeCreate (see page 265)	This function creates a TEXTENTRY (see page 276) 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.

	TeCreateKeyMembers (see page 270)	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).
	TeDelKeyMembers (see page 272)	This function will delete the KEYMEMBER (see page 277) list assigned to the object from memory. Pointer to the KEYMEMBER (see page 277) list is then initialized to NULL.
	TeDraw (see page 266)	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 (see page 308)(). The number of keys will depend on the horizontal and vertical parameters given (horizontalKeys*verticakKeys).
	TeGetKeyCommand (see page 268)	This function will return the currently used command by a key with the given index.
	TeIsKeyPressed (see page 271)	This function will test if a key given by its index in the TextEntry object has been pressed.
	TeMsgDefault (see page 273)	This function performs the actual state change based on the translated message given. The following state changes are supported:
	TeSetBuffer (see page 267)	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.
	TeSetKeyCommand (see page 269)	This function will assign a command (TE_DELETE_COM (see page 264), TE_SPACE_COM (see page 265) or TE_ENTER_COM (see page 264)) to a key with the given index.
	TeSetKeyText (see page 273)	This function will set the test assigned to a key with the given index.
	TeSpaceChar (see page 272)	This function will insert a space character to the end of the buffer. Drawing states TE_UPDATE_TEXT (see page 264) or TE_DRAW (see page 263) must be set to see the effect of this insertion.
	TeTranslateMsg (see page 275)	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
TE_DELETE_COM (see page 264)	This macro is used to assign a "delete" command on a key.
TE_DISABLED (see page 263)	Bit for disabled state.
TE_DRAW (see page 263)	Bit to indicate object must be redrawn.
TE_ECHO_HIDE (see page 263)	Bit to hide the entered characters and instead echo "*" characters to the display.
TE_ENTER_COM (see page 264)	This macro is used to assign an "enter" (carriage return) command on a key.
TE_HIDE (see page 263)	Bit to indicate object must be removed from screen.
TE_KEY_PRESSED (see page 262)	Bit for press state of one of the keys.
TE_SPACE_COM (see page 265)	This macro is used to assign an insert "space" command on a key.
TE_UPDATE_KEY (see page 263)	Bit to indicate redraw of a key is needed.
TE_UPDATE_TEXT (see page 264)	Bit to indicate redraw of the text displayed is needed.
TeGetBuffer (see page 267)	This macro will return the currently used buffer in the TextEntry object.

Structures

Name	Description
KEYMEMBER (see page 277)	Defines the parameters and the strings assigned for each key.
TEXTENTRY (see page 276)	Defines the parameters required for a TextEntry Object.

Description

This is file TextEntry.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  TextEntry
 *****************************************************************************
 * FileName:        TextEntry.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 10/24/08     ...
 *****************************************************************************/
#ifndef _TEXTENTRY_H
    #define _TEXTENTRY_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define TE_KEY_PRESSED  0x0004  // Bit for press state of one of the keys.
    #define TE_DISABLED     0x0002  // Bit for disabled state.
    #define TE_ECHO_HIDE    0x0008  // Bit to hide the entered characters and instead echo
"*" characters to the display.
    #define TE_DRAW         0x4000  // Bit to indicate object must be redrawn.
    #define TE_HIDE         0x8000  // Bit to indicate object must be removed from screen.
    #define TE_UPDATE_KEY   0x2000  // Bit to indicate redraw of a key is needed.
    #define TE_UPDATE_TEXT  0x1000  // Bit to indicate redraw of the text displayed is
needed.
```

```
/*********************************************************************
* Optional COMMANDS assigned to keys
*********************************************************************/
    #define TE_DELETE_COM   0x01    // This macro is used to assign a "delete" command on a
key.
    #define TE_SPACE_COM    0x02    // This macro is used to assign an insert "space"
command on a key.
    #define TE_ENTER_COM    0x03    // This macro is used to assign an "enter" (carriage
return) command on a key.

// User can use this command to customize application code in the message

// callback function. Use the returned translated TE_MSG_ENTER to detect the key

// pressed was assigned the enter command. Refer to TeTranslateMsg() for details.

/*********************************************************************
* Overview: Defines the parameters and the strings assigned for each key.
*********************************************************************/
typedef struct
{
    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) 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, TE_SPACE_COM or
TE_ENTER_COM.
    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.
} KEYMEMBER;

/*********************************************************************
* Overview: Defines the parameters required for a TextEntry Object.
*********************************************************************/
typedef struct
{
    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.

    // User creates and manages the buffer. Buffer can also be managed using the APIs
provided
    // to add a character, delete the last character or clear the buffer.
    WORD        CurrentLength;  // Current length of the string in the buffer. The maximum
value of this is equal to outputLenMax.

    // TextEntry object will update this parameter when adding, removing characters or
clearing the buffer
    // and switching buffers.
    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

    // the value of this parameter directly.
    KEYMEMBER   *pHeadOfList;   // Pointer to head of the list
    void        *pDisplayFont;  // Pointer to the font used in displaying the text.
} TEXTENTRY;

/*********************************************************************
* Function: 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)
*
* 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.
*
*
* PreCondition: If the object will use customized keys, the structure CUSTOMEKEYS must be
*               populated before calling this function.
*
* Input:  ID -      Unique user defined ID for the object instance
*         left-     Left most position of the object.
*         top -     Top most position of the object.
*         right -   Right most position of the object.
*         bottom -  Bottom most position of the object.
*         state          - state of the widget.
*         horizontalKeys - Number of horizontal keys
*         verticalKeys   - Number of vertical keys
*         pText          - array of pointer to the custom "text" assigned by the user.
*         bufferLength   - length of the buffer assigned by the user.
*         pDisplayFont   - pointer to the font image to be used on the editbox
*         pScheme- Pointer to the style scheme used.
*
* Output Returns the pointer to the object created.
*
* Side Effects: none.
*
*********************************************************************/
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
            );

/*********************************************************************
* Function: WORD TeDraw(void *pObj)
*
* 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*verticakKeys).
*
* PreCondition: Object must be created before this function is called.
*
* Input: pTe- Pointer to the object to be rendered.
*
* Output: 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.
*
* Side Effects: none.
*
*********************************************************************/
WORD TeDraw(void *pObj);
```

```
/*********************************************************************
* Function:  WORD TeTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input:    pTe-    The pointer to the object where the message will be
*                   evaluated to check if the message will affect the object.
*           pMsg-   Pointer to the message struct containing the message from
*                   the user interface.
*
* Output: 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
*
* Side Effects: none.
*
*********************************************************************/
WORD        TeTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: TeMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG* pMsg)
*
* Overview: This function performs the actual state change
*           based on the translated message given. The following state changes
*           are supported:
*   <TABLE>
*       Translated Message   Input Source   Set/Clear State Bit     Description
*       ##################   ############   ######                  ###########
*       TE_MSG_ADD_CHAR      Touch Screen,  Set TE_UPDATE_TEXT,      Add a character in the
buffer and update the text displayed.
```

```
*                                                   TE_UPDATE_KEY,
*                                               Clear TE_KEY_PRESSED
*       TE_MSG_SPACE         Touch Screen,  Set TE_UPDATE_TEXT,     Insert a space
character in the buffer and update the text displayed.
*                                                   TE_UPDATE_KEY,
*                                               Clear TE_KEY_PRESSED
*       TE_MSG_DELETE        Touch Screen,  Set TE_UPDATE_TEXT,     Delete the most recent
character in the buffer and update the text displayed.
*                                                   TE_UPDATE_KEY,
*                                               Clear TE_KEY_PRESSED
*       TE_MSG_ENTER         Touch Screen,  Set TE_UPDATE_TEXT,     User can define the use
of this event in the message callback. Object will just update the key.
*                                                   TE_UPDATE_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      A Key in the object
will be redrawn in the pressed state.
*                                                   TE_UPDATE_KEY
*
*   </TABLE>
*
* PreCondition: none
*
* Input: translatedMsg - The translated message.
*        pTe           - The pointer to the object whose state will be modified.
*        pMsg          - The pointer to the GOL message.
*
* Output: none
*
* Example:
*   See BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
void        TeMsgDefault(WORD translatedMsg, void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: void TeSetBuffer(TEXTENTRY *pTe, XCHAR *pText, WORD MaxSize)
*
* 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.
*
*
* PreCondition: none
*
* Input:    pTe- pointer to the object
*           pText- pointer to the new text buffer to be displayed
*           maxSize - maximum length of the new buffer to be used.
* Output:  none.
*
* Side Effects: none.
*
*********************************************************************/
void        TeSetBuffer(TEXTENTRY *pTe, XCHAR *pText, WORD MaxSize);

/*********************************************************************
* Macro: TeGetBuffer(pTe)
*
* Overview: This macro will return the currently used buffer in the
*           TextEntry object.
*
* PreCondition: none
*
* Input:    pTe- pointer to the object
*
```

```
* Output:  It will return a pointer to the buffer used.
*
* Side Effects: none.
*
*********************************************************************/
    #define TeGetBuffer(pTe)    (((TEXTENTRY *)pTe)->pTeOutput)

/*********************************************************************
* Function: void TeClearBuffer (TEXTENTRY *pTe)
*
* 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.
*
* PreCondition: none
*
* Input:    pTe- pointer to the object
*
* Output:  none
*
* Side Effects: none.
*
*********************************************************************/
void        TeClearBuffer(TEXTENTRY *pTe);

/*********************************************************************
* Function: BOOL TeIsKeyPressed(TEXTENTRY *pTe, WORD index)
*
* Overview: This function will test if a key given by its index
*           in the TextEntry object has been pressed.
*
* PreCondition: none
*
* Input:    pTe- pointer to the object
*           index- index to the key in the link list
* Output:   Returns a TRUE if the key is pressed. FALSE if key
*           is not pressed or the given index does not exist in
*           the list.
*
* Side Effects: none.
*
*********************************************************************/
BOOL        TeIsKeyPressed(TEXTENTRY *pTe, WORD index);

/*********************************************************************
* Function: void TeSetKeyCommand(TEXTENTRY *pTe,WORD index,WORD command)
*
* Overview: This function will assign a command (TE_DELETE_COM, TE_SPACE_COM
*           or TE_ENTER_COM) to a key with the given index.
*
* PreCondition: none
*
* Input:    pTe  -      pointer to the object
*           index  -    index to the key in the link list
*           command-    command assigned for the key
*
* Output:   Returns TRUE if successful and FALSE if not.
*
* Side Effects: none.
*
*********************************************************************/
BOOL        TeSetKeyCommand(TEXTENTRY *pTe, WORD index, WORD command);

/*********************************************************************
* Function: TeGetKeyCommand(pTe, index)
*
* Overview: This function will return the currently used command by a key
*           with the given index.
*
* PreCondition: none
*
* Input:    pTe- pointer to the object
```

```
*           index- index to the key in the link list
*
* Output:  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
*
* Side Effects: none.
*
*********************************************************************/
WORD        TeGetKeyCommand(TEXTENTRY *pTe, WORD index);

/*********************************************************************
* Function: TeSetKeyText(TEXTENTRY *pTe, WORD index, XCHAR *pText)
*
* Overview: This function will set the test assigned to a key with
*           the given index.
*
* PreCondition: none
*
* Input:    pTe  -      pointer to the object
*           index  -    index to the key in the link list
*           pText -     pointer to the new string to be used
*
* Output:   Returns TRUE if successful and FALSE if not.
*
* Side Effects: none.
*
*********************************************************************/
BOOL TeSetKeyText(TEXTENTRY *pTe, WORD index, XCHAR *pText);

/*********************************************************************
* Function: KEYMEMBER *TeCreateKeyMembers(TEXTENTRY *pTe,XCHAR *pText[])
*
* 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).
*
* PreCondition: none
*
* Input:    pTe  -      pointer to the object
*           pText -     pointer to the text defined by the user
*
* Output: Returns the pointer to the newly created KEYMEMBER list. A NULL is returned
*         if the list is not created succesfully.
*
* Side Effects: none.
*
*********************************************************************/
KEYMEMBER   *TeCreateKeyMembers(TEXTENTRY *pTe, XCHAR *pText[]);

/*********************************************************************
* Function: void TeDelKeyMembers(void *pObj)
*
* Overview: This function will delete the KEYMEMBER list assigned to
*           the object from memory. Pointer to the KEYMEMBER list is
*           then initialized to NULL.
*
* PreCondition: none
*
* Input:    pTe  -  pointer to the object
*
* Output:  none.
*
```

```
* Side Effects: none.
*
*****************************************************************/
void        TeDelKeyMembers(void *pObj);

/*****************************************************************
* Function: void TeSpaceChar(TEXTENTRY *pTe)
*
* 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.
*
* PreCondition: none
*
* Input:    pTe  -  pointer to the object
*
* Output:  none.
*
* Side Effects: none.
*
*****************************************************************/
void        TeSpaceChar(TEXTENTRY *pTe);

/*****************************************************************
* Function: void TeAddChar(TEXTENTRY *pTe)
*
* 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.
*
* PreCondition: none
*
* Input:    pTe  -  pointer to the object
*
* Output:
*
* Side Effects: none.
*
*****************************************************************/
void        TeAddChar(TEXTENTRY *pTe);
#endif // _TEXTENTRY_H
```

14.1.40 GraphicsConfig.h

Macros

Name	Description
COLOR_DEPTH (see page 41)	Specifies the color depth used in the demo.
GFX_free (see page 43)	This is macro GFX_free.
GFX_malloc (see page 43)	Define the malloc() and free() for versatility on OS based systems.
USE_ANALOGCLOCK (see page 36)	Enable Analog Clock Object.
USE_BITMAP_FLASH (see page 43)	Support for bitmaps located in internal flash
USE_BUTTON (see page 32)	Enable Button (see page 65) Object.
USE_BUTTON_MULTI_LINE (see page 32)	Enable Multi-Line (see page 347) Button (see page 65) Object
USE_CHECKBOX (see page 32)	Enable Checkbox (see page 115) Object.
USE_CUSTOM (see page 35)	Enable Custom Control Object (an example to create customized Object).
USE_DIGITALMETER (see page 33)	Enable DigitalMeter Object.
USE_EDITBOX (see page 33)	Enable Edit Box Object.
USE_FOCUS (see page 31)	Keyboard control on some objects can be used by enabling the GOL (see page 48) Focus (USE_FOCUS)support.

USE_FONT_FLASH (see page 42)	Support for fonts located in internal flash
USE_GOL (see page 35)	Enable Graphics Object Layer.
USE_GROUPBOX (see page 33)	Enable Group Box Object.
USE_KEYBOARD (see page 30)	Enable key board support.
USE_LISTBOX (see page 33)	Enable List Box Object.
USE_METER (see page 33)	Enable Meter (see page 196) Object.
USE_MULTIBYTECHAR (see page 37)	To enable support for unicode fonts, USE_MULTIBYTECHAR must be defined. This changes XCHAR (see page 345) definition. See XCHAR (see page 345) for details.
USE_NONBLOCKING_CONFIG (see page 42)	Comment this line to use blocking configuration
USE_PICTURE (see page 34)	Enable Picture (see page 210) Object.
USE_PROGRESSBAR (see page 34)	Enable Progress Bar (see page 352) Object.
USE_RADIOBUTTON (see page 34)	Enable Radio Button (see page 65) Object.
USE_ROUNDDIAL (see page 34)	Enable Dial (see page 124) Object.
USE_SLIDER (see page 35)	Enable Slider (see page 237) or Scroll (see page 419) Bar (see page 352) Object.
USE_STATICTEXT (see page 35)	Enable Static Text Object.
USE_TEXTENTRY (see page 36)	Enable TextEntry Object.
USE_TOUCHSCREEN (see page 30)	Enable touch screen support.
USE_WINDOW (see page 35)	Enable Window (see page 278) Object.

Description

This is file GraphicsConfig.h.

Body Source

```
/*****************************************************************************
 * Module for Microchip Graphics Library
 * This file contains compile time options for the Graphics Library.
 *****************************************************************************
 * FileName:        GraphicsConfig.h
 * Dependencies:    none
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        C30 V3.00/C32
 * Company:         Microchip Technology, Inc.
 *
 * 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.
 *
 * Author                   Date            Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
 * Anton Alkhimenok         10/28/2007      Initial Version
 * Pradeep Budagutta        10/28/2007      Display related defines
 *                                          moved to DisplayConfig.h
 ********************************************************************/
#ifndef _GRAPHICSCONFIG_H
    #define _GRAPHICSCONFIG_H

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

/*********************************************************************
* 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 USE_NONBLOCKING_CONFIG  // Comment this line to use blocking configuration

#if defined(__PIC24FJ256DA210__)
/*********************************************************************
* Overview: Use of double buffering scheme can be enabled/disabled here.
*           If enabled, updated screen areas will vissibly update at once
*           but some continuous updates requiring an update of a
*           large area every time may slow down.
*
*********************************************************************/
//    #define USE_DOUBLE_BUFFERING  // If enabled, position the JP11 jumper on the
PIC24FJ256DA210 development board to 2-3
#endif

/*********************************************************************
* Overview: Using Palettes, different colors can be used with the same
*           bit depth.
*
*********************************************************************/
//    #define USE_PALETTE // Comment this line if Palette is not required.

    #ifdef USE_PALETTE
        //#define USE_PALETTE_EXTERNAL // Uncomment this line if Palette must be stored in
External Memory.
    #endif

/*********************************************************************
* Overview: Keyboard control on some objects can be used by enabling
*           the GOL Focus (USE_FOCUS)support.
*
*********************************************************************/

#define USE_FOCUS

/*********************************************************************
* Overview: Input devices used defines the messages that Objects will
*           process. The following definitions indicate the usage of
*           the different input device:
*           - USE_TOUCHSCREEN - enables the touch screen support.
*           - USE_KEYBOARD - enables the key board support.
*
*********************************************************************/

#define USE_TOUCHSCREEN         // Enable touch screen support.
#define USE_KEYBOARD            // Enable key board support.

/*********************************************************************
* Overview: To save program memory, unused Widgets or Objects can be
*           removed at compile time.
*
*********************************************************************/
#define USE_GOL                 // Enable Graphics Object Layer.
#define USE_BUTTON              // Enable Button Object.
#define USE_BUTTON_MULTI_LINE   // Enable Multi-Line Button 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_ROUNDDIAL           // Enable Dial Object.
#define USE_METER               // Enable Meter Object.
#define USE_DIGITALMETER        // Enable DigitalMeter Object.
#define USE_TEXTENTRY           // Enable TextEntry Object.
#define USE_CUSTOM              // Enable Custom Control Object (an example to create
customized Object).
#define USE_ANALOGCLOCK         // Enable Analog Clock Object.

/*********************************************************************
* Overview: To enable support for unicode fonts, USE_MULTIBYTECHAR
*           must be defined. This changes XCHAR definition. See XCHAR
*           for details.
*
*********************************************************************/

#define USE_MULTIBYTECHAR

/*********************************************************************
* 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.
*   - USE_FONT_FLASH - Font in internal flash memory support.
*   - USE_FONT_EXTERNAL - Font in external memory support.
*
*********************************************************************/
    #define USE_FONT_FLASH          // Support for fonts located in internal flash

//#define USE_FONT_EXTERNAL     // Support for fonts located in external memory

/*********************************************************************
* 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.
*   - USE_BITMAP_FLASH - Font in internal flash memory support.
*   - USE_BITMAP_EXTERNAL - Font in external memory support.
*
*********************************************************************/
    #define USE_BITMAP_FLASH        // Support for bitmaps located in internal flash

//#define USE_BITMAP_EXTERNAL       // Support for bitmaps located in external memory


/*********************************************************************
* Overview: Define the malloc() and free() for versatility on OS
*           based systems.
*
*********************************************************************/
    #define GFX_malloc(size)        malloc(size)
    #define GFX_free(pObj)          free(pObj)

/*********************************************************************
* Overview: Specifies the color depth used in the demo.
*
*********************************************************************/

    #define COLOR_DEPTH             16

#endif // _GRAPHICSCONFIG_H
```

14.1.41 Window.c

This is file Window.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Window
 *****************************************************************************
 * FileName:        Window.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30 V3.00, 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Anton Alkhimenok     11/12/07    Version 1.0 release
 *****************************************************************************/
#include "Graphics/Graphics.h"

#ifdef USE_WINDOW

/*******************************************************************
* Function: WINDOW  *WndCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                              SHORT bottom, WORD state, XCHAR *pText, void* pBitmap,
*                              GOL_SCHEME *pScheme)
*
* Overview: creates the window
*
*******************************************************************/
WINDOW *WndCreate
(
    WORD        ID,
    SHORT       left,
    SHORT       top,
    SHORT       right,
    SHORT       bottom,
    WORD        state,
    void        *pBitmap,
```

```
    XCHAR       *pText,
    GOL_SCHEME  *pScheme
)
{
    WINDOW  *pW;

    pW = (WINDOW *)GFX_malloc(sizeof(WINDOW));
    if(pW == NULL)
        return (pW);

    pW->hdr.ID = ID;
    pW->hdr.pNxtObj = NULL;
    pW->hdr.type = OBJ_WINDOW;
    pW->hdr.left = left;
    pW->hdr.top = top;
    pW->hdr.right = right;
    pW->hdr.bottom = bottom;
    pW->pBitmap = pBitmap;
    pW->pText = pText;
    pW->hdr.state = state;
    pW->hdr.DrawObj = WndDraw;          // draw function
    pW->hdr.MsgObj = WndTranslateMsg;   // message function
    pW->hdr.MsgDefaultObj = NULL;       // default message function
    pW->hdr.FreeObj = NULL;             // free function

    // Set the style scheme to be used
    if(pScheme == NULL)
        pW->hdr.pGolScheme = _pDefaultGolScheme;
    else
        pW->hdr.pGolScheme = pScheme;

    pW->textHeight = 0;
    if(pText != NULL)
    {
        pW->textHeight = GetTextHeight(pW->hdr.pGolScheme->pFont);
    }

    GOLAddObject((OBJ_HEADER *)pW);

    return (pW);
}

/*********************************************************************
* Function: WndSetText(WINDOW *pW, XCHAR *pText)
*
* Overview: sets text
*
*********************************************************************/
void WndSetText(WINDOW *pW, XCHAR *pText)
{
    pW->pText = pText;
    pW->textHeight = GetTextHeight(pW->hdr.pGolScheme->pFont);
}

/*********************************************************************
* Function: WORD WndTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* Overview: translates the GOL message for the window
*
*********************************************************************/
WORD WndTranslateMsg(void *pObj, GOL_MSG *pMsg)
{
    WINDOW *pW;

    pW = (WINDOW *)pObj;

    // Evaluate if the message is for the window
    // Check if disabled first
    if(GetState(pW, WND_DISABLED))
        return (OBJ_MSG_INVALID);

        #ifdef USE_TOUCHSCREEN
```

```
    if(pMsg->type == TYPE_TOUCHSCREEN)
    {

        // Check if it falls in the title bar area
        if
        (
            (pW->hdr.left < pMsg->param1) &&
            (pW->hdr.right > pMsg->param1) &&
            (pW->hdr.top < pMsg->param2) &&
            (pW->hdr.top + WND_TITLE_HEIGHT + GOL_EMBOSS_SIZE > pMsg->param2)
        )
        {
            return (WND_MSG_TITLE);
        }

        // Check if it falls in the client area
        if
        (
            (pW->hdr.left + GOL_EMBOSS_SIZE < pMsg->param1) &&
            (pW->hdr.right - GOL_EMBOSS_SIZE > pMsg->param1) &&
            (pW->hdr.top + WND_TITLE_HEIGHT + GOL_EMBOSS_SIZE < pMsg->param2) &&
            (pW->hdr.bottom - GOL_EMBOSS_SIZE > pMsg->param2)
        )
        {
            return (WND_MSG_CLIENT);
        }
    }

        #endif
    return (OBJ_MSG_INVALID);
}

/*********************************************************************
* Function: WORD WndDraw(void *pObj)
*
* Overview: draws window
*
*********************************************************************/
WORD WndDraw(void *pObj)
{
    typedef enum
    {
        WND_REMOVE,
        WND_TITLE_BAR_DRAW,
        WND_TITLE_BAR_BITMAP,
        WND_TITLE_BAR_TEXT,
        WND_TITLE_BAR_TEXT_DRAW,
        WND_CLIENT,
        WND_CLIENT_DRAW
    } WND_DRAW_STATES;

    SHORT temp;
    WINDOW *pW;
    static WND_DRAW_STATES state = WND_REMOVE;

    pW = (WINDOW *)pObj;

    while(1)
    {
        if(IsDeviceBusy())
            return (0);

        switch(state)
        {
            case WND_REMOVE:
                if(GetState(pW, WND_HIDE))
                {
                    SetColor(pW->hdr.pGolScheme->CommonBkColor);
                    if(!Bar(pW->hdr.left, pW->hdr.top, pW->hdr.right, pW->hdr.bottom))
                    {
                        return (0);
                    }
```

```
            return (1);
        }

        if(GetState(pW, WND_DRAW_CLIENT))
        {
            state = WND_CLIENT;
            break;

        case WND_CLIENT:
            SetLineThickness(NORMAL_LINE);
            SetLineType(SOLID_LINE);
            GOLPanelDraw
            (
                pW->hdr.left,
                pW->hdr.top,
                pW->hdr.right,
                pW->hdr.bottom,
                0,
                pW->hdr.pGolScheme->CommonBkColor,
                pW->hdr.pGolScheme->EmbossLtColor,
                pW->hdr.pGolScheme->EmbossDkColor,
                NULL,
                GOL_EMBOSS_SIZE
            );

            state = WND_CLIENT_DRAW;

        case WND_CLIENT_DRAW:
            if(!GOLPanelDrawTsk())
                return (0);
        }

        state = WND_TITLE_BAR_DRAW;
        break;

    case WND_TITLE_BAR_DRAW:
        if(!GetState(pW, WND_DISABLED))
        {
            if(GetState(pW, WND_FOCUSED))
            {
                SetColor(pW->hdr.pGolScheme->Color1);
            }
            else
            {
                SetColor(pW->hdr.pGolScheme->Color0);
            }
        }
        else
        {
            SetColor(pW->hdr.pGolScheme->ColorDisabled);
        }

        if
        (
            !Bar
                (
                    pW->hdr.left + GOL_EMBOSS_SIZE,
                    pW->hdr.top + GOL_EMBOSS_SIZE,
                    pW->hdr.right - GOL_EMBOSS_SIZE,
                    pW->hdr.top + GOL_EMBOSS_SIZE + WND_TITLE_HEIGHT
                )
        )
        {
            return (0);
        }

        state = WND_TITLE_BAR_BITMAP;

        break;

    case WND_TITLE_BAR_BITMAP:
```

```
                if(pW->pBitmap != NULL)
                {
                    if
                    (
                        !PutImage
                            (
                                pW->hdr.left + GOL_EMBOSS_SIZE,
                                pW->hdr.top + GOL_EMBOSS_SIZE + ((WND_TITLE_HEIGHT -
GetImageHeight(pW->pBitmap)) >> 1),
                                pW->pBitmap,
                                IMAGE_NORMAL
                            )
                    )
                    {
                        return (0);
                    }
                }

                if(pW->pText != NULL)
                {
                    state = WND_TITLE_BAR_TEXT;
                    break;

                case WND_TITLE_BAR_TEXT:
                    SetFont(pW->hdr.pGolScheme->pFont);

                    if(!GetState(pW, WND_DISABLED))
                    {
                        if(GetState(pW, WND_FOCUSED))
                        {
                            SetColor(pW->hdr.pGolScheme->TextColor1);
                        }
                        else
                        {
                            SetColor(pW->hdr.pGolScheme->TextColor0);
                        }
                    }
                    else
                    {
                        SetColor(pW->hdr.pGolScheme->TextColorDisabled);
                    }

                    temp = pW->hdr.left + GOL_EMBOSS_SIZE + WND_INDENT;

                    if(pW->pBitmap != NULL)
                    {
                        temp += GetImageWidth(pW->pBitmap);
                    }

                    if(GetState(pW, WND_TITLECENTER))
                    {
                        temp = (temp + (pW->hdr.right - GetTextWidth(pW->pText,
pW->hdr.pGolScheme->pFont))) >> 1;
                    }

                    MoveTo(temp, pW->hdr.top + GOL_EMBOSS_SIZE + ((WND_TITLE_HEIGHT -
pW->textHeight) >> 1));

                    state = WND_TITLE_BAR_TEXT_DRAW;

                case WND_TITLE_BAR_TEXT_DRAW:
                    if(!OutText(pW->pText))
                        return (0);
                }

                state = WND_REMOVE;
                return (1);
        }   //end of switch
    }       //end of while
}

#endif // USE_WINDOW
```

14.1.42 Window.h

Functions

	Name	Description
	WndCreate (see page 281)	This function creates a WINDOW (see page 285) 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 (see page 282)	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 (see page 283)	This function sets the string used for the title bar.
	WndTranslateMsg (see page 284)	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
WND_DISABLED (see page 279)	Bit for disabled state
WND_DRAW (see page 280)	Bits to indicate whole window must be redrawn
WND_DRAW_CLIENT (see page 280)	Bit to indicate client area must be redrawn
WND_DRAW_TITLE (see page 280)	Bit to indicate title area must be redrawn
WND_FOCUSED (see page 280)	Bit for focus state
WND_HIDE (see page 280)	Bit to indicate window must be removed from screen
WND_TITLECENTER (see page 281)	Bit to center the text on the Title Area
WndGetText (see page 283)	This macro returns the address of the current text string used for the title bar.

Structures

Name	Description
WINDOW (see page 285)	The structure contains data for the window

Description

This is file Window.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  GOL Layer
 *  Window
 *****************************************************************************
 * FileName:        Window.h
 * Dependencies:    None
 * Processor:       PIC24F, PIC24H, dsPIC, 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 ****************************************************************************/
#ifndef _WINDOW_H
    #define _WINDOW_H

    #include <Graphics/GOL.h>
    #include "GenericTypeDefs.h"

// Indent for the title bar text from the left side of bitmap or title bar emboss
    #define WND_INDENT  2

// Height of the title bar
    #define WND_TITLE_HEIGHT    35

/*********************************************************************
* Object States Definition:
*********************************************************************/
    #define WND_FOCUSED     0x0001  // Bit for focus state
    #define WND_DISABLED    0x0002  // Bit for disabled state
    #define WND_TITLECENTER 0x0004  // Bit to center the text on the Title Area
    #define WND_HIDE        0x8000  // Bit to indicate window must be removed from screen
    #define WND_DRAW_CLIENT 0x4000  // Bit to indicate client area must be redrawn
    #define WND_DRAW_TITLE  0x2000  // Bit to indicate title area must be redrawn
    #define WND_DRAW        0x6000  // Bits to indicate whole window must be redrawn

/*****************************************************************************
 * Overview: The structure contains data for the window
 *****************************************************************************/
typedef struct
{
    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
} WINDOW;

/*********************************************************************
* Function: WINDOW  *WndCreate(WORD ID, SHORT left, SHORT top, SHORT right,
*                              SHORT bottom, WORD state,  void* pBitmap, XCHAR* pText,
*                              GOL_SCHEME *pScheme)
*
* 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.
*
* PreCondition: none
*
* Input: ID - Unique user defined ID for the object instance.
*        left - Left most position of the Object.
*        top - Top most position of the Object.
```

```
*        right - Right most position of the Object.
*        bottom - Bottom most position of the object.
*        state - Sets the initial state of the object.
*        pBitmap - Pointer to the bitmap used in the title bar.
*        pText - Pointer to the text used as a title of the window.
*        pScheme - Pointer to the style scheme used.
*
* Output: Returns the pointer to the object created
*
* Example:
*   <CODE>
*   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;
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
WINDOW  *WndCreate
        (
            WORD        ID,
            SHORT       left,
            SHORT       top,
            SHORT       right,
            SHORT       bottom,
            WORD        state,
            void        *pBitmap,
            XCHAR       *pText,
            GOL_SCHEME  *pScheme
        );

/*********************************************************************
* Macros:  WndGetText(pW)
*
* Overview: This macro returns the address of the current
*           text string used for the title bar.
*
* PreCondition: none
*
* Input: pW - Pointer to the object
*
* Output: Returns pointer to the text string being used.
*
* Example:
*   <CODE>
*   WINDOW *pWindow;
*   XCHAR textUsed = “USE THIS!”;
*
*       if (WndGetText(pWindow) == NULL)
*           WndSetText(&textUsed);
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
    #define WndGetText(pW)  pW->pText

/*********************************************************************
* Function: WndSetText(WINDOW *pW, XCHAR *pText)
*
* Overview: This function sets the string used for the title bar.
*
```

```
* PreCondition: none
*
* Input: pW - The pointer to the object whose text will be modified
*        pText - Pointer to the text that will be used
*
* Output: none
*
* Example:
*  See WndGetText() example.
*
* Side Effects: none
*
*********************************************************************/
void    WndSetText(WINDOW *pW, XCHAR *pText);

/*********************************************************************
* Function: WORD WndTranslateMsg(void *pObj, GOL_MSG *pMsg)
*
* 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.
*
*   <TABLE>
*       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.
*   </TABLE>
*
* PreCondition: none
*
* Input: pW    - The pointer to the object where the message will be
*                evaluated to check if the message will affect the object.
*        pMsg  - Pointer to the message struct containing the message from
*                the user interface.
*
* Output: 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
*
* Example:
*   Usage is similar to BtnTranslateMsg() example.
*
* Side Effects: none
*
*********************************************************************/
WORD    WndTranslateMsg(void *pObj, GOL_MSG *pMsg);

/*********************************************************************
* Function: WORD WndDraw(void *pObj)
*
* 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.
*
* PreCondition: Object must be created before this function is called.
```

```
 *
 * Input: pW - Pointer to the object to be rendered.
 *
 * Output: 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.
 *
 * Example:
 *   See WndCreate() example.
 *
 * Side Effects: none
 *
 ********************************************************************/
WORD WndDraw(void *pObj);
#endif // _WINDOW_H
```

14.2 Primitive Layer

Files

Name	Description
Primitive.c (see page 941)	This is file Primitive.c.
Primitive.h (see page 1008)	This is file Primitive.h.

Description

Lists all files describing the Primitive Layer.

14.2.1 Primitive.c

This is file Primitive.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Graphic Primitives Layer
 *****************************************************************************
 * FileName:        Primitive.c
 * Processor:       PIC24, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2011 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 05/11/10     Added dynamic Arc() where start and end
 *                                angle can be specified.
 * 03/04/11     - Removed SetColor(WHITE) in InitGraph(). Default color
 *                after InitGraph() is not 0 (BLACK in most displays).
 *              - removed USE_DRV_XX checks, replaced them with
 *                weak attributes.
 * 05/13/11     Add Transparent Color support in PutImage() and PutImageRLE()
 *              functions defined in this layer.
 * 05/20/11     Added GetCirclePoint() commonly used in Widgets.
 *
 ****************************************************************************/
#include "HardwareProfile.h"              // needed to provide values for GetMaxX() and
GetMaxY() macros
#include "Graphics/DisplayDriver.h"
#include "Graphics/Primitive.h"
#include "Compiler.h"

//////////////////////// LOCAL FUNCTIONS PROTOTYPES ////////////////////////////
#ifdef USE_BITMAP_FLASH
    void    PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
    void    PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
    void    PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
    void    PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);

#ifdef USE_COMP_RLE
    void    PutImageRLE4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
    void    PutImageRLE8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
#endif
#endif

#ifdef USE_BITMAP_EXTERNAL
    void    PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
    void    PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
    void    PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
    void    PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);

#ifdef USE_COMP_RLE
    void    PutImageRLE4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
    void    PutImageRLE8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
#endif
#endif

// Current line type
SHORT   _lineType;

// Current line thickness
BYTE    _lineThickness;

// Font orientation
BYTE    _fontOrientation;

// Current cursor x-coordinates
SHORT   _cursorX;

// Current cursor y-coordinates
SHORT   _cursorY;

// Pointer to the current font
void    *_font;

#ifdef USE_PALETTE
void    *_palette;
#endif
```

```
// First and last characters in the font
WORD    _fontFirstChar; // First character in the font table.
WORD    _fontLastChar;  // Last character in the font table.

// Installed font height
SHORT   _fontHeight;

// bevel drawing type (0 = full bevel, 0xF0 - top bevel only, 0x0F - bottom bevel only
BYTE _bevelDrawType;


#define COSINETABLEENTRIES  90
// Cosine table used to calculate angles when rendering circular objects and  arcs
// Make cosine values * 256 instead of 100 for easier math later
const SHORT   _CosineTable[COSINETABLEENTRIES+1] __attribute__((aligned(2))) =
                        {
                            256, 256, 256, 256, 255, 255, 255, 254, 254, 253,
                            252, 251, 250, 249, 248, 247, 246, 245, 243, 242,
                            241, 239, 237, 236, 234, 232, 230, 228, 226, 224,
                            222, 219, 217, 215, 212, 210, 207, 204, 202, 199,
                            196, 193, 190, 187, 184, 181, 178, 175, 171, 168,
                            165, 161, 158, 154, 150, 147, 143, 139, 136, 132,
                            128, 124, 120, 116, 112, 108, 104, 100, 96,  92,
                            88,  83,  79,  75,  71,  66,  62,  58,  53,  49,
                            44,  40,  36,  31,  27,  22,  18,  13,  9,   4,
                            0
                        };

/*****************************************************************************
* Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* PreCondition: none
*

* Input: left,top - top left corner coordinates,
*        right,bottom - bottom right corner coordinates
*
* Output: 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.
*
* Side Effects: none
*
* Overview: draws rectangle filled with current color
*
* Note: none
*
*****************************************************************************/
WORD __attribute__((weak)) Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    SHORT   x, y;

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    /* Ready */
        #else
    if(IsDeviceBusy() != 0)
        return (0);
        #endif
    for(y = top; y < bottom + 1; y++)
        for(x = left; x < right + 1; x++)
            PutPixel(x, y);

    return (1);
}

/*****************************************************************************
* Function: WORD Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
```

```
 *
 * PreCondition: none
 *
 * Input: x1,y1 - starting coordinates, x2,y2 - ending coordinates
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 * Overview: draws line
 *
 * Note: none
 *
 ***************************************************************************/
WORD __attribute__((weak)) Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
{
    SHORT   deltaX, deltaY;
    SHORT   error, stepErrorLT, stepErrorGE;
    SHORT   stepX, stepY;
    SHORT   steep;
    SHORT   temp;
    SHORT   style, type;

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    /* Ready */
        #else
    if(IsDeviceBusy() != 0)
        return (0);
        #endif

    // Move cursor
    MoveTo(x2, y2);

    if(x1 == x2)
    {
        if(y1 > y2)
        {
            temp = y1;
            y1 = y2;
            y2 = temp;
        }

        style = 0;
        type = 1;
        for(temp = y1; temp < y2 + 1; temp++)
        {
            if((++style) == _lineType)
            {
                type ^= 1;
                style = 0;
            }

            if(type)
            {
                PutPixel(x1, temp);
                if(_lineThickness)
                {
                    PutPixel(x1 + 1, temp);
                    PutPixel(x1 - 1, temp);
                }
            }
        }

        return (1);
    }
```

```
    if(y1 == y2)
    {
        if(x1 > x2)
        {
            temp = x1;
            x1 = x2;
            x2 = temp;
        }

        style = 0;
        type = 1;
        for(temp = x1; temp < x2 + 1; temp++)
        {
            if((++style) == _lineType)
            {
                type ^= 1;
                style = 0;
            }

            if(type)
            {
                PutPixel(temp, y1);
                if(_lineThickness)
                {
                    PutPixel(temp, y1 + 1);
                    PutPixel(temp, y1 - 1);
                }
            }
        }

        return (1);
    }

    stepX = 0;
    deltaX = x2 - x1;
    if(deltaX < 0)
    {
        deltaX = -deltaX;
        --stepX;
    }
    else
    {
        ++stepX;
    }

    stepY = 0;
    deltaY = y2 - y1;
    if(deltaY < 0)
    {
        deltaY = -deltaY;
        --stepY;
    }
    else
    {
        ++stepY;
    }

    steep = 0;
    if(deltaX < deltaY)
    {
        ++steep;
        temp = deltaX;
        deltaX = deltaY;
        deltaY = temp;
        temp = x1;
        x1 = y1;
        y1 = temp;
        temp = stepX;
        stepX = stepY;
        stepY = temp;
        PutPixel(y1, x1);
    }
```

```
    else
    {
        PutPixel(x1, y1);
    }

    // If the current error greater or equal zero
    stepErrorGE = deltaX << 1;

    // If the current error less than zero
    stepErrorLT = deltaY << 1;

    // Error for the first pixel
    error = stepErrorLT - deltaX;

    style = 0;
    type = 1;

    while(--deltaX >= 0)
    {
        if(error >= 0)
        {
            y1 += stepY;
            error -= stepErrorGE;
        }

        x1 += stepX;
        error += stepErrorLT;

        if((++style) == _lineType)
        {
            type ^= 1;
            style = 0;
        }

        if(type)
        {
            if(steep)
            {
                PutPixel(y1, x1);
                if(_lineThickness)
                {
                    PutPixel(y1 + 1, x1);
                    PutPixel(y1 - 1, x1);
                }
            }
            else
            {
                PutPixel(x1, y1);
                if(_lineThickness)
                {
                    PutPixel(x1, y1 + 1);
                    PutPixel(x1, y1 - 1);
                }
            }
        }
    }   // end of while

    return (1);
}

/*****************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: clears screen with current color and sets cursor to 0,0
```

```
*
* Note: none
*
*****************************************************************/
void __attribute__((weak)) ClearDevice(void)
{
    while(Bar(0, 0, GetMaxX(), GetMaxY()) == 0);
    MoveTo(0, 0);
}

/*****************************************************************
* Function:  void InitGraph(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: initializes LCD controller,
*           sets cursor position to upper left corner,
*           sets active and visual pages to page 0,
*           clears active page with BLACK,
*           sets color to WHITE,
*           disables clipping
*
* Note: none
*
*****************************************************************/
void InitGraph(void)
{

    // Current line type
    SetLineType(SOLID_LINE);

    // Current line thickness
    SetLineThickness(NORMAL_LINE);

    // Current cursor coordinates to 0,0
    MoveTo(0, 0);

    // Reset device
    ResetDevice();

    // Set color to BLACK
    SetColor(0);

    // set the transparent color check to be disabled
#ifdef USE_TRANSPARENT_COLOR
    TransparentColorDisable();
#endif

    // Clear screen
    ClearDevice();

    // Disable clipping
    SetClip(CLIP_DISABLE);

    // Set font orientation
    SetFontOrientation(ORIENT_HOR);

    // set Bevel drawing
    SetBevelDrawType(DRAWFULLBEVEL);
}

/*****************************************************************
* Function: WORD Arc(SHORT xL, SHORT yT, SHORT xR, SHORT yB, SHORT r1, SHORT r2, BYTE
octant);
*
* PreCondition: none
```

```
*
* Input: xL, yT - location of the upper left center in the x,y coordinate
*        xR, yB - location of the lower right left center in the x,y coordinate
*        r1, r2 - the two concentric circle radii, r1 as the radius
*                 of the smaller circle and and r2 as the radius of the
*                 larger circle.
*        octant - bitmask of the octant that will be drawn.
*                 Moving in a clockwise direction from x = 0, y = +radius
*                 bit0 : first octant   bit4 : fifth octant
*                 bit1 : second octant  bit5 : sixth octant
*                 bit2 : third octant   bit6 : seventh octant
*                 bit3 : fourth octant  bit7 : eigth octant
*
* Output: 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.
*
* Side Effects: none
*
* Overview: Draws the octant arc of a beveled figure with given centers, radii
*           and octant mask. When r1 is zero and r2 has some value, a filled
*           circle is drawn; when the radii have values, an arc of
*           thickness (r2-r1) is drawn; when octant = 0xFF, a full ring
*           is drawn. When r1 and r2 are zero, a rectangular object is drawn, where
*           xL, yT specifies the left top corner; xR, yB specifies the right bottom
*           corner.
*
* Note: none
*
*********************************************************************/
WORD __attribute__((weak)) Arc(SHORT xL, SHORT yT, SHORT xR, SHORT yB, SHORT r1, SHORT r2,
BYTE octant)
{

    // this is using a variant of the Midpoint (Bresenham's) Algorithm
    #ifndef USE_NONBLOCKING_CONFIG

    SHORT       y1Limit, y2Limit;
    SHORT       x1, x2, y1, y2;
    SHORT       err1, err2;
    SHORT       x1Cur, y1Cur, y1New;
    SHORT       x2Cur, y2Cur, y2New;
    DWORD_VAL   temp;

    temp.Val = SIN45 * r1;
    y1Limit = temp.w[1];
    temp.Val = SIN45 * r2;
    y2Limit = temp.w[1];

    temp.Val = (DWORD) (ONEP25 - ((LONG) r1 << 16));
    err1 = (SHORT) (temp.w[1]);

    temp.Val = (DWORD) (ONEP25 - ((LONG) r2 << 16));
    err2 = (SHORT) (temp.w[1]);

    x1 = r1;
    x2 = r2;
    y1 = 0;
    y2 = 0;

    x1Cur = x1;
    y1Cur = y1;
    y1New = y1;
    x2Cur = x2;
    y2Cur = y2;
    y2New = y2;

    while(y2 <= y2Limit)
    {   // just watch for y2 limit since outer circle
        // will have greater value.
```

```
        // Drawing of the rounded panel is done only when there is a change in the
        // x direction. Bars are drawn to be efficient.
        // detect changes in the x position. Every change will mean a bar will be drawn
        // to cover the previous area. y1New records the last position of y before the
        // change in x position.
        // y1New & y2New records the last y positions, must remember this to
        // draw the correct bars (non-overlapping).
        y1New = y1;
        y2New = y2;

        if(y1 <= y1Limit)
        {
            if(err1 > 0)
            {
                x1--;
                err1 += 5;
                err1 += (y1 - x1) << 1;
            }
            else
            {
                err1 += 3;
                err1 += y1 << 1;
            }

            y1++;
        }
        else
        {
            y1++;
            if(x1 < y1)
                x1 = y1;
        }

        if(err2 > 0)
        {
            x2--;
            err2 += 5;
            err2 += (y2 - x2) << 1;
        }
        else
        {
            err2 += 3;
            err2 += y2 << 1;
        }

        y2++;

        if((x1Cur != x1) || (x2Cur != x2))
        {
            if(octant & 0x01)
            {
                Bar(xR + y2Cur, yT - x2Cur, xR + y1New, yT - x1Cur);    // 1st octant
            }

            if(octant & 0x02)
            {
                Bar(xR + x1Cur, yT - y1New, xR + x2Cur, yT - y2Cur);    // 2nd octant
            }

            if(octant & 0x04)
            {
                Bar(xR + x1Cur, yB + y1Cur, xR + x2Cur, yB + y2New);    // 3rd octant
            }

            if(octant & 0x08)
            {
                Bar(xR + y1Cur, yB + x1Cur, xR + y2New, yB + x2Cur);    // 4th octant
            }

            if(octant & 0x10)
            {
                Bar(xL - y1New, yB + x1Cur, xL - y2Cur, yB + x2Cur);    // 5th octant
```

```
            }

            if(octant & 0x20)
            {
                Bar(xL - x2Cur, yB + y2Cur, xL - x1Cur, yB + y1New);    // 6th octant
            }

            if(octant & 0x40)
            {
                Bar(xL - x2Cur, yT - y2New, xL - x1Cur, yT - y1Cur);    // 7th octant
            }

            if(octant & 0x80)
            {
                Bar(xL - y2New, yT - x2Cur, xL - y1Cur, yT - x1Cur);    // 8th octant
            }

            // update current values
            x1Cur = x1;
            y1Cur = y1;
            x2Cur = x2;
            y2Cur = y2;
        }
    }                           // end of while loop

    // draw the width and height
    if((xR - xL) || (yB - yT))
    {

        // draw right
        if(octant & 0x02)
        {
            Bar(xR + r1, yT, xR + r2, (yB + yT) >> 1);
        }

        if(octant & 0x04)
        {
            Bar(xR + r1, ((yB + yT) >> 1), xR + r2, yB);
        }

        // draw bottom
        if(octant & 0x10)
        {
            Bar(xL, yB + r1, ((xR + xL) >> 1), yB + r2);
        }

        if(octant & 0x08)
        {
            Bar(((xR + xL) >> 1), yB + r1, xR, yB + r2);
        }

        if(xR - xL)
        {

            // draw top
            if(octant & 0x80)
            {
                Bar(xL, yT - r2, ((xR + xL) >> 1), yT - r1);
            }

            if(octant & 0x01)
            {
                Bar(((xR + xL) >> 1), yT - r2, xR, yT - r1);
            }
        }

        if(yT - yB)
        {

            // draw left
            if(octant & 0x40)
            {
```

```
                Bar(xL - r2, yT, xL - r1, ((yB + yT) >> 1));
            }

            if(octant & 0x20)
            {
                Bar(xL - r2, ((yB + yT) >> 1), xL - r1, yB);
            }
        }
    }

    return (1);
    #else

    typedef enum
    {
        BEGIN,
        QUAD11,
        BARRIGHT1,
        QUAD12,
        BARRIGHT2,
        QUAD21,
        BARLEFT1,
        QUAD22,
        BARLEFT2,
        QUAD31,
        BARTOP1,
        QUAD32,
        BARTOP2,
        QUAD41,
        BARBOTTOM1,
        QUAD42,
        BARBOTTOM2,
        CHECK,
    } OCTANTARC_STATES;

    DWORD_VAL temp;

    //  LONG temp1;
    static SHORT y1Limit, y2Limit;
    static SHORT x1, x2, y1, y2;
    static SHORT err1, err2;
    static SHORT x1Cur, y1Cur, y1New;
    static SHORT x2Cur, y2Cur, y2New;
    static OCTANTARC_STATES state = BEGIN;

    while(1)
    {
        if(IsDeviceBusy())
            return (0);
        switch(state)
        {
            case BEGIN:
                temp.Val = SIN45 * r1;
                y1Limit = temp.w[1];
                temp.Val = SIN45 * r2;
                y2Limit = temp.w[1];

                temp.Val = (DWORD) (ONEP25 - ((LONG) r1 << 16));
                err1 = (SHORT) (temp.w[1]);

                temp.Val = (DWORD) (ONEP25 - ((LONG) r2 << 16));
                err2 = (SHORT) (temp.w[1]);

                x1 = r1;
                x2 = r2;
                y1 = 0;
                y2 = 0;

                x1Cur = x1;
                y1Cur = y1;
                y1New = y1;
                x2Cur = x2;
```

```
            y2Cur = y2;
            y2New = y2;
            state = CHECK;

        case CHECK:
            arc_check_state : if(y2 > y2Limit)
            {
                state = BARRIGHT1;
                goto arc_draw_width_height_state;
            }

            // y1New & y2New records the last y positions
            y1New = y1;
            y2New = y2;

            if(y1 <= y1Limit)
            {
                if(err1 > 0)
                {
                    x1--;
                    err1 += 5;
                    err1 += (y1 - x1) << 1;
                }
                else
                {
                    err1 += 3;
                    err1 += y1 << 1;
                }

                y1++;
            }
            else
            {
                y1++;
                if(x1 < y1)
                    x1 = y1;
            }

            if(err2 > 0)
            {
                x2--;
                err2 += 5;
                err2 += (y2 - x2) << 1;
            }
            else
            {
                err2 += 3;
                err2 += y2 << 1;
            }

            y2++;

            state = QUAD11;
            break;

        case QUAD11:
            if((x1Cur != x1) || (x2Cur != x2))
            {

                // 1st octant
                if(octant & 0x01)
                {
                    if(Bar(xR + y2Cur, yT - x2Cur, xR + y1New, yT - x1Cur) == 0)
                        return (0);
                }
            }
            else
            {
                state = CHECK;
                goto arc_check_state;
            }
```

```
            state = QUAD12;
            break;

        case QUAD12:

            // 2nd octant
            if(octant & 0x02)
            {
                if(Bar(xR + x1Cur, yT - y1New, xR + x2Cur, yT - y2Cur) == 0)
                    return (0);
            }

            state = QUAD21;
            break;

        case QUAD21:

            // 3rd octant
            if(octant & 0x04)
            {
                if(Bar(xR + x1Cur, yB + y1Cur, xR + x2Cur, yB + y2New) == 0)
                    return (0);
            }

            state = QUAD22;
            break;

        case QUAD22:

            // 4th octant
            if(octant & 0x08)
            {
                if(Bar(xR + y1Cur, yB + x1Cur, xR + y2New, yB + x2Cur) == 0)
                    return (0);
            }

            state = QUAD31;
            break;

        case QUAD31:

            // 5th octant
            if(octant & 0x10)
            {
                if(Bar(xL - y1New, yB + x1Cur, xL - y2Cur, yB + x2Cur) == 0)
                    return (0);
            }

            state = QUAD32;
            break;

        case QUAD32:

            // 6th octant
            if(octant & 0x20)
            {
                if(Bar(xL - x2Cur, yB + y2Cur, xL - x1Cur, yB + y1New) == 0)
                    return (0);
            }

            state = QUAD41;
            break;

        case QUAD41:

            // 7th octant
            if(octant & 0x40)
            {
                if(Bar(xL - x2Cur, yT - y2New, xL - x1Cur, yT - y1Cur) == 0)
                    return (0);
            }
```

```
            state = QUAD42;
            break;

        case QUAD42:

            // 8th octant
            if(octant & 0x80)
            {
                if(Bar(xL - y2New, yT - x2Cur, xL - y1Cur, yT - x1Cur) == 0)
                    return (0);
            }

            // update current values
            x1Cur = x1;
            y1Cur = y1;
            x2Cur = x2;
            y2Cur = y2;
            state = CHECK;
            break;

        case BARRIGHT1:     // draw upper right
            arc_draw_width_height_state : if((xR - xL) || (yB - yT))
            {

                // draw right
                if(octant & 0x02)
                {
                    if(Bar(xR + r1, yT, xR + r2, (yB + yT) >> 1) == 0)
                        return (0);
                }
            }
            else
            {
                state = BEGIN;
                return (1);
            }

            state = BARRIGHT2;
            break;

        case BARRIGHT2:     // draw lower right
            if(octant & 0x04)
            {
                if(Bar(xR + r1, ((yB + yT) >> 1), xR + r2, yB) == 0)
                    return (0);
            }

            state = BARBOTTOM1;
            break;

        case BARBOTTOM1:    // draw left bottom
            // draw bottom
            if(octant & 0x10)
            {
                if(Bar(xL, yB + r1, ((xR + xL) >> 1), yB + r2) == 0)
                    return (0);
            }

            state = BARBOTTOM2;
            break;

        case BARBOTTOM2:    // draw right bottom
            if(octant & 0x08)
            {
                if(Bar(((xR + xL) >> 1), yB + r1, xR, yB + r2) == 0)
                    return (0);
            }

            state = BARTOP1;
            break;

        case BARTOP1:       // draw left top
```

```
                if(xR - xL)
                {

                    // draw top
                    if(octant & 0x80)
                    {
                        if(Bar(xL, yT - r2, ((xR + xL) >> 1), yT - r1) == 0)
                            return (0);
                    }

                    state = BARTOP2;
                }
                else
                    state = BARLEFT1;   // no width go directly to height bar
                break;

            case BARTOP2:               // draw right top
                if(octant & 0x01)
                {
                    if(Bar(((xR + xL) >> 1), yT - r2, xR, yT - r1) == 0)
                        return (0);
                }

                state = BARLEFT1;
                break;

            case BARLEFT1:              // draw upper left
                if(yT - yB)
                {

                    // draw left
                    if(octant & 0x40)
                    {
                        if(Bar(xL - r2, yT, xL - r1, ((yB + yT) >> 1)) == 0)
                            return (0);
                    }

                    state = BARLEFT2;
                }
                else
                {
                    state = BEGIN;      // no height go back to BEGIN
                    return (1);
                }

                break;

            case BARLEFT2:              // draw lower left
                if(octant & 0x20)
                {
                    if(Bar(xL - r2, ((yB + yT) >> 1), xL - r1, yB) == 0)
                        return (0);
                }

                state = BEGIN;
                return (1);
        }                               // end of switch
    }   // end of while
    #endif // USE_NONBLOCKING_CONFIG
}

/*****************************************************************************
* Function: WORD Bevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
*
* PreCondition: None
*
* Input: x1, y1 - coordinate position of the upper left center of the
*                 circle that draws the rounded corners,
*        x2, y2 - coordinate position of the lower right center of the
*                 circle that draws the rounded corners,
*        rad - defines the redius of the circle,
*
```

```
* Output: 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: Draws a beveled figure on the screen.
*           For a pure circular object x1 = x2 and y1 = y2.
*           For a rectangular object radius = 0.
*
* Note: none
*
*********************************************************************/
WORD __attribute__((weak)) Bevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
{
    SHORT       style, type, xLimit, xPos, yPos, error;
    DWORD_VAL   temp;

    #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    /* Ready */
    #else
    if(IsDeviceBusy() != 0)
        return (0);
    #endif
    temp.Val = SIN45 * rad;
    xLimit = temp.w[1] + 1;
    temp.Val = (DWORD) (ONEP25 - ((LONG) rad << 16));
    error = (SHORT) (temp.w[1]);
    yPos = rad;

    style = 0;
    type = 1;

    if(rad)
    {
        for(xPos = 0; xPos <= xLimit; xPos++)
        {
            if((++style) == _lineType)
            {
                type ^= 1;
                style = 0;
            }

            if(type)
            {
                PutPixel(x2 + xPos, y1 - yPos);         // 1st quadrant
                PutPixel(x2 + yPos, y1 - xPos);
                PutPixel(x2 + xPos, y2 + yPos);         // 2nd quadrant
                PutPixel(x2 + yPos, y2 + xPos);
                PutPixel(x1 - xPos, y2 + yPos);         // 3rd quadrant
                PutPixel(x1 - yPos, y2 + xPos);
                PutPixel(x1 - yPos, y1 - xPos);         // 4th quadrant
                PutPixel(x1 - xPos, y1 - yPos);

                if(_lineThickness)
                {
                    PutPixel(x2 + xPos, y1 - yPos - 1); // 1st quadrant
                    PutPixel(x2 + xPos, y1 - yPos + 1);
                    PutPixel(x2 + yPos + 1, y1 - xPos);
                    PutPixel(x2 + yPos - 1, y1 - xPos);
                    PutPixel(x2 + xPos, y2 + yPos - 1); // 2nd quadrant
                    PutPixel(x2 + xPos, y2 + yPos + 1);
                    PutPixel(x2 + yPos + 1, y2 + xPos);
                    PutPixel(x2 + yPos - 1, y2 + xPos);
                    PutPixel(x1 - xPos, y2 + yPos - 1); // 3rd quadrant
                    PutPixel(x1 - xPos, y2 + yPos + 1);
                    PutPixel(x1 - yPos + 1, y2 + xPos);
                    PutPixel(x1 - yPos - 1, y2 + xPos);
                    PutPixel(x1 - yPos + 1, y1 - xPos); // 4th quadrant
                    PutPixel(x1 - yPos - 1, y1 - xPos);
```

```
                    PutPixel(x1 - xPos, y1 - yPos + 1);
                    PutPixel(x1 - xPos, y1 - yPos - 1);
                }
            }

            if(error > 0)
            {
                yPos--;
                error += 5 + ((xPos - yPos) << 1);
            }
            else
                error += 3 + (xPos << 1);
        }
    }
    // Must use lines here since this can also be used to draw focus of round buttons
    if(x2 - x1)
    {
        while(!Line(x1, y1 - rad, x2, y1 - rad));

        // draw top
    }

    if(y2 - y1)
    {
        while(!Line(x1 - rad, y1, x1 - rad, y2));

        // draw left
    }

    if((x2 - x1) || (y2 - y1))
    {
        while(!Line(x2 + rad, y1, x2 + rad, y2));

        // draw right
        while(!Line(x1, y2 + rad, x2, y2 + rad));

        // draw bottom
    }

    return (1);
}

/*********************************************************************
* Function: WORD FillBevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
*
* PreCondition: None
*
* Input: x1, y1 - coordinate position of the upper left center of the
*                 circle that draws the rounded corners,
*        x2, y2 - coordinate position of the lower right center of the
*                 circle that draws the rounded corners,
*        rad - defines the redius of the circle,
*
* Output: 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: 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.
*
* Note: none
*
*********************************************************************/
WORD FillBevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
{
    #ifndef USE_NONBLOCKING_CONFIG

    SHORT       yLimit, xPos, yPos, err;
    SHORT       xCur, yCur, yNew;
```

```
    DWORD_VAL   temp;

    // note that octants here is defined as:
    // from yPos=-radius, xPos=0 in the clockwise direction octant 1 to 8 are labeled
    // assumes an origin at 0,0. Quadrants are defined in the same manner
    if(rad)
    {
        temp.Val = SIN45 * rad;
        yLimit = temp.w[1];
        temp.Val = (DWORD) (ONEP25 - ((LONG) rad << 16));
        err = (SHORT) (temp.w[1]);
        xPos = rad;
        yPos = 0;

        xCur = xPos;
        yCur = yPos;
        yNew = yPos;

        while(yPos <= yLimit)
        {

            // Drawing of the rounded panel is done only when there is a change in the
            // x direction. Bars are drawn to be efficient.
            // detect changes in the x position. Every change will mean a bar will be drawn
            // to cover the previous area. y1New records the last position of y before the
            // change in x position.
            // y1New records the last y position
            yNew = yPos;

            if(err > 0)
            {
                xPos--;
                err += 5 + ((yPos - xPos) << 1);
            }
            else
                err += 3 + (yPos << 1);
            yPos++;

            if(xCur != xPos)
            {
                if (_bevelDrawType & DRAWBOTTOMBEVEL)
                {
                    // 6th octant to 3rd octant
                    Bar(x1 - xCur, y2 + yCur, x2 + xCur, y2 + yNew);

                    // 5th octant to 4th octant
                    Bar(x1 - yNew, y2 + xPos, x2 + yNew, y2 + xCur);
                }

                if (_bevelDrawType & DRAWTOPBEVEL)
                {
                    // 8th octant to 1st octant
                    Bar(x1 - yNew, y1 - xCur, x2 + yNew, y1 - xPos);

                    // 7th octant to 2nd octant
                    Bar(x1 - xCur, y1 - yNew, x2 + xCur, y1 - yCur);
                }
                // update current values
                xCur = xPos;
                yCur = yPos;
            }
        }
    }

    // this covers both filled rounded object and filled rectangle.
    if((x2 - x1) || (y2 - y1))
    {
        if (_bevelDrawType == DRAWFULLBEVEL)
            Bar(x1 - rad, y1, x2 + rad, y2);
        else if (_bevelDrawType == DRAWTOPBEVEL)
            Bar(x1 - rad, y1, x2 + rad, y1+((y2-y1)>>1));
        else
```

```
        Bar(x1 - rad, y1+((y2-y1)>>1), x2 + rad, y2);
    }

    return (1);
    #else

    typedef enum
    {
        BEGIN,
        CHECK,
        Q8TOQ1,
        Q7TOQ2,
        Q6TOQ3,
        Q5TOQ4,
        WAITFORDONE,
        FACE
    } FILLCIRCLE_STATES;

    DWORD_VAL temp;
    static LONG err;
    static SHORT yLimit, xPos, yPos;
    static SHORT xCur, yCur, yNew;

    static FILLCIRCLE_STATES state = BEGIN;

    while(1)
    {
        if(IsDeviceBusy())
            return (0);
        switch(state)
        {
            case BEGIN:
                if(!rad)
                {   // no radius object is a filled rectangle
                    state = FACE;
                    break;
                }

                // compute variables
                temp.Val = SIN45 * rad;
                yLimit = temp.w[1];
                temp.Val = (DWORD) (ONEP25 - ((LONG) rad << 16));
                err = (SHORT) (temp.w[1]);
                xPos = rad;
                yPos = 0;
                xCur = xPos;
                yCur = yPos;
                yNew = yPos;
                state = CHECK;

            case CHECK:
                bevel_fill_check : if(yPos > yLimit)
                {
                    state = FACE;
                    break;
                }

                // y1New records the last y position
                yNew = yPos;

                // calculate the next value of x and y
                if(err > 0)
                {
                    xPos--;
                    err += 5 + ((yPos - xPos) << 1);
                }
                else
                    err += 3 + (yPos << 1);
                yPos++;
                state = Q6TOQ3;

            case Q6TOQ3:
```

```
            if(xCur != xPos)
            {

                // 6th octant to 3rd octant
                if (_bevelDrawType & DRAWBOTTOMBEVEL)
                {
                    if(Bar(x1 - xCur, y2 + yCur, x2 + xCur, y2 + yNew) == 0)
                        return (0);
                }
                state = Q5TOQ4;
                break;
            }

            state = CHECK;
            goto bevel_fill_check;

        case Q5TOQ4:

            if (_bevelDrawType & DRAWBOTTOMBEVEL)
            {
                // 5th octant to 4th octant
                if(Bar(x1 - yNew, y2 + xPos, x2 + yNew, y2 + xCur) == 0)
                    return (0);
            }
            state = Q8TOQ1;
            break;

        case Q8TOQ1:

            // 8th octant to 1st octant
            if (_bevelDrawType & DRAWTOPBEVEL)
            {
                if(Bar(x1 - yNew, y1 - xCur, x2 + yNew, y1 - xPos) == 0)
                    return (0);
            }
            state = Q7TOQ2;
            break;

        case Q7TOQ2:

            // 7th octant to 2nd octant
            if (_bevelDrawType & DRAWTOPBEVEL)
            {
                if(Bar(x1 - xCur, y1 - yNew, x2 + xCur, y1 - yCur) == 0)
                    return (0);
            }
            // update current values
            xCur = xPos;
            yCur = yPos;
            state = CHECK;
            break;



        case FACE:
            if((x2 - x1) || (y2 - y1))
            {
                if (_bevelDrawType == DRAWFULLBEVEL)
                {
                    if(Bar(x1 - rad, y1, x2 + rad, y2) == 0)
                        return (0);
                }
                else if (_bevelDrawType == DRAWTOPBEVEL)
                {
                    if(Bar(x1 - rad, y1, x2 + rad, y1+((y2-y1)>>1)) == 0)
                        return (0);
                }
                else
                {
                    if(Bar(x1 - rad, y1+((y2-y1)>>1), x2 + rad, y2) == 0)
                        return (0);
                }
```

```
                    state = WAITFORDONE;
                }
                else
                {
                    state = BEGIN;
                    return (1);
                }

            case WAITFORDONE:
                if(IsDeviceBusy())
                    return (0);
                state = BEGIN;
                return (1);
        }           // end of switch
    }               // end of while
    #endif // end of USE_NONBLOCKING_CONFIG
}

/*****************************************************************
* Function: WORD DrawPoly(SHORT numPoints, SHORT* polyPoints)
*
* PreCondition: none
*
* Input: numPoints - points number, polyPoints - pointer to points array
*
* Output: 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.
*
* Side Effects: none
*
* Overview: draws line polygon
*
* Note: none
*
*****************************************************************/
WORD __attribute__((weak)) DrawPoly(SHORT numPoints, SHORT *polyPoints)
{
    #ifndef USE_NONBLOCKING_CONFIG

    SHORT   counter;
    SHORT   sx, sy, ex, ey;

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    /* Ready */
        #else
    if(IsDeviceBusy() != 0)
        return (0);
        #endif
    sx = *polyPoints++;
    sy = *polyPoints++;
    for(counter = 0; counter < numPoints - 1; counter++)
    {
        ex = *polyPoints++;
        ey = *polyPoints++;
        while(Line(sx, sy, ex, ey) == 0);
        sx = ex;
        sy = ey;
    }

    #else

    typedef enum
    {
        POLY_BEGIN,
        POLY_DRAWING,
    } DRAWPOLY_STATES;
```

```
    static DRAWPOLY_STATES state = POLY_BEGIN;
    static SHORT counter;
    SHORT sx, sy, ex, ey;
    while(1)
    {
        if(IsDeviceBusy())
            return (0);
        switch(state)
        {
            case POLY_BEGIN:
                counter = 1;
                state = POLY_DRAWING;

            case POLY_DRAWING:
                if(counter == 0 || counter >= numPoints)
                {
                    state = POLY_BEGIN;
                    break;
                }

                while(counter < numPoints)
                {
                    sx = polyPoints[(counter - 1) * 2];
                    sy = polyPoints[(counter * 2) - 1];
                    ex = polyPoints[counter * 2];
                    ey = polyPoints[counter * 2 + 1];
                    if(Line(sx, sy, ex, ey) == 0)
                    {
                        return (0);
                    }

                    counter++;
                }

                state = POLY_BEGIN;
                return (1);
        }
    }
    #endif
    return (1);
}

/*****************************************************************
* Function: void SetFont(void* font)
*
* PreCondition: none
*
* Input: pointer to the font image
*
* Output: none
*
* Side Effects: none
*
* Overview: defines current font
*
* Note: none
*
*****************************************************************/
void __attribute__((weak)) SetFont(void *font)
{
    FONT_HEADER *pHeader;

        #ifdef USE_FONT_EXTERNAL
    FONT_HEADER header;
        #endif
    _font = font;
    switch(*((SHORT *)font))
    {
                #ifdef USE_FONT_FLASH

        case FLASH:
```

```
            pHeader = (FONT_HEADER *) ((FONT_FLASH *)font)->address;
            break;

                #endif // USE_FONT_FLASH
                #ifdef USE_FONT_EXTERNAL

        case EXTERNAL:
            pHeader = &header;
            ExternalMemoryCallback(font, 0, sizeof(FONT_HEADER), pHeader);
            break;

                #endif // USE_FONT_EXTERNAL

        default:
            return;
    }

    _fontFirstChar = pHeader->firstChar;
    _fontLastChar = pHeader->lastChar;
    _fontHeight = pHeader->height;
}

/*********************************************************************
* Function: WORD OutChar(XCHAR ch)
*
* PreCondition: none
*
* Input: character code
*
* Output: 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.
*
* Side Effects: none
*
* Overview: outputs a character
*
* Note: none
*
*********************************************************************/
WORD __attribute__((weak)) OutChar(XCHAR ch)
{
        #ifdef USE_FONT_FLASH
    GLYPH_ENTRY *pChTable = NULL;
        #endif
    BYTE        *pChImage = NULL;

        #ifdef USE_FONT_EXTERNAL
    GLYPH_ENTRY chTable;
    BYTE        chImage[EXTERNAL_FONT_BUFFER_SIZE];
    WORD        imageSize;
    DWORD_VAL   glyphOffset;
        #endif
    SHORT       chWidth = 0;
    SHORT       xCnt, yCnt, x = 0, y;
    BYTE        temp = 0, mask;

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    /* Ready */
        #else
    if(IsDeviceBusy() != 0)
        return (0);
        #endif
    if((XCHAR)ch < (XCHAR)_fontFirstChar)
        return (-1);
    if((XCHAR)ch > (XCHAR)_fontLastChar)
        return (-1);
```

```
    switch(*((SHORT *)_font))
    {
                #ifdef USE_FONT_FLASH

        case FLASH:
            pChTable = (GLYPH_ENTRY *) (((FONT_FLASH *)_font)->address +
sizeof(FONT_HEADER)) + ((XCHAR)ch - (XCHAR)_fontFirstChar);

            pChImage = (BYTE *) (((FONT_FLASH *)_font)->address +
((DWORD)(pChTable->offsetMSB) << 8) + pChTable->offsetLSB);

            chWidth = pChTable->width;

            break;
                #endif
                #ifdef USE_FONT_EXTERNAL

        case EXTERNAL:

            // get glyph entry
            ExternalMemoryCallback
            (
                _font,
                sizeof(FONT_HEADER) + ((XCHAR)ch - (XCHAR)_fontFirstChar) *
sizeof(GLYPH_ENTRY),
                sizeof(GLYPH_ENTRY),
                &chTable
            );

            chWidth = chTable.width;

            // width of glyph in bytes
            imageSize = 0;
            if(chWidth & 0x0007)
                imageSize = 1;
            imageSize += (chWidth >> 3);

            // glyph image size
            imageSize *= _fontHeight;

            // get glyph image
            glyphOffset.w[1] = (chTable.offsetMSB >> 8);
            glyphOffset.w[0] = (chTable.offsetMSB << 8) + (chTable.offsetLSB);

            ExternalMemoryCallback(_font, glyphOffset.Val, imageSize, &chImage);
            pChImage = (BYTE *) &chImage;

            break;
                #endif

        default:
            break;
    }

    if(_fontOrientation == ORIENT_HOR)
    {
        y = GetY();
        for(yCnt = 0; yCnt < _fontHeight; yCnt++)
        {
            x = GetX();
            mask = 0;
            for(xCnt = 0; xCnt < chWidth; xCnt++)
            {
                if(mask == 0)
                {
                    temp = *pChImage++;
                    mask = 0x01;
                }

                if(temp & mask)
                {
                    PutPixel(x, y);
```

```
                }

                x++;
                mask <<= 1;
            }

            y++;
        }

        // move cursor
        _cursorX = x;
    }
    else
    {
        y = GetX();
        for(yCnt = 0; yCnt < _fontHeight; yCnt++)
        {
            x = GetY();
            mask = 0;
            for(xCnt = 0; xCnt < chWidth; xCnt++)
            {
                if(mask == 0)
                {
                    temp = *pChImage++;
                    mask = 0x01;
                }

                if(temp & mask)
                {
                    PutPixel(y, x);
                }

                x--;
                mask <<= 1;
            }

            y++;
        }

        // move cursor
        _cursorY = x;
    }

    return (1);
}

/*****************************************************************************
* Function: WORD OutText(XCHAR* textString)
*
* PreCondition: none
*
* Input: textString - pointer to text string
*
* Output: non-zero if drawing done (used for NON-BLOCKING configuration)
*
* Side Effects: none
*
* Overview: outputs text from current position
*
* Note: none
*
*****************************************************************************/
WORD __attribute__((weak)) OutText(XCHAR *textString)
{
    #ifndef USE_NONBLOCKING_CONFIG

    XCHAR   ch;
    while((XCHAR)15 < (XCHAR)(ch = *textString++))
        while(OutChar(ch) == 0);
    return (1);

    #else
```

```
    XCHAR       ch;
    static WORD counter = 0;

    while((XCHAR)(ch = *(textString + counter)) > (XCHAR)15)
    {
        if(OutChar(ch) == 0)
            return (0);
        counter++;
    }

    counter = 0;
    return (1);
    #endif
}

/*****************************************************************
* Function: WORD OutTextXY(SHORT x, SHORT y, XCHAR* textString)
*
* PreCondition: none
*
* Input: x,y - starting coordinates, textString - pointer to text string
*
* Output: non-zero if drawing done (used for NON-BLOCKING configuration)
*
* Side Effects: none
*
* Overview: outputs text from x,y position
*
* Note: none
*
*****************************************************************/
WORD __attribute__((weak)) OutTextXY(SHORT x, SHORT y, XCHAR *textString)
{
    #ifndef USE_NONBLOCKING_CONFIG
    MoveTo(x, y);
    OutText(textString);
    return (1);

    #else

    static BYTE start = 1;

    if(start)
    {
        MoveTo(x, y);
        start = 0;
    }

    if(OutText(textString) == 0)
    {
        return (0);
    }
    else
    {
        start = 1;
        return (1);
    }

    #endif
}

/*****************************************************************
* Function: SHORT GetTextWidth(XCHAR* textString, void* font)
*
* PreCondition: none
*
* Input: textString - pointer to the text string,
*        font - pointer to the font
*
* Output: text width in pixels
*
```

```
* Side Effects: none
*
* Overview: returns text width for the font
*
* Note: none
*
*********************************************************************/
SHORT __attribute__((weak)) GetTextWidth(XCHAR *textString, void *font)
{
        #if defined (USE_FONT_RAM) || defined (USE_FONT_FLASH)
    GLYPH_ENTRY *pChTable;
    FONT_HEADER *pHeader;
        #endif
        #ifdef USE_FONT_EXTERNAL
    GLYPH_ENTRY chTable;
    FONT_HEADER header;
        #endif

        #if defined (USE_FONT_RAM) || defined (USE_FONT_FLASH) || defined
(USE_FONT_EXTERNAL)
    SHORT       textWidth;
    XCHAR       ch;
    XCHAR       fontFirstChar;
    XCHAR       fontLastChar;
        #endif

    switch(*((SHORT *)font))
    {
                #ifdef USE_FONT_RAM

        case RAM:
            pHeader = (FONT_HEADER *) ((FONT_RAM *)font)->address;
            fontFirstChar = pHeader->firstChar;
            fontLastChar = pHeader->lastChar;
            pChTable = (GLYPH_ENTRY *) (pHeader + 1);
            textWidth = 0;
            while((unsigned XCHAR)15 < (unsigned XCHAR)(ch = *textString++))
            {
                if((unsigned XCHAR)ch < (unsigned XCHAR)fontFirstChar)
                    continue;
                if((unsigned XCHAR)ch > (unsigned XCHAR)fontLastChar)
                    continue;
                textWidth += (pChTable + ((unsigned XCHAR)ch - (unsigned
XCHAR)fontFirstChar))->width;
            }

            return (textWidth);
                #endif

                #ifdef USE_FONT_FLASH

        case FLASH:
            pHeader = (FONT_HEADER *) ((FONT_FLASH *)font)->address;
            fontFirstChar = pHeader->firstChar;
            fontLastChar = pHeader->lastChar;
            pChTable = (GLYPH_ENTRY *) (pHeader + 1);
            textWidth = 0;
            while((XCHAR)15 < (XCHAR)(ch = *textString++))
            {
                if((XCHAR)ch < (XCHAR)fontFirstChar)
                    continue;
                if((XCHAR)ch > (XCHAR)fontLastChar)
                    continue;
                textWidth += (pChTable + ((XCHAR)ch - (XCHAR)fontFirstChar))->width;
            }

            return (textWidth);
                #endif
                #ifdef USE_FONT_EXTERNAL

        case EXTERNAL:
            ExternalMemoryCallback(font, 0, sizeof(FONT_HEADER), &header);
```

```
            fontFirstChar = header.firstChar;
            fontLastChar = header.lastChar;
            textWidth = 0;
            while((XCHAR)15 < (XCHAR)(ch = *textString++))
            {
                if((XCHAR)ch < (XCHAR)fontFirstChar)
                    continue;
                if((XCHAR)ch > (XCHAR)fontLastChar)
                    continue;
                ExternalMemoryCallback
                (
                    font,
                    sizeof(FONT_HEADER) + sizeof(GLYPH_ENTRY) * ((XCHAR)ch -
(XCHAR)fontFirstChar),
                    sizeof(GLYPH_ENTRY),
                    &chTable
                );
                textWidth += chTable.width;
            }

            return (textWidth);
                #endif

        default:
            return (0);
    }
}

/*****************************************************************
* Function: SHORT GetTextHeight(void* font)
*
* PreCondition: none
*
* Input: pointer to the font
*
* Output: character height in pixels
*
* Side Effects: none
*
* Overview: returns characters height for the font
*
* Note: none
*
*****************************************************************/
SHORT __attribute__((weak)) GetTextHeight(void *font)
{
        #ifdef USE_FONT_EXTERNAL

    char    height;
        #endif
    switch(*((SHORT *)font))
    {
                #ifdef USE_FONT_RAM
        case RAM:
            return ((FONT_HEADER *) ((FONT_RAM *)font)->address)->height;
                #endif

                #ifdef USE_FONT_FLASH
        case FLASH:
            return ((FONT_HEADER *) ((FONT_FLASH *)font)->address)->height;
                #endif

                #ifdef USE_FONT_EXTERNAL
        case EXTERNAL:
            ExternalMemoryCallback(font, sizeof(FONT_HEADER) - 2, 1, &height);
            return (height);
                #endif

        default:
            return (0);
    }
}
```

```
/*****************************************************************
* Function: SHORT GetImageWidth(void* image)
*
* PreCondition: none
*
* Input: image - image pointer
*
* Output: none
*
* Side Effects: none
*
* Overview: returns image width
*
* Note: none
*
*****************************************************************/
SHORT __attribute__((weak)) GetImageWidth(void *image)
{
    #ifdef USE_BITMAP_EXTERNAL

    SHORT   width;
    #endif
    switch(*((SHORT *)image))
    {
            #ifdef USE_BITMAP_FLASH

        case FLASH:
            return (*((FLASH_WORD *) ((IMAGE_FLASH *)image)->address + 2));
            #endif
            #ifdef USE_BITMAP_EXTERNAL

        case EXTERNAL:
            ExternalMemoryCallback(image, 4, 2, &width);
            return (width);
            #endif

        default:
            return (0);
    }
}

/*****************************************************************
* Function: SHORT GetImageHeight(void* image)
*
* PreCondition: none
*
* Input: image - image pointer
*
* Output: none
*
* Side Effects: none
*
* Overview: returns image height
*
* Note: none
*
*****************************************************************/
SHORT __attribute__((weak)) GetImageHeight(void *image)
{
    #ifdef USE_BITMAP_EXTERNAL

    SHORT   height;
    #endif
    switch(*((SHORT *)image))
    {
            #ifdef USE_BITMAP_FLASH

        case FLASH:
            return (*((FLASH_WORD *) ((IMAGE_FLASH *)image)->address + 1));
            #endif
            #ifdef USE_BITMAP_EXTERNAL
```

```
        case EXTERNAL:
            ExternalMemoryCallback(image, 2, 2, &height);
            return (height);
            #endif


        default:
            return (0);
    }
}

    #ifdef USE_BITMAP_FLASH

/*********************************************************************
* Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner,
*        image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void __attribute__((weak)) PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE
stretch)
{
    register FLASH_BYTE *flashAddress;
    BYTE                temp = 0;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                xc, yc;
    WORD                pallete[2];
    BYTE                mask;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[0] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[1] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
```

```
                    temp = *flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                // Set color
                    #ifdef USE_PALETTE
                        SetColor(1);
                    #else
                        SetColor(pallete[1]);
                    #endif
                }
                else
                {
                    // Set color
                    #ifdef USE_PALETTE
                        SetColor(0);
                    #else
                        SetColor(pallete[0]);
                    #endif
                }

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    // adjust to next location
                    xc++;
                    // Shift to the next pixel
                    mask >>= 1;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            mask = 0;

            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress++;
                    mask = 0x80;
                }

                    // Set color
                if(mask & temp)
                {
                    // Set color
                    #ifdef USE_PALETTE
                        SetColor(1);
                    #else
                        SetColor(pallete[1]);
                    #endif
                }
                else
                {
                    // Set color
                    #ifdef USE_PALETTE
                        SetColor(0);
                    #else
                        SetColor(pallete[0]);
```

```
                    #endif
                }
                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
                // Shift to the next pixel
                mask >>= 1;
            }
            yc+=2;
        }
    }

}

/*****************************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 16 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
        #if (COLOR_DEPTH >= 4)

/* */
void __attribute__((weak)) PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE
stretch)
{
    register FLASH_BYTE *flashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                xc, yc;
    BYTE                temp = 0;
    WORD                pallete[16];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    yc = top;
```

```
    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                // second pixel in byte
                // Set color
                #ifdef USE_PALETTE
                    SetColor(temp >> 4);
                #else
                    SetColor(pallete[temp >> 4]);
                #endif
                }
                else
                {
                    temp = *flashAddress++;

                // first pixel in byte
                // Set color
                #ifdef USE_PALETTE
                    SetColor(temp & 0x0f);
                #else
                    SetColor(pallete[temp & 0x0f]);
                #endif
                }

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            for(x = 0; x < sizeX; x++)
            {

               if(x & 0x0001)
               {

                // second pixel in byte
                // Set color
                #ifdef USE_PALETTE
                    SetColor(temp >> 4);
                #else
                    SetColor(pallete[temp >> 4]);
                #endif
                }
                else
                {
                    temp = *flashAddress++;

                // first pixel in byte
                // Set color
```

```
                #ifdef USE_PALETTE
                    SetColor(temp & 0x0f);
                #else
                    SetColor(pallete[temp & 0x0f]);
                #endif
                }

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
            }
            yc+=2;
        }
    }
}
        #endif // #if (COLOR_DEPTH >= 4)

/*********************************************************************
* Function: void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 256 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
        #if (COLOR_DEPTH >= 8)

/* */
void __attribute__((weak)) PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE
stretch)
{
    register FLASH_BYTE *flashAddress;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                xc, yc;
    BYTE                temp;
    WORD                pallete[256];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
```

```
    }

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *flashAddress++;

                // Set color
                #ifdef USE_PALETTE
                    SetColor(temp);
                #else
                    SetColor(pallete[temp]);
                #endif

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *flashAddress++;

                // Set color
                #ifdef USE_PALETTE
                    SetColor(temp);
                #else
                    SetColor(pallete[temp]);
                #endif

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
            }
            yc+=2;
        }
    }

}
        #endif // #if (COLOR_DEPTH >= 8)

/*****************************************************************************
* Function: void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
```

```
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs hicolor image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
        #if (COLOR_DEPTH == 16)

/* */
void __attribute__((weak)) PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE
stretch)
{
    register FLASH_WORD *flashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                xc, yc;
    WORD                temp;

    // Move pointer to size information
    flashAddress = (FLASH_WORD *)image + 1;

    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *flashAddress++;
                SetColor(temp);

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *flashAddress++;
                SetColor(temp);

                #ifdef USE_TRANSPARENT_COLOR
```

```
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
            }
            yc+=2;
        }
    }

}

        #endif //#if (COLOR_DEPTH == 16)
    #endif // #ifdef USE_BITMAP_FLASH

    #ifdef USE_BITMAP_EXTERNAL

/*****************************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************************/
void __attribute__((weak)) PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[2];
    BYTE            lineBuffer[((GetMaxX() + 1) / 8) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp = 0;
    BYTE            mask;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            xc, yc;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
```

```
    sizeY = bmp.height;

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;

            pData = lineBuffer;
            mask = 0;

            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                    // Read 8 pixels from flash
                    if(mask == 0)
                    {
                        temp = *pData++;
                        mask = 0x80;
                    }

                    // Set color
                    if(mask & temp)
                    {
                        // Set color
                        #ifdef USE_PALETTE
                            SetColor(1);
                        #else
                            SetColor(pallete[1]);
                        #endif
                    }
                    else
                    {
                        // Set color
                        #ifdef USE_PALETTE
                            SetColor(0);
                        #else
                            SetColor(pallete[0]);
                        #endif
                    }


                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
                // Shift to the next pixel
                mask >>= 1;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;

            pData = lineBuffer;
            mask = 0;
```

```
            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                      // Read 8 pixels from flash
                    if(mask == 0)
                    {
                        temp = *pData++;
                        mask = 0x80;
                    }

                    // Set color
                    if(mask & temp)
                    {
                        // Set color
                        #ifdef USE_PALETTE
                            SetColor(1);
                        #else
                            SetColor(pallete[1]);
                        #endif
                    }
                    else
                    {
                        // Set color
                        #ifdef USE_PALETTE
                            SetColor(0);
                        #else
                            SetColor(pallete[0]);
                        #endif
                    }


                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
                // Shift to the next pixel
                mask >>= 1;

            }
            yc+=2;
        }
    }

}

/*****************************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
```

```
*******************************************************************/
        #if (COLOR_DEPTH >= 4)

/* */
void __attribute__((weak)) PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[16];
    BYTE            lineBuffer[((GetMaxX() + 1) / 2) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp = 0;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            xc, yc;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;

            pData = lineBuffer;

            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    // Set color
                    #ifdef USE_PALETTE
                        SetColor(temp >> 4);
                    #else
                        SetColor(pallete[temp >> 4]);
                    #endif
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    // Set color
```

```
                    #ifdef USE_PALETTE
                        SetColor(temp & 0x0f);
                    #else
                        SetColor(pallete[temp & 0x0f]);
                    #endif
                }

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;

            pData = lineBuffer;

            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    // Set color
                    #ifdef USE_PALETTE
                        SetColor(temp >> 4);
                    #else
                        SetColor(pallete[temp >> 4]);
                    #endif
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    // Set color
                    #ifdef USE_PALETTE
                        SetColor(temp & 0x0f);
                    #else
                        SetColor(pallete[temp & 0x0f]);
                    #endif
                }

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
            }
            yc+=2;
```

```
        }
    }
}

        #endif

/*****************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************/
        #if (COLOR_DEPTH >= 8)

/* */
void __attribute__((weak)) PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[256];
    BYTE            lineBuffer[(GetMaxX() + 1)];
    BYTE            *pData;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            xc, yc;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
            memOffset += sizeX;

            pData = lineBuffer;

            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;

                // Set color
```

```
                #ifdef USE_PALETTE
                    SetColor(temp);
                #else
                    SetColor(pallete[temp]);
                #endif

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
            memOffset += sizeX;

            pData = lineBuffer;

            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;

                // Set color
                #ifdef USE_PALETTE
                    SetColor(temp);
                #else
                    SetColor(pallete[temp]);
                #endif

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
            }
            yc+=2;
        }
    }

}

        #endif

/*****************************************************************************
* Function: void PutImage16BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
```

```
 *
 * Overview: outputs monochrome image starting from left,top coordinates
 *
 * Note: image must be located in external memory
 *
 ********************************************************************/
        #if (COLOR_DEPTH == 16)

/* */
void __attribute__((weak)) PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE
stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            lineBuffer[(GetMaxX() + 1)];
    WORD            *pData;
    WORD            byteWidth;

    WORD            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            xc, yc;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    byteWidth = sizeX << 1;

    yc = top;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance
    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;

            pData = lineBuffer;

            xc = left;
            for(x = 0; x < sizeX; x++)
                {
                temp = *pData++;
                SetColor(temp);

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                        // Write pixel(s) to screen
                        PutPixel(xc, yc);
                    xc++;
            }
            yc++;
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
```

```
            memOffset += byteWidth;

            pData = lineBuffer;

            xc = left;
            for(x = 0; x < sizeX; x++)
                {
                temp = *pData++;
                SetColor(temp);

                #ifdef USE_TRANSPARENT_COLOR
                    if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                #endif
                    {
                        // Write pixel(s) to screen, basically writes a tile of 4x4 pixels
to the screen
                        PutPixel(xc,   yc);
                        PutPixel(xc,   yc+1);
                        PutPixel(xc+1, yc);
                        PutPixel(xc+1, yc+1);
                    }
                    xc += 2;
            }
            yc+=2;
        }
    }
}

        #endif
    #endif

/*****************************************************************************
* Function: WORD PutImage(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner,
*        image - image pointer,
*        stretch - image stretch factor
*
* Output: 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.
*
* Side Effects: none
*
* Overview: outputs image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
WORD __attribute__((weak)) PutImage(SHORT left, SHORT top, void *image, BYTE stretch)
{
    FLASH_BYTE  *flashAddress;
    BYTE        colorDepth;
    WORD        colorTemp;
    WORD        resType;

#if defined(USE_COMP_RLE)
    GFX_IMAGE_HEADER *pimghdr = (GFX_IMAGE_HEADER *)image;
#endif

#ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0) Nop();

    /* Ready */
#else
    if(IsDeviceBusy() != 0)
        return (0);
```

```
#endif

    // Save current color
    colorTemp = GetColor();
    resType = *((WORD *)image);


    switch(resType & (GFX_MEM_MASK | GFX_COMP_MASK))
    {

                #ifdef USE_COMP_RLE
                #ifdef USE_BITMAP_FLASH
        case (FLASH | COMP_RLE):

            // Image address
            flashAddress = pimghdr->LOCATION.progByteAddress;
            // Read color depth
            colorDepth = pimghdr->colorDepth;

            // Draw picture
            switch(colorDepth)
            {
                    #if (COLOR_DEPTH >= 4)
                case 4:
                    PutImageRLE4BPP(left, top, flashAddress, stretch); break;
                    #endif

                    #if (COLOR_DEPTH >= 8)
                case 8:
                    PutImageRLE8BPP(left, top, flashAddress, stretch); break;
                    #endif

                    default:    break;
            }
            break;
                #endif //#ifdef USE_BITMAP_FLASH

                #ifdef USE_BITMAP_EXTERNAL
        case (EXTERNAL | COMP_RLE):

            // Get color depth
            ExternalMemoryCallback(image, 1, 1, &colorDepth);

            // Draw picture
            switch(colorDepth)
            {
                    #if (COLOR_DEPTH >= 4)
                case 4:
                    PutImageRLE4BPPExt(left, top, image, stretch); break;
                    #endif

                    #if (COLOR_DEPTH >= 8)
                case 8:
                    PutImageRLE8BPPExt(left, top, image, stretch); break;
                    #endif

                default:    break;
            }
            break;
                #endif //#ifdef USE_BITMAP_EXTERNAL
                #endif // #ifdef USE_COMP_RLE

                #ifdef USE_BITMAP_FLASH

        case (FLASH | COMP_NONE):

            // Image address
            flashAddress = ((IMAGE_FLASH *)image)->address;

            // Read color depth
            colorDepth = *(flashAddress + 1);
```

```
            // Draw picture
            switch(colorDepth)
            {
                case 1:
                    PutImage1BPP(left, top, flashAddress, stretch); break;

                    #if (COLOR_DEPTH >= 4)
                case 4:
                    PutImage4BPP(left, top, flashAddress, stretch); break;
                    #endif

                    #if (COLOR_DEPTH >= 8)
                case 8:
                    PutImage8BPP(left, top, flashAddress, stretch); break;
                    #endif

                    #if (COLOR_DEPTH == 16)
                case 16:
                    PutImage16BPP(left, top, flashAddress, stretch); break;
                    #endif

                default:    break;

            }

            break;
                #endif // #ifdef USE_BITMAP_FLASH

                #ifdef USE_BITMAP_EXTERNAL

        case (EXTERNAL | COMP_NONE):

            // Get color depth
            ExternalMemoryCallback(image, 1, 1, &colorDepth);

            // Draw picture
            switch(colorDepth)
            {
                case 1:
                    PutImage1BPPExt(left, top, image, stretch); break;

                    #if (COLOR_DEPTH >= 4)
                case 4:
                    PutImage4BPPExt(left, top, image, stretch); break;
                    #endif

                    #if (COLOR_DEPTH >= 8)
                case 8:
                    PutImage8BPPExt(left, top, image, stretch); break;
                    #endif

                    #if (COLOR_DEPTH == 16)
                case 16:
                    PutImage16BPPExt(left, top, image, stretch); break;
                    #endif

                default:    break;
            }

            break;
                #endif //#ifdef USE_BITMAP_EXTERNAL

                #if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210)
                #ifdef USE_COMP_IPU
        case (FLASH | COMP_IPU):
        case (EXTERNAL | COMP_IPU):
        case (EDS_EPMP | COMP_IPU):
                #endif // #ifdef USE_COMP_IPU
        case (EDS_EPMP | COMP_NONE):

            // this requires special processing of images in Extended Data Space
            // call the driver specific function to perform the processing
```

```
            PutImageDrv(left, top, image, stretch);
            break;

                #endif //#if defined (__PIC24FJ256DA210__)

        default:
            break;
    }

    // Restore current color
    SetColor(colorTemp);
    return (1);
}

/*****************************************************************
* Function:  SHORT GetSineCosine(SHORT v, WORD type)
*
* PreCondition: none
*
* Input: v - the angle used to calculate the sine or cosine value.
*            The angle must be in the range of -360 to 360 degrees.
*        type - sets if the angle calculation is for a sine or cosine
*               - GETSINE (0) - get the value of sine(v).
*               - GETCOSINE (1) - return the value of cosine(v).
*
* Output: Returns the sine or cosine of the angle given.
*
* Side Effects: none
*
* Overview: This calculates the sine or cosine value of the given angle.
*
* Note: none
*
*****************************************************************/
SHORT GetSineCosine(SHORT v, WORD type)
{
    // if angle is neg, convert to pos equivalent
    if (v < 0)
        v += 360;
    //v /= DEGREECOUNT;                 // convert into ticks from degrees, tick = 3 deg

    if(v >= COSINETABLEENTRIES * 3)
    {
        v -= COSINETABLEENTRIES * 3;
        return ((type == GETSINE) ? -(_CosineTable[v]) : (_CosineTable[COSINETABLEENTRIES -
v]));
    }
    else if(v >= COSINETABLEENTRIES * 2)
    {
        v -= COSINETABLEENTRIES * 2;
        return ((type == GETSINE) ? -(_CosineTable[(COSINETABLEENTRIES - v)]) :
-(_CosineTable[v]));
    }
    else if(v >= COSINETABLEENTRIES)
    {
        v -= COSINETABLEENTRIES;
        return ((type == GETSINE) ? (_CosineTable[v]) : -(_CosineTable[COSINETABLEENTRIES -
v]));
    }
    else
    {
        return ((type == GETSINE) ? (_CosineTable[COSINETABLEENTRIES - v]) :
(_CosineTable[v]));
    }


}

/*****************************************************************
* Function:  WORD DrawArc(SHORT cx, SHORT cy, SHORT r1, SHORT r2, SHORT startAngle, SHORT
endAngle)
*
```

```
* PreCondition: none
*
* Input: cx - the location of the center of the arc in the x direction.
*        cy - the location of the center of the arc in the y direction.
*        r1 - the smaller radius of the arc.
*        r2 - the larger radius of the arc.
*        startAngle - start angle of the arc.
*        endAngle - end angle of the arc.
*
* Output: Returns 1 if the rendering is done, 0 if not yet done.
*
* Side Effects: 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 successfuly
*           and returns a 0 when it is not yet finished. The next call to the
*           function will continue the rendering.
*
* Note: none
*
*********************************************************************/
WORD __attribute__((weak)) DrawArc(SHORT cx, SHORT cy, SHORT r1, SHORT r2, SHORT
startAngle, SHORT endAngle)
{

    SHORT i;
    SHORT x1,y1,x2,y2;

    for (i=startAngle; i <= endAngle; i++)
    {
        // get inner arc x,y position
        y1 = (r1*Sine(i))/256;
        x1 = (r1*Cosine(i))/256;

        // get outer arc x,y position
        if (r1 != r2)
        {
            y2 = (r2*Sine(i))/256;
            x2 = (r2*Cosine(i))/256;

            // check if we need to double the line to cover all pixels
            if ((x1 == x2) || (y1 == y2))
            {
                SetLineThickness(NORMAL_LINE);
            }
            else
            {
                SetLineThickness(THICK_LINE);
            }
            // draw the lines to fill the arc
            while(!Line(cx+x1, cy+y1, cx+x2, cy+y2));
        }
        else
        {
            PutPixel(cx+x1, cy+y1);
        }

    }
    return 1;
}


#ifdef USE_GRADIENT
#define WAIT_UNTIL_FINISH(x)    while(!x)

#if (COLOR_DEPTH == 24)
#define GetRed(color)       (((color) & 0xFF0000) >> 16)
#define GetGreen(color)     (((color) & 0x00FF00) >> 8)
#define GetBlue(color)      ((color) & 0x0000FF)
#else
#define GetRed(color)       (((color) & 0xF800) >> 8)
#define GetGreen(color)     (((color) & 0x07E0) >> 3)
```

```
#define GetBlue(color)      (((color) & 0x001F) << 3)
#endif

void BarGradient(short left, short top, short right, short bottom, GFX_COLOR color1,
GFX_COLOR color2, DWORD length, BYTE direction)
{

    WORD startRed, startBlue, startGreen;
    WORD endRed, endBlue, endGreen;

    long rdiff=0,gdiff=0,bdiff=0;
    short i,steps;

    SetColor(color2);

    // if length is 100, why waste the bar call?
    switch(direction)
    {
    case GRAD_UP:
        length = length * (bottom - top);
        length /= 100;
        steps = length;
        WAIT_UNTIL_FINISH(Bar(left,top,right,bottom-steps));
        break;

    case GRAD_DOWN:
        length = length * (bottom - top);
        length /= 100;
        steps = length;
        WAIT_UNTIL_FINISH(Bar(left,top+steps,right,bottom));
        break;

    case GRAD_RIGHT:
        length = length * (right - left);
        length /= 100;
        steps = length;
        WAIT_UNTIL_FINISH(Bar(left+steps,top,right,bottom));
        break;

    case GRAD_LEFT:
        length = length * (right - left);
        length /= 100;
        steps = length;
        WAIT_UNTIL_FINISH(Bar(left,top,right-steps,bottom));
        break;

    case GRAD_DOUBLE_VER:
        steps = (right - left) >> 1;
        break;

    case GRAD_DOUBLE_HOR:
        steps = (bottom - top) >> 1;
        break;

    default:
        return;
    }

    startRed    = GetRed(color1);
    startGreen  = GetGreen(color1);
    startBlue   = GetBlue(color1);

    endRed      = GetRed(color2);
    endGreen    = GetGreen(color2);
    endBlue     = GetBlue(color2);

    ////////////////////////////////////

    //Find the step size for the red portion//
    rdiff = ((long)endRed - (long)startRed) << 8;
    rdiff /= steps;
```

```
    //Find the step size for the green portion//
    gdiff = ((long)endGreen - (long)startGreen) << 8;
    gdiff /= steps;

    //Find the step size for the blue portion//
    bdiff = ((long)endBlue - (long)startBlue) << 8;
    bdiff /= steps;

    short barSize = 1;
    color1 = RGBConvert(startRed, startGreen, startBlue);

    // PERCENTAGE BASED CODE
    for(i=0; i < steps; i++)
    {
        //Calculate the starting RGB values
        endRed      = startRed + ((rdiff*i) >> 8);
        endGreen    = startGreen + ((gdiff*i) >> 8);
        endBlue     = startBlue + ((bdiff*i) >> 8);

        color2 = RGBConvert(endRed, endGreen, endBlue);

        if(color2 == color1)
        {
            barSize++;
            continue;
        }

        SetColor(color2);
        color1 = color2;

        switch(direction)          //This switch statement draws the gradient depending on
direction chosen
        {
        case GRAD_DOWN:
            WAIT_UNTIL_FINISH(Bar(left, top, right, top + barSize));
            top += barSize;
        break;

        case GRAD_UP:
            WAIT_UNTIL_FINISH(Bar(left,bottom - barSize,right,bottom));
            bottom -= barSize;
        break;

        case GRAD_RIGHT:
            WAIT_UNTIL_FINISH(Bar(left, top, left + barSize, bottom));
            left += barSize;
        break;

        case GRAD_LEFT:
            WAIT_UNTIL_FINISH(Bar(right - barSize, top, right, bottom));
            right -= barSize;
        break;

        case GRAD_DOUBLE_VER:
            WAIT_UNTIL_FINISH(Bar(right - barSize, top, right, bottom));
            right -= barSize;
            WAIT_UNTIL_FINISH(Bar(left, top, left + barSize, bottom));
            left += barSize;
        break;

        case GRAD_DOUBLE_HOR:
            WAIT_UNTIL_FINISH(Bar(left, bottom - barSize, right, bottom));
            bottom -= barSize;
            WAIT_UNTIL_FINISH(Bar(left, top, right, top + barSize));
            top += barSize;
        break;

        default:
            break;
        }

        barSize = 1;
```

```
    }

    if(barSize > 1)
    {

        SetColor(RGBConvert(endRed, endGreen, endBlue));

        switch(direction)          //This switch statement draws the gradient depending on
direction chosen
        {
        case GRAD_DOWN:
            WAIT_UNTIL_FINISH(Bar(left, top, right, top + barSize));
        break;

        case GRAD_UP:
            WAIT_UNTIL_FINISH(Bar(left,bottom - barSize,right,bottom));
        break;

        case GRAD_RIGHT:
            WAIT_UNTIL_FINISH(Bar(left, top, left + barSize, bottom));
        break;

        case GRAD_LEFT:
            WAIT_UNTIL_FINISH(Bar(right - barSize, top, right, bottom));
        break;

        case GRAD_DOUBLE_VER:
            WAIT_UNTIL_FINISH(Bar(right - barSize, top, right, bottom));
            WAIT_UNTIL_FINISH(Bar(left, top, left + barSize, bottom));
        break;

        case GRAD_DOUBLE_HOR:
            WAIT_UNTIL_FINISH(Bar(left, bottom - barSize, right, bottom));
            WAIT_UNTIL_FINISH(Bar(left, top, right, top + barSize));
        break;

        default:
            break;
        }

    }

}

WORD BevelGradient(short x1, short y1, short x2, short y2,SHORT rad, GFX_COLOR color1,
GFX_COLOR color2, DWORD length, BYTE direction)
{
    WORD i;
    WORD sred,sblue,sgreen;
    WORD ered,eblue,egreen;
    GFX_COLOR EndColor;
    long rdiff=0,gdiff=0,bdiff=0;
    short steps;
    EndColor = color2;

    switch(direction)     //This switch statement calculates the amount of transitions
needed
    {
    case GRAD_UP:
    case GRAD_DOWN:
        length = length * (y2 - y1 +(rad << 1));
        length /= 100;
        steps = length;
        break;

    case GRAD_RIGHT:
    case GRAD_LEFT:
        length = length * (x2 - x1 +(rad << 1));
        length /= 100;
        steps = length;
        break;
```

```
    case GRAD_DOUBLE_VER:
        steps = (x2 - x1 +(rad << 1)) >> 1;
        break;

    case GRAD_DOUBLE_HOR:
        steps = (y2 - y1 +(rad << 1)) >> 1;
        break;

    default:
        return (1);
    }

    //Calculate the starting RGB values
    sred    = GetRed(color1);
    sgreen  = GetGreen(color1);
    sblue   = GetBlue(color1);

    ered    = GetRed(color2);
    egreen  = GetGreen(color2);
    eblue   = GetBlue(color2);

    ///////////////////////////////////

    //Find the step size for the red portion//
    rdiff = ((long)ered - (long)sred) << 8;
    rdiff /= steps;

    //Find the step size for the green portion//
    gdiff = ((long)egreen - (long)sgreen) << 8;
    gdiff /= steps;

    //Find the step size for the blue portion//
    bdiff = ((long)eblue - (long)sblue) << 8;
    bdiff /= steps;

    typedef enum
    {
        BEGIN,
        CHECK,
        Q8TOQ1,
        Q7TOQ2,
        Q6TOQ3,
        Q5TOQ4,
        WAITFORDONE,
        FACE
    } FILLCIRCLE_STATES;

    DWORD_VAL temp;
    static LONG err;
    static SHORT yLimit, xPos, yPos;
    static SHORT xCur, yCur, yNew;

    static FILLCIRCLE_STATES state = BEGIN;

    while(1)
    {
        if(IsDeviceBusy())
            return (0);
        switch(state)
        {
            case BEGIN:
                if(!rad)
                {   // no radius object is a filled rectangle
                    state = FACE;
                    break;
                }

                // compute variables
                temp.Val = SIN45 * rad;
                yLimit = temp.w[1];
                temp.Val = (DWORD) (ONEP25 - ((LONG) rad << 16));
                err = (SHORT) (temp.w[1]);
```

```
                xPos = rad;
                yPos = 0;
                xCur = xPos;
                yCur = yPos;
                yNew = yPos;
                state = CHECK;

            case CHECK:
                bevel_fill_check : if(yPos > yLimit)
                {
                    state = FACE;
                    break;
                }

                // y1New records the last y position
                yNew = yPos;

                // calculate the next value of x and y
                if(err > 0)
                {
                    xPos--;
                    err += 5 + ((yPos - xPos) << 1);
                }
                else
                    err += 3 + (yPos << 1);
                yPos++;
                state = Q6TOQ3;

            case Q6TOQ3:
                if(xCur != xPos)
                {

                    // 6th octant to 3rd octant
                    //if (_bevelDrawType & DRAWBOTTOMBEVEL)
                    {
                        if(direction == GRAD_DOUBLE_VER || direction == GRAD_DOUBLE_HOR)
                            i = (y1 - yCur) - y1 + rad;
                        else
                            i = (y2 + yCur) - y1 + rad;

                        ered = sred + ((rdiff*i) >> 8);
                        egreen = sgreen + ((gdiff*i) >> 8);
                        eblue = sblue + ((bdiff*i) >> 8);

                        color2 = RGBConvert(ered,egreen,eblue);

                        SetColor(color2);

                        switch(direction)    //Direction matter because different portions
of the circle are drawn
                        {
                           case GRAD_LEFT:
                               if(i>length) SetColor(EndColor);
                               if(Bar(x1 - yNew, y1 - xCur, x1 - yCur, y2 + xCur) == 0)
return (0);
                                break;

                           case GRAD_RIGHT:
                           case GRAD_DOUBLE_VER:
                                if(i>length)
                                    SetColor(EndColor);

                                if(Bar(x2 + yCur, y1 - xCur, x2 + yNew, y2 + xCur) == 0)
                                    return (0);
                                break;

                           case GRAD_UP:
                                if(i>length)
                                    SetColor(EndColor);

                                if(Bar(x1 - xCur, y1 - yNew, x2 + xCur, y1 - yCur) == 0)
                                    return (0);
```

```
                                break;

                            case GRAD_DOWN:
                            case GRAD_DOUBLE_HOR:
                                 if(i>length)
                                     SetColor(EndColor);
                                 if(Bar(x1 - xCur, y2 + yCur, x2 + xCur, y2 + yNew) == 0)
                                     return (0);

                            default:
                                break;
                        }

                    }
                    state = Q5TOQ4;
                    break;
                }

                state = CHECK;
                goto bevel_fill_check;

            case Q5TOQ4:

                //if (_bevelDrawType & DRAWBOTTOMBEVEL)
                {
                    if(direction == GRAD_DOUBLE_VER || direction == GRAD_DOUBLE_HOR)
                    i = y1 + xPos - y1 + rad;
                    else

                    // 5th octant to 4th octant
                    i = (y2 + xPos) - y1 + rad ;

                    //Calculate the starting RGB values
                    ered = sred + ((rdiff*i) >> 8);
                    egreen = sgreen + ((gdiff*i) >> 8);
                    eblue = sblue + ((bdiff*i) >> 8);

                    color2 = RGBConvert(ered,egreen,eblue);
                    SetColor(color2);

                    switch(direction)    //Direction matter because different portions of
the circle are drawn
                    {
                       case GRAD_LEFT:
                           if(i>length)
                               SetColor(EndColor);
                           if(Bar(x1 - xCur, y1 - yNew, x1 - xPos, y2 + yNew) == 0)
                               return (0);
                            break;

                       case GRAD_RIGHT:
                       case GRAD_DOUBLE_VER:
                           if(i>length)
                               SetColor(EndColor);
                           if(Bar(x2 + xPos, y1 - yNew, x2 + xCur, y2 + yNew) == 0)
                               return (0);
                            break;

                       case GRAD_UP:
                            if(i>length)
                                SetColor(EndColor);
                            if(Bar(x1 - yNew, y1 - xCur, x2 + yNew, y1 - xPos) == 0)
                                return (0);
                            break;

                       case GRAD_DOWN:
                       case GRAD_DOUBLE_HOR:
                            if(i>length)
                                SetColor(EndColor);
                            if(Bar(x1 - yNew, y2 + xPos, x2 + yNew, y2 + xCur) == 0)
                                return (0);
                       default:
```

```
                        break;
                    }

                }
                state = Q8TOQ1;
                break;

            case Q8TOQ1:

                // 8th octant to 1st octant
                //if (_bevelDrawType & DRAWTOPBEVEL)
                {
                    i = (y1 - xCur) - y1 + rad;

                    //Calculate the starting RGB values
                    ered = sred + ((rdiff*i) >> 8);
                    egreen = sgreen + ((gdiff*i) >> 8);
                    eblue = sblue + ((bdiff*i) >> 8);

                    color2 = RGBConvert(ered,egreen,eblue);
                    SetColor(color2);

                    switch(direction)    //Direction matter because different portions of
the circle are drawn
                    {
                       case GRAD_LEFT:
                           if(i>length)
                               SetColor(EndColor);
                           if(Bar(x2 + xPos, y1 - yNew, x2 + xCur, y2 + yNew) == 0)
                               return (0);
                            break;

                       case GRAD_RIGHT:
                       case GRAD_DOUBLE_VER:
                           if(i>length)
                               SetColor(EndColor);
                           if(Bar(x1 - xCur, y1 - yNew, x1 - xPos, y2 + yNew) == 0)
                               return (0);
                            break;

                       case GRAD_UP:
                            if(i>length)
                                SetColor(EndColor);
                            if(Bar(x1 - yNew, y2 + xPos, x2 + yNew, y2 + xCur) == 0)
                                return (0);
                            break;

                       case GRAD_DOWN:
                       case GRAD_DOUBLE_HOR:
                            if(i>length)
                                SetColor(EndColor);
                            if(Bar(x1 - yNew, y1 - xCur, x2 + yNew, y1 - xPos) == 0)
                                return (0);
                            break;

                       default:
                           break;
                    }

                }
                state = Q7TOQ2;
                break;

            case Q7TOQ2:

                // 7th octant to 2nd octant
                //if (_bevelDrawType & DRAWTOPBEVEL)
                {
                    i = (y1 - yNew) - y1 + rad;

                    //Calculate the starting RGB values
                    ered = sred + ((rdiff*i) >> 8);
```

```
                    egreen = sgreen + ((gdiff*i) >> 8);
                    eblue = sblue + ((bdiff*i) >> 8);

                    color2 = RGBConvert(ered,egreen,eblue);

                    SetColor(color2);

                    switch(direction)    //Direction matter because different portions of
the circle are drawn
                    {
                       case GRAD_LEFT:
                           if(i>length)
                               SetColor(EndColor);
                           if(Bar(x2 + yCur, y1 - xCur, x2 + yNew, y2 + xCur) == 0)
                               return (0);
                            break;

                       case GRAD_RIGHT:
                       case GRAD_DOUBLE_VER:
                           if(i>length)
                               SetColor(EndColor);
                           if(Bar(x1 - yNew, y1 - xCur, x1 - yCur, y2 + xCur) == 0)
                               return (0);
                            break;

                       case GRAD_UP:
                            if(i>length)
                                SetColor(EndColor);
                            if(Bar(x1 - xCur, y2 + yCur, x2 + xCur, y2 + yNew) == 0)
                                return (0);
                            break;

                       case GRAD_DOWN:
                       case GRAD_DOUBLE_HOR:
                            if(i>length)
                                SetColor(EndColor);
                            if(Bar(x1 - xCur, y1 - yNew, x2 + xCur, y1 - yCur) == 0)
                                return (0);
                            break;

                       default:
                           break;
                    }

                }
                // update current values
                xCur = xPos;
                yCur = yPos;
                state = CHECK;
                break;



            case FACE:
                if((x2 - x1) || (y2 - y1))
                {
                i = (y1) - y1 + rad;
                //Calculate the starting RGB values
                ered = sred + ((rdiff*i) >> 8);
                egreen = sgreen + ((gdiff*i) >> 8);
                eblue = sblue + ((bdiff*i) >> 8);

                color1 = RGBConvert(ered,egreen,eblue);

                if(i>length)
                    color1 = EndColor;

                i = (y2) - y1 + rad;
                //Calculate the ending RGB values
                ered = sred + ((rdiff*i) >> 8);
                egreen = sgreen + ((gdiff*i) >> 8);
                eblue = sblue + ((bdiff*i) >> 8);
```

```
                color2 = RGBConvert(ered,egreen,eblue);

                if(i>length)
                    color2 = EndColor;

                length -= rad;   //Subtract the radius from the length needed for
gradient

                if(direction == GRAD_UP || direction == GRAD_DOWN || direction ==
GRAD_DOUBLE_HOR)
                {
                   if(length>= y2-y1)
                   {
                       length = 100;
                   }
                   else
                   {
                       length *= 100;
                       length /= (y2 -y1);
                   }
                   BarGradient(x1-rad, y1, x2+rad, y2,color1,color2,length,direction);
                }
                else
                {
                  if(length>=x2-x1)
                  {
                      length = 100;
                  }
                  else
                  {
                       length *= 100;
                       length /= (x2 -x1);
                  }
                   BarGradient(x1, y1-rad, x2, y2+rad,color1,color2,length,direction);
                }

                    state = WAITFORDONE;
                }
                else
                {
                    state = BEGIN;
                    return (1);
                }

            case WAITFORDONE:
                if(IsDeviceBusy())
                    return (0);
                state = BEGIN;
                return (1);
        }           // end of switch
    }               // end of while

}
#endif


#ifdef USE_COMP_RLE

    #ifdef USE_BITMAP_FLASH

        #if (COLOR_DEPTH >= 8)
/*********************************************************************
* Function: WORD DecodeRLE8(FLASH_BYTE *flashAddress, BYTE *pixelrow, WORD size)
*
* PreCondition: tempFlashAddress should point to the beginning of a RLE compressed block
*
* Input: flashAddress - Address of the beginning of a RLE compressed block
*        pixelrow - Pointer to an array where the decoded row must be stored
*        size - Size of the decoded data in bytes
*
* Output: Number of source bytes traversed
```

```
*
* Side Effects: none
*
* Overview: Decodes the data
*
*********************************************************************/
WORD DecodeRLE8(FLASH_BYTE *flashAddress, BYTE *pixel_row, WORD size)
{
    WORD sourceOffset = 0;
    WORD decodeSize = 0;

    while(decodeSize < size)
    {
        BYTE code = *flashAddress++;
        BYTE value = *flashAddress++;
        sourceOffset += 2;

        if(code > 0)
        {
            decodeSize += code;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                code -= (decodeSize - size);
            }

            while(code)
            {
                *pixel_row++ = value;
                code--;
            }
        }
        else
        {
            decodeSize += value;
            sourceOffset += value;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                value -= (decodeSize - size);
            }

            while(value)
            {
                *pixel_row++ = *flashAddress++;
                value--;
            }
        }
    }
    return (sourceOffset);
}

/*********************************************************************
* Function: void PutImageRLE8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - Should be NORMAL when RLE is used
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs the image starting from left,top coordinates
*
* Note: image must be located in internal memory
*
*********************************************************************/
void PutImageRLE8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
```

```
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                xc, yc;
    BYTE                temp;
    BYTE                stretchX, stretchY;
    GFX_COLOR           pallete[256];
    WORD                counter;
    BYTE                pixelrow[GetMaxX() + 1];
    WORD                offset;
    BYTE                *pixelAddress;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    #if COLOR_DEPTH == 24
    flashAddress += 2;      // pad word for alignment
    #endif

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        #if COLOR_DEPTH == 16
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
        #endif

        #if COLOR_DEPTH == 24
        pallete[counter] = *((FLASH_DWORD *)flashAddress);
        flashAddress += 4;
        #endif
    }

    yc = top;
    tempFlashAddress = flashAddress;
    for(y = 0; y < sizeY; y++)
    {
        offset = DecodeRLE8(tempFlashAddress, pixelrow, sizeX);
        tempFlashAddress += offset;

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pixelAddress = pixelrow;
            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *pixelAddress;
                pixelAddress++;

                // Set color
            #ifdef USE_PALETTE
                SetColor(temp);
            #else
                SetColor(pallete[temp]);
            #endif

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    #ifdef USE_TRANSPARENT_COLOR
                        if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                    #endif
                            PutPixel(xc, yc);
```

```
                        xc++;
                    }
                }

                yc++;
            }
        }
    }
        #endif //#if (COLOR_DEPTH >= 8)

        #if (COLOR_DEPTH >= 4)

/*********************************************************************
* Function: WORD DecodeRLE4(FLASH_BYTE *flashAddress, BYTE *pixelrow, WORD size)
*
* PreCondition: tempFlashAddress should point to the beginning of a RLE compressed block
*
* Input: flashAddress - Address of the beginning of a RLE compressed block
*        pixelrow - Pointer to an array where the decoded row must be stored
*        size - Size of the decoded data in bytes
*
* Output: Number of source bytes traversed
*
* Side Effects: none
*
* Overview: Decodes the data
*
*********************************************************************/
WORD DecodeRLE4(FLASH_BYTE *flashAddress, BYTE *pixel_row, WORD size)
{
    WORD sourceOffset = 0;
    WORD decodeSize = 0;

    while(decodeSize < size)
    {
        BYTE code = *flashAddress++;
        BYTE value = *flashAddress++;
        BYTE counter;

        sourceOffset += 2;

        if(code > 0)
        {
            decodeSize += code;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                code -= (decodeSize - size);
            }

            for(counter = 0; counter < code; counter++)
            {
                if(counter & 0x01)
                {
                    *pixel_row++ = (value >> 4) & 0x0F;
                }
                else
                {
                    *pixel_row++ = (value) & 0x0F;
                }
            }
        }
        else
        {
            decodeSize += value;
            sourceOffset += (value + 1) >> 1;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                value -= (decodeSize - size);
            }
```

```
            for(counter = 0; counter < value; counter++)
            {
                if(counter & 0x01)
                {
                    *pixel_row++ = (*flashAddress >> 4) & 0x0F;
                    flashAddress++;
                }
                else
                {
                    *pixel_row++ = (*flashAddress) & 0x0F;
                }
            }
        }
    }

    return (sourceOffset);
}

/*************************************************************************
* Function: void PutImageRLE4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - Should be NORMAL when RLE is used
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs the image starting from left,top coordinates
*
* Note: image must be located in internal memory
*
*************************************************************************/
void PutImageRLE4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                xc, yc;
    BYTE                temp = 0;
    register BYTE       stretchX, stretchY;
    GFX_COLOR           pallete[16];
    WORD                counter;
    BYTE                pixelrow[GetMaxX() + 1];
    WORD                offset;
    BYTE                *pixelAddress;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    #if COLOR_DEPTH == 24
    flashAddress += 2;          // pad for alignment
    #endif

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        #if COLOR_DEPTH == 16
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
        #endif

        #if COLOR_DEPTH == 24
```

```
        pallete[counter] = *((FLASH_DWORD *)flashAddress);
        flashAddress += 4;
        #endif
    }

    yc = top;
    tempFlashAddress = flashAddress;
    for(y = 0; y < sizeY; y++)
    {
        offset = DecodeRLE4(tempFlashAddress, pixelrow, sizeX);
        tempFlashAddress += offset;

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pixelAddress = pixelrow;
            xc = left;
            for(x = 0; x < sizeX; x++)
            {
                temp = *pixelAddress;
                pixelAddress++;

                #ifdef USE_PALETTE
                    SetColor(temp);
                #else
                    SetColor(pallete[temp]);
                #endif

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    #ifdef USE_TRANSPARENT_COLOR
                        if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                    #endif
                            PutPixel(xc, yc);
                        xc++;
                }
            }

            yc++;
        }
    }
}
        #endif //#if (COLOR_DEPTH >= 4)

    #endif //    #ifdef USE_BITMAP_FLASH

    #ifdef USE_BITMAP_EXTERNAL

        #if (COLOR_DEPTH >= 8)

/*****************************************************************************
* Function: WORD DecodeRLE8Ext(void *image, DWORD  memAddress, BYTE *pixelrow, WORD size)
*
* PreCondition: tempFlashAddress should point to the beginning of a RLE compressed block
*
* Input: image - External memory image pointer
*        memAddress - Address of the beginning of a RLE compressed block
*        pixelrow - Pointer to an array where the decoded row must be stored
*        size - Size of the decoded data in bytes
*
* Output: Number of source bytes traversed
*
* Side Effects: none
*
* Overview: Decodes the data from the external flash
*
*****************************************************************************/
WORD DecodeRLE8Ext(void *image, DWORD memAddress, BYTE *pixel_row, WORD size)
{
    WORD sourceOffset = 0;
    WORD decodeSize = 0;
```

```
    while(decodeSize < size)
    {
        BYTE code, value;

        ExternalMemoryCallback(image, memAddress, sizeof(BYTE), &code);
        memAddress++;
        ExternalMemoryCallback(image, memAddress, sizeof(BYTE), &value);
        memAddress++;

        sourceOffset += 2;

        if(code > 0)
        {
            decodeSize += code;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                code -= (decodeSize - size);
            }

            while(code)
            {
                *pixel_row++ = value;
                code--;
            }
        }
        else
        {
            decodeSize += value;
            sourceOffset += value;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                value -= (decodeSize - size);
            }

            ExternalMemoryCallback(image, memAddress, value * sizeof(BYTE), pixel_row);
            pixel_row += value;
            memAddress += value;
        }
    }
    return (sourceOffset);
}

/*********************************************************************
* Function: void PutImageRLE8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*********************************************************************/
void PutImageRLE8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[256];
    BYTE            pixelrow[(GetMaxX() + 1)];
    BYTE            *pixelAddress;

    BYTE            temp;
    WORD            sizeX, sizeY;
```

```
    WORD            x, y;
    WORD            xc, yc;
    BYTE            stretchX, stretchY;
    WORD            offset;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    yc = top;
    for(y = 0; y < sizeY; y++)
    {
        offset = DecodeRLE8Ext(image, memOffset, pixelrow, sizeX);
        memOffset += offset;

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pixelAddress = pixelrow;
            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *pixelAddress;
                pixelAddress++;

                // Set color
            #ifdef USE_PALETTE
                SetColor(temp);
            #else
                SetColor(pallete[temp]);
            #endif

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    #ifdef USE_TRANSPARENT_COLOR
                        if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                    #endif
                            PutPixel(xc, yc);
                        xc++;
                }
            }

            yc++;
        }
    }
}
        #endif //#if (COLOR_DEPTH >= 8)

        #if (COLOR_DEPTH >= 4)

/*****************************************************************
* Function: WORD DecodeRLE4Ext(void *image, DWORD memAddress, BYTE *pixelrow, WORD size)
*
* PreCondition: tempFlashAddress should point to the beginning of a RLE compressed block
*
* Input: image - External memory image pointer
*        memAddress - Address of the beginning of a RLE compressed block
*        pixelrow - Pointer to an array where the decoded row must be stored
*        size - Size of the decoded data in bytes
*
```

```
 * Output: Number of source bytes traversed
 *
 * Side Effects: none
 *
 * Overview: Decodes the data
 *
 ********************************************************************/
WORD DecodeRLE4Ext(void *image, DWORD memAddress, BYTE *pixel_row, WORD size)
{
    WORD sourceOffset = 0;
    WORD decodeSize = 0;

    while(decodeSize < size)
    {
        BYTE code, value;
        BYTE counter, temp;

        ExternalMemoryCallback(image, memAddress, sizeof(BYTE), &code);
        memAddress++;
        ExternalMemoryCallback(image, memAddress, sizeof(BYTE), &value);
        memAddress++;

        sourceOffset += 2;

        if(code > 0)
        {
            decodeSize += code;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                code -= (decodeSize - size);
            }

            for(counter = 0; counter < code; counter++)
            {
                if(counter & 0x01)
                {
                    *pixel_row++ = (value >> 4) & 0x0F;
                }
                else
                {
                    *pixel_row++ = (value) & 0x0F;
                }
            }
        }
        else
        {
            decodeSize += value;
            sourceOffset += (value + 1) >> 1;

            if(decodeSize > size) //To avoid writing oustide the array bounds
            {
                value -= (decodeSize - size);
            }

            for(counter = 0; counter < value; counter++)
            {
                if(counter & 0x01)
                {
                    *pixel_row++ = (temp >> 4) & 0x0F;
                    memAddress++;
                }
                else
                {
                    ExternalMemoryCallback(image, memAddress, sizeof(BYTE), &temp);
                    *pixel_row++ = temp & 0x0F;
                }
            }
        }
    }

    return (sourceOffset);
```

```
}

/*****************************************************************
* Function: void PutImageRLE4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************/
void PutImageRLE4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[16];
    BYTE            pixelrow[(GetMaxX() + 1)];
    BYTE            *pixelAddress;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            xc, yc;
    BYTE            stretchX, stretchY;
    WORD            offset;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    yc = top;
    for(y = 0; y < sizeY; y++)
    {
        offset = DecodeRLE4Ext(image, memOffset, pixelrow, sizeX);
        memOffset += offset;

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pixelAddress = pixelrow;
            xc = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *pixelAddress;
                pixelAddress++;

                // Set color
            #ifdef USE_PALETTE
                SetColor(temp);
            #else
                SetColor(pallete[temp]);
            #endif

                // Write pixel to screen
```

```
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    #ifdef USE_TRANSPARENT_COLOR
                        if (!((GetTransparentColor() == GetColor()) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE)))
                    #endif
                            PutPixel(xc, yc);
                        xc++;
                }
            }

            yc++;
        }
    }
}
        #endif //#if (COLOR_DEPTH >= 4)

    #endif //#ifdef USE_BITMAP_EXTERNAL

#endif //#ifdef USE_COMP_RLE

void GetCirclePoint(SHORT radius, SHORT angle, SHORT *x, SHORT *y)
{
    DWORD   rad;
    SHORT   ang;
    SHORT   temp;

    ang = angle % 45;
    if((angle / 45) & 0x01)
        ang = 45 - ang;

    rad = radius;
    // there is a shift by 8 bits here since this function assumes a shift on the
calculations for accuracy
    // and to avoid long and complex multiplications.
    rad *= ((DWORD) GetSineCosine(ang, GETCOSINE) << 8);

    *x = ((DWORD_VAL) rad).w[1];
    rad = radius;
    rad *= ((DWORD) GetSineCosine(ang, GETSINE) << 8);

    *y = ((DWORD_VAL) rad).w[1];

    if(((angle > 45) && (angle < 135)) || ((angle > 225) && (angle < 315)))
    {
        temp = *x;
        *x = *y;
        *y = temp;
    }

    if((angle > 90) && (angle < 270))
    {
        *x = -*x;
    }

    if((angle > 180) && (angle < 360))
    {
        *y = -*y;
    }
}
```

14.2.2 Primitive.h

Enumerations

Name	Description
GFX_GRADIENT_TYPE (see page 371)	Enumeration for gradient type

GFX_RESOURCE (see page 368)	Memory type enumeration to determine the source of data. Used in interpreting bitmap and font from different memory sources.

Functions

	Name	Description
	AlphaBlendWindow (see page 400)	This alphablends a foreground and a background stored in frames to a destination window. The function uses windows insides frames. Each window shares the same width and height parameters.
	Arc (see page 356)	Draws the octant arc of a beveled figure with given centers, radii and octant mask. When r1 is zero and r2 has some value, a filled circle is drawn; when the radii have values, an arc of thickness (r2-r1) is drawn; when octant = 0xFF, a full ring is drawn. When r1 and r2 are zero, a rectangular object is drawn, where xL, yT specifies the left top corner; xR, yB specifies the right bottom corner.
	Bar (see page 352)	This function draws a bar given the left, top and right, bottom corners with the current color.
	Bevel (see page 357)	Draws a beveled figure on the screen. For a pure circular object x1 = x2 and y1 = y2. For a rectangular object radius = 0.
	ClearDevice (see page 367)	This function clears the screen with the current color and sets the graphic cursor position to (0, 0). Clipping is NOT supported by ClearDevice().
	DrawPoly (see page 354)	This function draws a polygon with the current line type using the given number of points. The polygon points are stored in an array arranged in the following order:
	FillBevel (see page 358)	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.
	GetImageHeight (see page 363)	This function returns the image height.
	GetImageWidth (see page 363)	This function returns the image width.
	GetTextHeight (see page 344)	This macro returns the height of the specified font. All characters in a given font table have a constant height.
	GetTextWidth (see page 345)	This function returns the width of the specified string for the specified font. The string must be terminated by a line feed or zero.
	GFXGetPageOriginAddress (see page 402)	This function calculates the address of a certain 0,0 location of a page in memory
	GFXGetPageXYAddress (see page 402)	This function calculates the address of a certain x,y location in memory
	InitGraph (see page 367)	This function initializes the display controller, sets the line type to SOLID_LINE (see page 351), sets the screen to all BLACK (see page 389), sets the current drawing color to WHITE (see page 394), 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.
	Line (see page 347)	This function draws a line with the current line type from the start point to the end point.
	OutChar (see page 342)	This function outputs a character from the current graphic cursor position. OutChar() uses the current active font set with SetFont (see page 340)().
	OutText (see page 343)	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 (see page 340)().

	OutTextXY (see page 343)	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 (see page 340)().
	PutImage (see page 362)	This function outputs image starting from left,top coordinates.
	SetFont (see page 340)	This function sets the current font used in OutTextXY (see page 343)(), OutText (see page 343)() and OutChar (see page 342)() functions.

Macros

Name	Description
Circle (see page 355)	This macro draws a circle with the given center and radius.
DASHED_LINE (see page 351)	Dashed Line (see page 347) Style
DOTTED_LINE (see page 351)	Dotted Line (see page 347) Style
EXTERNAL_FONT_BUFFER_SIZE (see page 366)	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.
FillCircle (see page 355)	This macro draws a filled circle. Uses the FillBevel (see page 358)() function.
GetFontOrientation (see page 341)	This macro returns font orientation (0 == horizontal, 1 == vertical).
GetX (see page 360)	This macro returns the current graphic cursor x-coordinate.
GetY (see page 360)	This macro returns the current graphic cursor y-coordinate.
IMAGE_NORMAL (see page 365)	Normal image stretch code
IMAGE_X2 (see page 365)	Stretched image stretch code
LineRel (see page 348)	This macro draws a line with the current line type from the current graphic cursor position to the position defined by displacement.
LineTo (see page 349)	This macro draws a line with the current line type from the current graphic cursor position to the given x, y position.
MoveRel (see page 361)	This macro moves the graphic cursor relative to the current location. The given dX and dY displacement can be positive or negative numbers.
MoveTo (see page 361)	This macro moves the graphic cursor to new x,y position.
NORMAL_LINE (see page 351)	Normal Line (see page 347) (thickness is 1 pixel)
Rectangle (see page 353)	This macro draws a rectangle with the given left, top and right, bottom corners. Current line type is used.
SetBevelDrawType (see page 359)	This macro sets the fill bevel type to be drawn.
SetFontOrientation (see page 341)	This macro sets font orientation vertical or horizontal.
SetLineThickness (see page 349)	This macro sets sets line thickness to 1 pixel or 3 pixels.
SetLineType (see page 350)	This macro sets the line type to draw.
SOLID_LINE (see page 351)	Solid Line (see page 347) Style
THICK_LINE (see page 352)	Thick Line (see page 347) (thickness is 3 pixels)
XCHAR (see page 345)	This macro sets the data type for the Fonts. There are three types • #define XCHAR unsigned short // use multibyte characters (0-2^16 range) • #define XCHAR unsigned char // use unsigned char (0-255 range) • #define XCHAR char // use signed char (0-127 range)

Structures

Name	Description
BITMAP_HEADER (see page 364)	Structure describing the bitmap header.
FONT_FLASH (see page 340)	Structure for font stored in FLASH memory.
FONT_HEADER (see page 339)	Structure describing the font header.

GFX_EXTDATA (see page 370)	This structure is used to describe external memory.
GFX_GRADIENT_STYLE (see page 371)	This structure is used to describe the gradient style.
GFX_IMAGE_HEADER (see page 368)	Structure for images stored in various system memory (Flash, External Memory (SPI, Parallel Flash, or memory in EPMP).
IMAGE_FLASH (see page 369)	Structure for images stored in FLASH memory.
IMAGE_RAM (see page 370)	Structure for images stored in RAM memory.

Types

Name	Description
FONT_EXTERNAL (see page 340)	Structure for font stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)
IMAGE_EXTERNAL (see page 372)	Structure for images stored in EXTERNAL memory space. (example: External SPI or parallel Flash, EDS_EPMP)

Description

This is file Primitive.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Graphic Primitives Layer
 *****************************************************************************
 * FileName:        Primitive.h
 * Processor:       PIC24F, PIC24H, dsPIC, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07    Version 1.0 release
 * 18/05/09    Version 1.1 - All Primitive functions have
 *                           return value(To support 2d-Acceleration)
 * 05/12/10    Added sine and cosine functions.
 * 08/20/10    Modified TYPE_MEMORY enum for more types
 *             and changed the enumeration to GFX_RESOURCE. Also removed
 *             unused types.
 *             Added new structure GFX_IMAGE_HEADER.
 *             Deprecated the following (renamed):
 *                TYPE_MEMORY
 *                EXTDATA
 *                BITMAP_FLASH
```

```
 *                BITMAP_RAM
 *                BITMAP_EXTERNAL
 * 03/09/11    Removed USE_DRV_xxx.
 * 04/01/11    Added GetSineCosine().
 ****************************************************************************/
#ifndef _PRIMITIVE_H
    #define _PRIMITIVE_H

#include "GenericTypeDefs.h"
#include "GraphicsConfig.h"
#include "gfxcolors.h"
/*********************************************************************
* Overview: Primitive lines are drawn using line type and line thickness.
*           There are 3 line styles and 2 types of line thickness.
*
*********************************************************************/

// Solid Line Style
    #define SOLID_LINE  0

// Dotted Line Style
    #define DOTTED_LINE 1

// Dashed Line Style
    #define DASHED_LINE 4

// Normal Line (thickness is 1 pixel)
    #define NORMAL_LINE 0

// Thick Line (thickness is 3 pixels)
    #define THICK_LINE  1

/*********************************************************************
* Overview: Drawing bitmaps will have two modes, normal rendering
*           and stretched rendering. Stretched rendering effectively
*           doubles the image size in the horizontal and vertical
*           direction.
*
*********************************************************************/

// Normal image stretch code
    #define IMAGE_NORMAL    1

// Stretched image stretch code
    #define IMAGE_X2    2

// Current line style
extern SHORT    _lineType;

// Current line thickness
extern BYTE     _lineThickness;

// constants used for circle/arc computation
    #define SIN45       46341   // sin(45) * 2^16)
    #define ONEP25      81920   // 1.25 * 2^16
// constants used to get sine(v) and cosine(v)
    #define GETSINE     0x00
    #define GETCOSINE   0x01

// Current cursor coordinates
extern SHORT    _cursorX;
extern SHORT    _cursorY;

// Font orientation
extern BYTE     _fontOrientation;

    #define ORIENT_HOR  0
    #define ORIENT_VER  1

/*********************************************************************
* Overview: This macro sets the data type for the Fonts.
*           There are three types
```

```
*           - #define XCHAR   unsigned short        // use multibyte characters (0-2^16
range)
*           - #define XCHAR   unsigned char     // use unsigned char (0-255 range)
*           - #define XCHAR   char              // use signed char (0-127 range)
*
*********************************************************************/
    #if defined (USE_MULTIBYTECHAR)
        #define XCHAR   unsigned short
    #elif defined (USE_UNSIGNED_XCHAR)
        #define XCHAR   unsigned char
    #else
        #define XCHAR   char
    #endif

// bevel drawing type (0 = full bevel, 0xF0 - top bevel only, 0x0F - bottom bevel only
extern BYTE _bevelDrawType;

    #define DRAWFULLBEVEL   0xFF
    #define DRAWTOPBEVEL    0xF0
    #define DRAWBOTTOMBEVEL 0x0F

    #ifdef __PIC32MX__
        // Flash data with 32bit pointers
        #define FLASH_BYTE  const BYTE
        #define FLASH_WORD  const WORD
        #define FLASH_DWORD const DWORD
    #else
        // Flash data with 24bit pointers
        #define FLASH_BYTE  char __prog__
        #define FLASH_WORD  short int __prog__
        #define FLASH_DWORD unsigned long __prog__
    #endif

/*********************************************************************
* Overview: Memory type enumeration to determine the source of data.
*           Used in interpreting bitmap and font from different
*           memory sources.
*
*********************************************************************/
typedef enum
{
    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
                            // 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)
} GFX_RESOURCE;

#define GFX_COMP_MASK   0xF000
#define GFX_MEM_MASK    0x000F

/*********************************************************************
* Overview: This structure is used to describe external memory.
*
*********************************************************************/
typedef struct
{
    GFX_RESOURCE type;       // must be EXTERNAL or 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
                             //   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)
} GFX_EXTDATA;

/* &&&&
```

```
/*****************************************************************
* Overview: Structure describing the glyph entry.
*
*****************************************************************/
typedef struct
{
    BYTE    width;          // width of the glyph
    BYTE    offsetLSB;      // Least Significant Byte of the glyph location offset
    WORD    offsetMSB;      // Most Significant (2) Bytes of the glyph location offset
} GLYPH_ENTRY;

/*****************************************************************
* Overview: Structure for font stored in FLASH memory.
*
*****************************************************************/
typedef struct
{
    GFX_RESOURCE  type;     // must be FLASH
    const char    *address; // font image address in FLASH
} FONT_FLASH;

/*****************************************************************
* Overview: Structure for font stored in RAM memory.
*
*****************************************************************/
typedef struct
{
    GFX_RESOURCE  type;     // must be RAM
    char          *address; // font image address in RAM
} FONT_RAM;

/*****************************************************************
* Overview: Structure for font stored in EXTERNAL memory space.
*           (example: External SPI or parallel Flash, EDS_EPMP)
*
*****************************************************************/
typedef GFX_EXTDATA FONT_EXTERNAL;

/*****************************************************************
* Overview: Structure for images stored in various system memory
*           (Flash, External Memory (SPI, Parallel Flash,
*           or memory in EPMP).
*
*****************************************************************/
typedef struct
{
    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
    union
    {
        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
#if defined(__PIC24F__)
        __eds__ char *edsAddress;      // for addresses in EDS
#endif
    } LOCATION;

    WORD             width;            // width of the image
    WORD             height;           // height of the image
    DWORD            param1;           // size of the IPU compressed data
    DWORD            param2;           // size of the IPU decompressed data
```

```
    WORD            colorDepth;       // color depth of the image
} GFX_IMAGE_HEADER;

/* &&&&
```

```
WORD            Arc(SHORT xL, SHORT yT, SHORT xR, SHORT yB, SHORT r1, SHORT r2, BYTE
octant);

/*********************************************************************
* Function:  void InitGraph(void)
*
* 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.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void            InitGraph(void);

/*********************************************************************
* Macros:  GetX()
*
* Overview: This macro returns the current graphic cursor x-coordinate.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define GetX()  _cursorX

/*********************************************************************
* Macros:  GetX()
*
* Overview: This macro returns the current graphic cursor y-coordinate.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define GetY()  _cursorY

/*********************************************************************
* Macros:  MoveTo(x,y)
*
* Overview: This macro moves the graphic cursor to new x,y position.
*
* PreCondition: none
*
* Input: x - Specifies the new x position of the graphic cursor.
*        y - Specifies the new y position of the graphic cursor.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
```

```
    #define MoveTo(x, y) \
    _cursorX = x;        \
    _cursorY = y;

/*****************************************************************
* Macros:  MoveRel(dX,dY)
*
* Overview: This macro moves the graphic cursor relative to the
*           current location. The given dX and dY displacement can
*           be positive or negative numbers.
*
* PreCondition: none
*
* Input: dX - Specifies the displacement of the graphic cursor for
*             the horizontal direction.
*        dY - Specifies the displacement of the graphic cursor for
*             the vertical direction.
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
    #define MoveRel(dX, dY) \
    _cursorX += dX;         \
    _cursorY += dY;

/*****************************************************************
* Macro: SetFontOrientation(orient)
*
* Overview: This macro sets font orientation vertical or horizontal.
*
* PreCondition: none
*
* Input: orient - should be non-zero if the font orientation is vertical
*
* Output: none
*
*****************************************************************/
    #define SetFontOrientation(orient)  _fontOrientation = orient;

/*****************************************************************
* Macro: GetFontOrientation()
*
* Overview: This macro returns font orientation (0 == horizontal, 1 == vertical).
*
* PreCondition: none
*
* Input: none
*
* Output: font orientation (0 == horizontal, 1 == vertical)
*
*****************************************************************/
    #define GetFontOrientation()    _fontOrientation

/*****************************************************************
* Function: WORD OutChar(XCHAR ch)
*
* Overview: This function outputs a character from the current graphic
*           cursor position. OutChar() uses the current active font
*           set with SetFont().
*
* PreCondition: none
*
* Input: ch - The character code to be displayed.
*
* Output: 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.
*
```

```
 *
 * 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.
 *
 ********************************************************************/
WORD    OutChar(XCHAR ch);

/********************************************************************
 * Function: WORD OutText(XCHAR* textString)
 *
 * 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().
 *
 *
 * Input: textString - Pointer to the string to be displayed.
 *
 * Output: 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.
 *
 * 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.
 *
 ********************************************************************/
WORD    OutText(XCHAR *textString);

/********************************************************************
 * Function: WORD OutTextXY(SHORT x, SHORT y, XCHAR* textString)
 *
 * 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().
 *
 * Input: x - Defines the x starting position of the string.
 *        y - Defines the y starting position of the string.
 *        textString - Pointer to the string to be displayed.
 *
 * Output: 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:
 *   <CODE>
 *   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);
*   }
*   </CODE>
*
* 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.
*
*********************************************************************/
WORD    OutTextXY(SHORT x, SHORT y, XCHAR *textString);

/*********************************************************************
* Function: SHORT GetTextHeight(void* font)
*
* Overview: This macro returns the height of the specified font.
*           All characters in a given font table have a constant
*           height.
*
* Input: font - Pointer to the font image.
*
* Output: Returns the font height.
*
* Example:
*   See OutTextXY() example.
*
* Side Effects: none
*
*********************************************************************/
SHORT   GetTextHeight(void *font);

/*********************************************************************
* Function: SHORT GetTextWidth(XCHAR* textString, void* font)
*
* 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.
*
* Input: textString - Pointer to the string.
*        font - Pointer to the font image.
*
* Output: Returns the string width in the specified font.
*
* Example:
*   See OutTextXY() example.
*
* Side Effects: none
*
*********************************************************************/
SHORT   GetTextWidth(XCHAR *textString, void *font);

/*********************************************************************
* Function: void SetFont(void* font)
*
* Overview: This function sets the current font used in OutTextXY(),
*           OutText() and OutChar() functions.
*
* Input: font - Pointer to the new font image to be used.
*
* Output: none
*
* Example:
*   See OutTextXY() example.
*
* Side Effects: none
*
*********************************************************************/
```

```
void    SetFont(void *font);

/*****************************************************************
* Macros: SetLineType(lnType)
*
* Overview: This macro sets the line type to draw.
*
* Input: lnType - The type of line to be used.
*           Supported line types:
*               - SOLID_LINE
*               - DOTTED_LINE
*               - DASHED_LINE
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
    #define SetLineType(lnType) _lineType = lnType;

/*****************************************************************
* Macros: SetLineThickness(lnThickness)
*
* Overview: This macro sets sets line thickness to 1 pixel or 3 pixels.
*
* Input: lnThickness - Line thickness code (0 - 1 pixel; 1 - 3 pixels)
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
    #define SetLineThickness(lnThickness)   _lineThickness = lnThickness;

/*****************************************************************
* Function: WORD Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
*
* Overview: This function draws a line with the current line type
*           from the start point to the end point.
*
* Input: x1 - x coordinate of the start point.
*        y1 - y coordinate of the start point.
*        x2 - x coordinate of the end point.
*        y2 - y coordinate of the end point.
*
* Output: 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.
*
*
* Side Effects: The graphic cursor position is moved to the end
*               point of the line.
*
*****************************************************************/
WORD Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2);

/*****************************************************************
* Macros: LineRel(dX, dY)
*
* Overview: This macro draws a line with the current line type from
*           the current graphic cursor position to the position defined
*           by displacement.
*
* Input: dX - Displacement from the current x position.
*        dY - Displacement from the current y position.
*
* Output: 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.
*
* Side Effects: The graphic cursor position is moved to the end
*               point of the line.
*
*********************************************************************/
    #define LineRel(dX, dY) Line(GetX(), GetY(), GetX() + dX, GetY() + dY)

/*********************************************************************
* Macros: LineTo(x,y)
*
* Overview: This macro draws a line with the current line type from
*           the current graphic cursor position to the given x, y position.
*
* Input: x - End point x position.
*        y - End point y poisiton.
*
* Output: 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.
*
* Side Effects: The graphic cursor position is moved to the end
*               point of the line.
*
*********************************************************************/
    #define LineTo(x, y)    Line(_cursorX, _cursorY, x, y)

/*********************************************************************
* Macro: Circle(x, y, radius)
*
* Overview: This macro draws a circle with the given center and radius.
*
* Input: x - Center x position.
*        y - Center y position.
*        radius - the radius of the circle.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
    #define Circle(x, y, radius)    Bevel(x, y, x, y, radius)

/*********************************************************************
* Macro: SetBevelDrawType(type)
*
* Overview: This macro sets the fill bevel type to be drawn.
*
* Input: type - is set using the following.
*        - DRAWFULLBEVEL to draw the full shape
*        - DRAWTOPBEVEL to draw the upper half portion
*        - DRAWBOTTOMBEVEL to draw the lower half portion
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define SetBevelDrawType(type)      (_bevelDrawType = type)

/*********************************************************************
* Function: WORD Bevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
*
* Overview: Draws a beveled figure on the screen.
*           For a pure circular object x1 = x2 and y1 = y2.
*           For a rectangular object radius = 0.
```

```
 *
 * Input: x1 - x coordinate position of the upper left center of the circle that
 *             draws the rounded corners.
 *        y1 - y coordinate position of the upper left center of the circle that
 *             draws the rounded corners.
 *        x2 - x coordinate position of the lower right center of the circle that
 *             draws the rounded corners.
 *        y2 - y coordinate position of the lower right center of the circle that
 *             draws the rounded corners.
 *        rad - defines the redius of the circle, that draws the rounded corners.
 *
 * Output: 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.
 *
 *
 * Side Effects: none
 *
 ***********************************************************************/
WORD    Bevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad);

/***********************************************************************
 * Function: WORD FillBevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad)
 *
 * Overview: 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.
 *
 * Input: x1 - x coordinate position of the upper left center of the circle that
 *             draws the rounded corners.
 *        y1 - y coordinate position of the upper left center of the circle that
 *             draws the rounded corners.
 *        x2 - x coordinate position of the lower right center of the circle that
 *             draws the rounded corners.
 *        y2 - y coordinate position of the lower right center of the circle that
 *             draws the rounded corners.
 *        rad - defines the redius of the circle, that draws the rounded corners.
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 ***********************************************************************/
WORD    FillBevel(SHORT x1, SHORT y1, SHORT x2, SHORT y2, SHORT rad);

/***********************************************************************
 * Macro: FillCircle(SHORT x1, SHORT y1, SHORT rad)
 *
 * Overview: This macro draws a filled circle. Uses the FillBevel() function.
 *
 * Input: x1 - x coordinate position of the center of the circle.
 *        y1 - y coordinate position of the center of the circle.
 *        rad - defines the redius of the circle.
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 ***********************************************************************/
    #define FillCircle(x1, y1, rad) FillBevel(x1, y1, x1, y1, rad)

/***********************************************************************
```

```
* Macro: Rectangle(left, top, right, bottom)
*
* Overview: This macro draws a rectangle with the given left,
*           top and right, bottom corners. Current line type is used.
*
* Input: 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.
*
* Output: 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.
*
* Side Effects: none
*
*********************************************************************/
    #define Rectangle(left, top, right, bottom) Bevel(left, top, right, bottom, 0)

/*********************************************************************
* Function: WORD DrawPoly(SHORT numPoints, SHORT* polyPoints)
*
* Overview: This function draws a polygon with the current line
*           type using the given number of points. The polygon points
*           are stored in an array arranged in the following order:
*   <PRE>
*            SHORT polyPoints[numPoints] = {x0, y0, x1, y1, x2, y2 … xn, yn};
*            Where n = numPoints - 1
*   </PRE>
*
* Input: numPoints - Defines the number of points in the polygon.
*        polyPoints - Pointer to the array of polygon points.
*
* Output: 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.
*
*
* Side Effects: none
*
*********************************************************************/
WORD    DrawPoly(SHORT numPoints, SHORT *polyPoints);

/*********************************************************************
* Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* Overview: This function draws a bar given the left, top and right,
*           bottom corners with the current color.
*
* Input: 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.
*
* Output: 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.
*
*
* Side Effects: none
*
*********************************************************************/
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom);

/*********************************************************************
* Function: void ClearDevice(void)
```

```
*
* 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().
*
* Input: none
*
* Output: none
*
* Example:
*   <CODE>
*   void ClearScreen(void)
*   {
*       SetColor(WHITE);        // set color to WHITE
*       ClearDevice();          // set screen to all WHITE
*   }
*   </CODE>
*
* Side Effects: none
*
*********************************************************************/
void    ClearDevice(void);

/*********************************************************************
* Function: WORD PutImage(SHORT left, SHORT top, void* bitmap, BYTE stretch)
*
* Overview: This function outputs image starting from left,top coordinates.
*
* Input: left - x coordinate position of the left top corner.
*        top - y coordinate position of the left top corner.
*        bitmap - pointer to the bitmap.
*        stretch - The image stretch factor.
*
* Output: 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.
*
*
* Side Effects: none
*
*********************************************************************/
WORD    PutImage(SHORT left, SHORT top, void *bitmap, BYTE stretch);

/*********************************************************************
* Function: SHORT GetImageWidth(void* bitmap)
*
* Overview: This function returns the image width.
*
* Input: bitmap - Pointer to the bitmap.
*
* Output: Returns the image width in pixels.
*
* Side Effects: none
*
*********************************************************************/
SHORT   GetImageWidth(void *bitmap);

/*********************************************************************
* Function: SHORT GetImageHeight(void* bitmap)
*
* Overview: This function returns the image height.
*
* Input: bitmap - Pointer to the bitmap.
*
* Output: Returns the image height in pixels.
*
* Side Effects: none
*
*********************************************************************/
SHORT   GetImageHeight(void *bitmap);
```

```
/*****************************************************************
* Function: WORD ExternalMemoryCallback(GFX_EXTDATA* memory, LONG offset, WORD nCount,
void* buffer)
*
* 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.
*
* Input:  memory - Pointer to the external memory bitmap or font structures
*                  (FONT_EXTERNAL or BITMAP_EXTERNAL).
*         offset - Data offset.
*         nCount - Number of bytes to be transferred into the buffer.
*         buffer - Pointer to the buffer.
*
* Output: Returns the number of bytes were transferred.
*
* Example:
*   <CODE>
*   // 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);
*  }
*  </CODE>
*
* Side Effects: none
*
*****************************************************************/
WORD    ExternalMemoryCallback(GFX_EXTDATA *memory, LONG offset, WORD nCount, void *buffer);

/*****************************************************************
* Function:  SHORT GetSineCosine(SHORT v, WORD type)
*
* PreCondition: none
*
* Input: v - the angle used to retrieve the sine or cosine value.
*            The angle must be in the range of -360 to 360 degrees.
*        type - sets if the angle calculation is for a sine or cosine
*               - GETSINE (0) - get the value of sine(v).
*               - GETCOSINE (1) - return the value of cosine(v).
*
* Output: Returns the sine or cosine of the angle given.
*
* Side Effects: none
*
* Overview: Using a lookup table, the sine or cosine values of the given angle
*           is returned.
```

```
*
* Note: none
*
*****************************************************************/
SHORT GetSineCosine(SHORT v, WORD type);

/*****************************************************************
* Function:  SHORT Sine(SHORT v)
*
* PreCondition: none
*
* Input: v - the angle used to calculate the sine value.
*            The angle must be in the range of -360 to 360 degrees.
*
* Output: Returns the sine of the given angle.
*
* Side Effects: none
*
* Overview: This calculates the sine value of the given angle.
*
* Note: none
*
*****************************************************************/
#define Sine(v)     GetSineCosine(v, GETSINE)

/*****************************************************************
* Function:  SHORT Cosine(SHORT v)
*
* PreCondition: none
*
* Input: v - the angle used to calculate the cosine value.
*            The angle must be in the range of -360 to 360 degrees.
*
* Output: Returns the cosine of the given angle.
*
* Side Effects: none
*
* Overview: This calculates the cosine value of the given angle.
*
* Note: none
*
*****************************************************************/
#define Cosine(v)       GetSineCosine(v, GETCOSINE)


/*****************************************************************
* Function:  WORD DrawArc(SHORT cx, SHORT cy, SHORT r1, SHORT r2, SHORT startAngle, SHORT
endAngle)
*
* PreCondition: none
*
* Input: cx - the location of the center of the arc in the x direction.
*        cy - the location of the center of the arc in the y direction.
*        r1 - the smaller radius of the arc.
*        r2 - the larger radius of the arc.
*        startAngle - start angle of the arc.
*        endAngle - end angle of the arc.
*
* Output: Returns 1 if the rendering is done, 0 if not yet done.
*
* Side Effects: 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 successfuly
*           and returns a 0 when it is not yet finished. The next call to the
*           function will continue the rendering.
*
* Note: none
*
*****************************************************************/
WORD DrawArc(SHORT cx, SHORT cy, SHORT r1, SHORT r2, SHORT startAngle, SHORT endAngle);
```

```
void GetCirclePoint(SHORT radius, SHORT angle, SHORT *x, SHORT *y);

#ifdef USE_GRADIENT

/*********************************************************************
* Function:  BarGradient(short left, short top, short right, short bottom, GFX_COLOR
color1, GFX_COLOR color2, DWORD length, BYTE direction);
*
* PreCondition: none
*
* Input: 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.
*        color1 - start color for the gradient
*        color2 - end color for the gradient
*        length - From 0-100%. How much of a gradient is wanted
*        direction - Gradient Direction
* Output: Returns 1 if the rendering is done, 0 if not yet done.
*
* Side Effects: none
*
* Overview: This renders a bar onto the screen, but instead of one color a gradient is drawn
* depending on the direction, length, and colors chosen
*
* Note: none
*
*********************************************************************/
void        BarGradient(short left, short top, short right, short bottom, GFX_COLOR color1,
GFX_COLOR color2, DWORD length, BYTE direction);

/*********************************************************************
* Function:  BevelGradient(short x1, short y1, short x2, short y2,SHORT rad, GFX_COLOR
color1, GFX_COLOR color2, DWORD length, BYTE direction);
*
* PreCondition: none
*
* Input: x1 - x coordinate position of the upper left center of the circle that
*             draws the rounded corners.
*        y1 - y coordinate position of the upper left center of the circle that
*             draws the rounded corners.
*        x2 - x coordinate position of the lower right center of the circle that
*             draws the rounded corners.
*        y2 - y coordinate position of the lower right center of the circle that
*             draws the rounded corners.
*        rad - defines the redius of the circle, that draws the rounded corners.
*        color1 - start color for the gradient
*        color2 - end color for the gradient
*        length - From 0-100%. How much of a gradient is wanted
*        direction - Gradient Direction
*
* Output: none
*
* Side Effects: none
*
* Overview: This renders a gradient on the screen. It works the same as the fillbevel
function,
* except a gradient out of color1 and color2 is drawn depending on the direction.
*
* Note: none
*
*********************************************************************/
WORD        BevelGradient(short x1, short y1, short x2, short y2,SHORT rad, GFX_COLOR
color1, GFX_COLOR color2, DWORD length, BYTE direction);
/*********************************************************************
* Overview: Enumeration for gradient type
*********************************************************************/
typedef enum
{
       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

} GFX_GRADIENT_TYPE;

/*********************************************************************
* Overview: This structure is used to describe the gradient style.
*
*********************************************************************/
typedef struct
{

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

} GFX_GRADIENT_STYLE;
#endif

#ifdef USE_ALPHABLEND
extern SHORT _GFXForegroundPage;        // foreground page (or buffer) used in alpha
blending
extern SHORT _GFXBackgroundPage;        // background page (or buffer)  used in alpha
blending
extern SHORT _GFXDestinationPage;       // destination page (or buffer)  used in alpha
blending


/*********************************************************************
* Function: void AlphaBlendWindow(DWORD foregroundWindowAddr, DWORD backgroundWindowAddr,
*                      DWORD destinationWindowAddr,
*                      WORD  width, WORD height,
*                      BYTE  alphaPercentage)
* PreCondition: none
*
* Input:  foregroundWindowAddr -  the starting address of the foreground window
*       backgroundWindowAddr -  the starting address of the background window
*       destinationWindowAddr - the starting address of the destination window
*       width - the width of the alpha blend window
*         height - the height of the alpha blend window
*         alphaPercentage - the amount of transparency to give the foreground Window
*
*
* Output: none
*
* Side Effects: none
*
* Overview: This alphablends a foreground and a background stored in frames to a
destination window. The function
* uses windows insides frames. Each window shares the same width and height parameters.
*
* Note: none
*********************************************************************/
extern void AlphaBlendWindow(DWORD foregroundWindowAddr, DWORD backgroundWindowAddr,
                      DWORD destinationWindowAddr,
                      WORD  width, WORD height,
                      BYTE  alphaPercentage);

/*********************************************************************
* Function: DWORD GFXGetPageXYAddress(SHORT pageNumber, WORD x, WORD y)
* PreCondition: none
*
* Input:  pageNumber - the page number
*         x - the x (horizontal) offset from 0,0 of the pagenumber
*         y - the y (vertical) offset from the 0,0 of the pagenumber
*
```

```
* Output: The address of an XY position of a certain page in memory
*
* Side Effects: none
*
* Overview: This function calculates the address of a certain x,y location in
* memory
*
* Note: none
*********************************************************************/
extern DWORD GFXGetPageXYAddress(SHORT pageNumber, WORD x, WORD y);

/*********************************************************************
* Function: DWORD GFXGetPageOriginAddress(SHORT pageNumber)
* PreCondition: none
*
* Input:  pageNumber - the page number
*
* Output: The address of the start of a certain page in memory
*
* Side Effects: none
*
* Overview: This function calculates the address of a certain 0,0 location of a
* page in memory
*
* Note: none
*********************************************************************/
extern DWORD GFXGetPageOriginAddress(SHORT pageNumber);

#endif


#endif // _PRIMITIVE_H
```

14.3 Device Driver Layer

Files

Name	Description
drvTFT001.c (see page 1031)	This is file drvTFT001.c.
drvTFT001.h (see page 1057)	This is file drvTFT001.h.
HIT1270.c (see page 1060)	This is file HIT1270.c.
HIT1270.h (see page 1075)	This is file HIT1270.h.
HX8347.c (see page 1077)	This is file HX8347.c.
HX8347.h (see page 1095)	This is file HX8347.h.
SH1101A_SSD1303.c (see page 1096)	This is file SH1101A_SSD1303.c.
SH1101A_SSD1303.h (see page 1104)	This is file SH1101A_SSD1303.h.
SSD1339.c (see page 1106)	This is file SSD1339.c.
SSD1339.h (see page 1122)	This is file SSD1339.h.
SSD1926.c (see page 1124)	This is file SSD1926.c.
SSD1926.h (see page 1168)	This is file SSD1926.h.
TCON_HX8238.c (see page 1187)	This is file TCON_HX8238.c.
TCON_SSD1289.c (see page 1192)	This is file TCON_SSD1289.c.
TCON_HX8257.c (see page 1198)	This is file TCON_HX8257.c.
CustomDisplayDriver.c (see page 1202)	This is file CustomDisplayDriver.c.
UC1610.c (see page 1204)	This is file UC1610.c.
UC1610.h (see page 1216)	This is file UC1610.h.
TCON_Custom.c (see page 1218)	This is file TCON_Custom.c.

Description

Lists all Device Driver files currently included in the library that can be used in the Device Driver Layer.

14.3.1 drvTFT001.c

This is file drvTFT001.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  LCD controller driver
 *  LG LGDP4531
 *  Renesas R61505
 *  Renesas R61580
 *  Samsung S6D0129
 *  Samsung S6D0139
 *  Orise Tech. SPFD5408
 *  Ilitek ILI9320
 *****************************************************************************
 * FileName:        drvTFT001.c
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 01/30/08     combined version for landscape and portrait
 * 01/30/08     PIC32 support
 * 06/25/09     dsPIC & PIC24H support
 * 06/26/09     16-bit PMP support
 * 03/11/11     - Modified dependencies
 *              - changes for Graphics Library Version 3.00
 *****************************************************************************/
//#include "Graphics/Graphics.h"

#include "HardwareProfile.h"

#if defined (GFX_USE_DISPLAY_CONTROLLER_S6D0129)  || defined
(GFX_USE_DISPLAY_CONTROLLER_S6D0139) || \
    defined (GFX_USE_DISPLAY_CONTROLLER_LGDP4531) || defined
```

```
(GFX_USE_DISPLAY_CONTROLLER_R61505)  || \
    defined (GFX_USE_DISPLAY_CONTROLLER_SPFD5408) || defined
(GFX_USE_DISPLAY_CONTROLLER_ILI9320) || \
    defined (GFX_USE_DISPLAY_CONTROLLER_R61580)

#include "Compiler.h"
#include "TimeDelay.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/drvTFT001.h"
#include "Graphics/Primitive.h"

#ifdef USE_GFX_PMP
    #include "Graphics/gfxpmp.h"
#elif USE_GFX_EPMP
    #include "Graphics/gfxepmp.h"
#endif

// Unsupported Graphics Library Features
#ifdef USE_TRANSPARENT_COLOR
    #warning "This driver does not support the transparent feature on PutImage(). Build
will use the PutImage() functions defind in the Primitive.c"
#endif

// Color
GFX_COLOR   _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT       _clipRgn;

// Clipping region borders
SHORT       _clipLeft;
SHORT       _clipTop;
SHORT       _clipRight;
SHORT       _clipBottom;

////////////////////// LOCAL FUNCTIONS PROTOTYPES ///////////////////////////
#ifndef USE_TRANSPARENT_COLOR
void        PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);

void        PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
#endif

/*********************************************************************
* Macro:  WritePixel(color)
*
* PreCondition: none
*
* Input: color
*
* Output: none
*
* Side Effects: none
*
* Overview: writes pixel at the current address
*
* Note: chip select should be enabled
*
*********************************************************************/
#ifdef USE_16BIT_PMP
#define WritePixel(color)   DeviceWrite(color)
#else
#define WritePixel(color)   { DeviceWrite(((WORD_VAL)color).v[1]);
```

```
DeviceWrite(((WORD_VAL)color).v[0]);}
#endif

/*********************************************************************
* Macros:  SetAddress(addr)
*
* Overview: Writes address pointer.
*
* PreCondition: none
*
* Input: add0 -  32-bit address.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
#ifdef USE_16BIT_PMP
#define SetAddress(addr)                    \
    {                                       \
    DisplaySetCommand();                    \
    DeviceWrite(0x0020);                    \
    DisplaySetData();                       \
    DeviceWrite((WORD) addr & 0x00ff);      \
    DisplaySetCommand();                    \
    DeviceWrite(0x0021);                    \
    DisplaySetData();                       \
    DeviceWrite((WORD) ((DWORD) addr >> 8));\
    DisplaySetCommand();                    \
    DeviceWrite(0x0022);                    \
    DisplaySetData();                       \
    }
#else
#define SetAddress(addr)                    \
    {\
    DisplaySetCommand();                    \
    DeviceWrite(0);                         \
    DeviceWrite(0x20);                      \
    DisplaySetData();                       \
    DeviceWrite(0);                         \
    DeviceWrite(((DWORD_VAL) (DWORD) addr).v[0]);\
    DisplaySetCommand();                    \
    DeviceWrite(0);                         \
    DeviceWrite(0x21);                      \
    DisplaySetData();                       \
    DeviceWrite(((DWORD_VAL) (DWORD) addr).v[2]);\
    DeviceWrite(((DWORD_VAL) (DWORD) addr).v[1]);\
    DisplaySetCommand();                    \
    DeviceWrite(0);                         \
    DeviceWrite(0x22);                      \
    DisplaySetData();                       \
    }
#endif

/*********************************************************************
* Function:  void  SetReg(WORD index, WORD value)
*
* PreCondition: none
*
* Input: index - register number
*        value - value to be set
*
* Output: none
*
* Side Effects: none
*
* Overview: sets graphics controller register
*
* Note: none
*
*********************************************************************/
void SetReg(WORD index, WORD value)
```

```
{
#ifdef USE_16BIT_PMP
    DisplayEnable();
    DisplaySetCommand();
    DeviceWrite(index);
    DisplaySetData();
    DeviceWrite(value);
    DisplayDisable();
#else
    DisplayEnable();
    DisplaySetCommand();
    DeviceWrite(((WORD_VAL)index).v[1]);
    DeviceWrite(((WORD_VAL)index).v[0]);
    DisplaySetData();
    DeviceWrite(((WORD_VAL)value).v[1]);
    DeviceWrite(((WORD_VAL)value).v[0]);
    DisplayDisable();
#endif
}

/*********************************************************************
* Function:  void DisplayBrightness(WORD level)
*
* PreCondition: none
*
* Input: level - Brightness level. Valid values are 0 to 100.
*           - 0: brightness level is zero or display is turned off
*           - 1: brightness level is maximum
*
* Output: none
*
* Side Effects: none
*
* Overview: Sets the brightness of the display.
*
* Note: none
*
*********************************************************************/
void DisplayBrightness(WORD level)
{
    // If the brightness can be controlled (for example through PWM)
    // add code that will control the PWM here.

    if (level > 0)
    {
        DisplayBacklightOn();
    }
    else if (level == 0)
    {
        DisplayBacklightOff();
    }

}

/*********************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*********************************************************************/
void ResetDevice(void)
{
```

```
    // Set FLASH CS pin as output
    DisplayFlashConfig();

    // Initialize the device
    DriverInterfaceInit();

    DelayMs(5);

    // Power on LCD
    DisplayPowerOn();
    DisplayPowerConfig();

    DelayMs(2);

    #if defined (GFX_USE_DISPLAY_CONTROLLER_LGDP4531)

    ////////////////////////////////////////////////////////
    // Synchronization after reset
    DisplayEnable();
    DeviceWrite(0);
    DeviceWrite(0);
    DisplayDisable();

    // Setup display
    SetReg(0x0010, 0x0628);
    SetReg(0x0012, 0x0006);
    SetReg(0x0013, 0x0A32);
    SetReg(0x0011, 0x0040);
    SetReg(0x0015, 0x0050);
    SetReg(0x0012, 0x0016);
    DelayMs(15);
    SetReg(0x0010, 0x5660);
    DelayMs(15);
    SetReg(0x0013, 0x2A4E);
    SetReg(0x0001, 0x0100);
    SetReg(0x0002, 0x0300);

        #if (DISP_ORIENTATION == 0)
    SetReg(0x0003, 0x1030);
        #else
    SetReg(0x0003, 0x1038);
        #endif
    SetReg(0x0008, 0x0202);
    SetReg(0x000A, 0x0000);
    SetReg(0x0030, 0x0000);
    SetReg(0x0031, 0x0402);
    SetReg(0x0032, 0x0106);
    SetReg(0x0033, 0x0700);
    SetReg(0x0034, 0x0104);
    SetReg(0x0035, 0x0301);
    SetReg(0x0036, 0x0707);
    SetReg(0x0037, 0x0305);
    SetReg(0x0038, 0x0208);
    SetReg(0x0039, 0x0F0B);
    DelayMs(15);
    SetReg(0x0041, 0x0002);
    SetReg(0x0060, 0x2700);
    SetReg(0x0061, 0x0001);
    SetReg(0x0090, 0x0119);
    SetReg(0x0092, 0x010A);
    SetReg(0x0093, 0x0004);
    SetReg(0x00A0, 0x0100);
    SetReg(0x0007, 0x0001);
    DelayMs(15);
    SetReg(0x0007, 0x0021);
    DelayMs(15);
    SetReg(0x0007, 0x0023);
    DelayMs(15);
    SetReg(0x0007, 0x0033);
    DelayMs(15);
    SetReg(0x0007, 0x0133);
    DelayMs(15);
```

```
    SetReg(0x00A0, 0x0000);
    DelayMs(20);

    ////////////////////////////////////////////////////////
    #elif defined (GFX_USE_DISPLAY_CONTROLLER_R61505)

    // Setup display
    SetReg(0x0000, 0x0000);
    SetReg(0x0007, 0x0001);
    DelayMs(5);
    SetReg(0x0017, 0x0001);
    DelayMs(5);
    SetReg(0x0010, 0x17b0);
    SetReg(0x0011, 0x0007);
    SetReg(0x0012, 0x011a);
    SetReg(0x0013, 0x0f00);
    SetReg(0x0015, 0x0000);
    SetReg(0x0029, 0x0009);
    SetReg(0x00fd, 0x0000);
    DelayMs(5);
    SetReg(0x0012, 0x013a);
    DelayMs(50);
    SetReg(0x0001, 0x0100);
    SetReg(0x0002, 0x0700);

        #if (DISP_ORIENTATION == 0)
    SetReg(0x0003, 0x1030);
        #else
    SetReg(0x0003, 0x1038);
        #endif
    SetReg(0x0008, 0x0808);
    SetReg(0x0009, 0x0000);
    SetReg(0x000a, 0x0000);
    SetReg(0x000c, 0x0000);
    SetReg(0x000d, 0x0000);
    SetReg(0x0030, 0x0000);
    SetReg(0x0031, 0x0000);
    SetReg(0x0032, 0x0000);
    SetReg(0x0033, 0x0000);
    SetReg(0x0034, 0x0000);
    SetReg(0x0035, 0x0000);
    SetReg(0x0036, 0x0000);
    SetReg(0x0037, 0x0707);
    SetReg(0x0038, 0x0707);
    SetReg(0x0039, 0x0707);
    SetReg(0x003a, 0x0303);
    SetReg(0x003b, 0x0303);
    SetReg(0x003c, 0x0707);
    SetReg(0x003d, 0x0808);
    SetReg(0x0050, 0x0000);
    SetReg(0x0051, 0x00ef);
    SetReg(0x0052, 0x0000);
    SetReg(0x0053, 0x013f);
    SetReg(0x0060, 0x2700);
    SetReg(0x0061, 0x0001);
    SetReg(0x006a, 0x0000);
    SetReg(0x0090, 0x0010);
    SetReg(0x0092, 0x0000);
    SetReg(0x0093, 0x0000);
    SetReg(0x0007, 0x0021);
    DelayMs(1);
    SetReg(0x0007, 0x0061);
    DelayMs(50);
    SetReg(0x0007, 0x0173);
    SetReg(0x0020, 0x0000);
    SetReg(0x0021, 0x0000);
    SetReg(0x0022, 0x0000);
    SetReg(0x0030, 0x0707);
    SetReg(0x0031, 0x0407);
    SetReg(0x0032, 0x0203);
    SetReg(0x0033, 0x0303);
    SetReg(0x0034, 0x0303);
```

```
    SetReg(0x0035, 0x0202);
    SetReg(0x0036, 0x001f);
    SetReg(0x0037, 0x0707);
    SetReg(0x0038, 0x0407);
    SetReg(0x0039, 0x0203);
    SetReg(0x003a, 0x0303);
    SetReg(0x003b, 0x0303);
    SetReg(0x003c, 0x0202);
    SetReg(0x003d, 0x001f);
    SetReg(0x0020, 0x0000);
    SetReg(0x0021, 0x0000);

    ////////////////////////////////////////////////////////
    #elif defined (GFX_USE_DISPLAY_CONTROLLER_S6D0129) || defined
(GFX_USE_DISPLAY_CONTROLLER_S6D0139)

    // Setup display
    SetReg(0x0000, 0x0001);
    SetReg(0x0011, 0x1a00);
    SetReg(0x0014, 0x2020);
    SetReg(0x0010, 0x0900);
    SetReg(0x0013, 0x0040);
    SetReg(0x0013, 0x0060);
    SetReg(0x0013, 0x0070);
    SetReg(0x0011, 0x1a04);
    SetReg(0x0010, 0x2f00);
    SetReg(0x0001, 0x0127);
    SetReg(0x0002, 0x0700);

        #if (DISP_ORIENTATION == 0)
    SetReg(0x0003, 0x1030);
        #else
    SetReg(0x0003, 0x1038);
        #endif
    SetReg(0x0007, 0x0000);
    SetReg(0x0008, 0x0808);
    SetReg(0x0009, 0x0000);
    SetReg(0x000b, 0x0000);
    SetReg(0x000c, 0x0000);
    SetReg(0x0040, 0x0000);
    SetReg(0x0041, 0x0000);
    SetReg(0x0042, 0x013f);
    SetReg(0x0043, 0x0000);
    SetReg(0x0044, 0x00ef);
    SetReg(0x0045, 0x0000);
    SetReg(0x0046, 0xef00);
    SetReg(0x0047, 0x013f);
    SetReg(0x0048, 0x0000);
    SetReg(0x0007, 0x0014);
    SetReg(0x0007, 0x0016);
    SetReg(0x0007, 0x0017);
    SetReg(0x0020, 0x0000);
    SetReg(0x0021, 0x0000);
    SetReg(0x0022, 0x0000);

    ////////////////////////////////////////////////////////
    #elif defined (GFX_USE_DISPLAY_CONTROLLER_SPFD5408)

    // Setup display
    SetReg(0x0000, 0x0000);
    SetReg(0x0001, 0x0000);
    SetReg(0x0002, 0x0700);

        #if (DISP_ORIENTATION == 0)
    SetReg(0x0003, 0x1010);
        #else
    SetReg(0x0003, 0x1028);
        #endif
    SetReg(0x0004, 0x0000);
    SetReg(0x0008, 0x0207);
    SetReg(0x0009, 0x0000);
    SetReg(0x000a, 0x0000);
```

```
    SetReg(0x000c, 0x0000);
    SetReg(0x000d, 0x0000);
    SetReg(0x000f, 0x0000);
    SetReg(0x0007, 0x0101);
    SetReg(0x0010, 0x12b0);
    SetReg(0x0011, 0x0007);
    SetReg(0x0017, 0x0001);
    SetReg(0x0012, 0x01bb);
    SetReg(0x0013, 0x1300);
    SetReg(0x0029, 0x0010);

    SetReg(0x0030, 0x0100);
    SetReg(0x0031, 0x0c19);
    SetReg(0x0032, 0x111e);
    SetReg(0x0033, 0x3819);
    SetReg(0x0034, 0x350b);
    SetReg(0x0035, 0x0e08);
    SetReg(0x0036, 0x0d07);
    SetReg(0x0037, 0x0318);
    SetReg(0x0038, 0x0705);
    SetReg(0x0039, 0x0303);
    SetReg(0x003a, 0x0905);
    SetReg(0x003b, 0x0801);
    SetReg(0x003c, 0x030e);
    SetReg(0x003d, 0x050d);
    SetReg(0x003e, 0x0106);
    SetReg(0x003f, 0x0408);

    SetReg(0x0050, 0x0000);
    SetReg(0x0051, 0x00ef);
    SetReg(0x0052, 0x0000);
    SetReg(0x0053, 0x013f);
    SetReg(0x0060, 0xa700);
    SetReg(0x0061, 0x0001);
    SetReg(0x006a, 0x0000);
    SetReg(0x0080, 0x0000);
    SetReg(0x0081, 0x0000);
    SetReg(0x0082, 0x0000);
    SetReg(0x0083, 0x0000);
    SetReg(0x0084, 0x0000);
    SetReg(0x0085, 0x0000);
    SetReg(0x0090, 0x0010);
    SetReg(0x0092, 0x0000);
    SetReg(0x0093, 0x0103);
    SetReg(0x0095, 0x0110);
    SetReg(0x0097, 0x0000);
    SetReg(0x0098, 0x0000);
    SetReg(0x00f0, 0x5408);
    SetReg(0x00f3, 0x0010);
    SetReg(0x00f4, 0x001f);
    SetReg(0x00f0, 0x0000);
    SetReg(0x0007, 0x0133);

    ////////////////////////////////////////////////////////
    #elif defined (GFX_USE_DISPLAY_CONTROLLER_ILI9320)
    SetReg(0x0000, 0x0001); //start Int. osc
    DelayMs(15);
    SetReg(0x0001, 0x0100); //Set SS bit (shift direction of outputs is from S720 to S1)
    SetReg(0x0002, 0x0700); //select  the line inversion
        #if (DISP_ORIENTATION == 0)
    SetReg(0x0003, 0x1030); //Entry mode(Horizontal : increment,Vertical : increment, AM=0)
        #else
    SetReg(0x0003, 0x1038); //Entry mode(Horizontal : increment,Vertical : increment, AM=1)
        #endif
    SetReg(0x0004, 0x0000); //Resize control(No resizing)
    SetReg(0x0008, 0x0202); //front and back porch 2 lines
    SetReg(0x0009, 0x0000); //select normal scan
    SetReg(0x000A, 0x0000); //display control 4
    SetReg(0x000C, 0x0000); //system interface(2 transfer /pixel), internal sys clock,
    SetReg(0x000D, 0x0000); //Frame marker position
    SetReg(0x000F, 0x0000); //selects clk, enable and sync signal polarity,
    SetReg(0x0010, 0x0000); //
```

```
    SetReg(0x0011, 0x0000); //power control 2 reference voltages = 1:1,
    SetReg(0x0012, 0x0000); //power control 3 VRH
    SetReg(0x0013, 0x0000); //power control 4 VCOM amplitude
    DelayMs(20);
    SetReg(0x0010, 0x17B0); //power control 1 BT,AP
    SetReg(0x0011, 0x0137); //power control 2 DC,VC
    DelayMs(50);
    SetReg(0x0012, 0x0139); //power control 3 VRH
    DelayMs(50);
    SetReg(0x0013, 0x1d00); //power control 4 vcom amplitude
    SetReg(0x0029, 0x0011); //power control 7 VCOMH
    DelayMs(50);
    SetReg(0x0030, 0x0007);
    SetReg(0x0031, 0x0403);
    SetReg(0x0032, 0x0404);
    SetReg(0x0035, 0x0002);
    SetReg(0x0036, 0x0707);
    SetReg(0x0037, 0x0606);
    SetReg(0x0038, 0x0106);
    SetReg(0x0039, 0x0007);
    SetReg(0x003c, 0x0700);
    SetReg(0x003d, 0x0707);
    SetReg(0x0020, 0x0000); //starting Horizontal GRAM Address
    SetReg(0x0021, 0x0000); //starting Vertical GRAM Address
    SetReg(0x0050, 0x0000); //Horizontal GRAM Start Position
    SetReg(0x0051, 0x00EF); //Horizontal GRAM end Position
    SetReg(0x0052, 0x0000); //Vertical GRAM Start Position
    SetReg(0x0053, 0x013F); //Vertical GRAM end Position
    SetReg(0x0060, 0x2700); //starts scanning from G1, and 320 drive lines
    SetReg(0x0061, 0x0001); //fixed base display
    SetReg(0x006a, 0x0000); //no scroll
    SetReg(0x0090, 0x0010); //set Clocks/Line =16, Internal Operation Clock
Frequency=fosc/1,
    SetReg(0x0092, 0x0000); //set gate output non-overlap period=0
    SetReg(0x0093, 0x0003); //set Source Output Position=3
    SetReg(0x0095, 0x0110); //RGB interface(Clocks per line period=16 clocks)
    SetReg(0x0097, 0x0110); //set Gate Non-overlap Period 0 locksc
    SetReg(0x0098, 0x0110); //
    SetReg(0x0007, 0x0173); //display On
    ////////////////////////////////////////////////////////
    #elif defined (GFX_USE_DISPLAY_CONTROLLER_R61580)

    // Synchronization after reset
    DelayMs(2);
    SetReg(0x0000, 0x0000);
    SetReg(0x0000, 0x0000);
    SetReg(0x0000, 0x0000);
    SetReg(0x0000, 0x0000);

    // Setup display
    SetReg(0x00A4, 0x0001); // CALB=1
    DelayMs(2);
    SetReg(0x0060, 0xA700); // Driver Output Control
    SetReg(0x0008, 0x0808); // Display Control BP=8, FP=8
    SetReg(0x0030, 0x0111); // y control
    SetReg(0x0031, 0x2410); // y control
    SetReg(0x0032, 0x0501); // y control
    SetReg(0x0033, 0x050C); // y control
    SetReg(0x0034, 0x2211); // y control
    SetReg(0x0035, 0x0C05); // y control
    SetReg(0x0036, 0x2105); // y control
    SetReg(0x0037, 0x1004); // y control
    SetReg(0x0038, 0x1101); // y control
    SetReg(0x0039, 0x1122); // y control
    SetReg(0x0090, 0x0019); // 80Hz
    SetReg(0x0010, 0x0530); // Power Control
    SetReg(0x0011, 0x0237);
    SetReg(0x0012, 0x01BF);
    SetReg(0x0013, 0x1300);
    DelayMs(100);

    SetReg(0x0001, 0x0100);
```

```
    SetReg(0x0002, 0x0200);
        #if (DISP_ORIENTATION == 0)
    SetReg(0x0003, 0x1030);
        #else
    SetReg(0x0003, 0x1038);
        #endif
    SetReg(0x0009, 0x0001);
    SetReg(0x000A, 0x0008);
    SetReg(0x000C, 0x0001);
    SetReg(0x000D, 0xD000);
    SetReg(0x000E, 0x0030);
    SetReg(0x000F, 0x0000);
    SetReg(0x0020, 0x0000);
    SetReg(0x0021, 0x0000);
    SetReg(0x0029, 0x0077);
    SetReg(0x0050, 0x0000);
    SetReg(0x0051, 0xD0EF);
    SetReg(0x0052, 0x0000);
    SetReg(0x0053, 0x013F);
    SetReg(0x0061, 0x0001);
    SetReg(0x006A, 0x0000);
    SetReg(0x0080, 0x0000);
    SetReg(0x0081, 0x0000);
    SetReg(0x0082, 0x005F);
    SetReg(0x0093, 0x0701);
    SetReg(0x0007, 0x0100);
    SetReg(0x0022, 0x0000);
    #else
        #error Graphics controller is not supported.
    #endif
    DelayMs(20);
}

#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel
*
* Note: none
*
*********************************************************************/
```

```
void PutPixel(SHORT x, SHORT y)
{
    DWORD   address;
    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    #if (DISP_ORIENTATION == 0)
    address = (long)LINE_MEM_PITCH * y + x;

    #else
    y = GetMaxY() - y;
    address = (long)LINE_MEM_PITCH * x + y;
    #endif
    DisplayEnable();
    SetAddress(address);
    WritePixel(_color);
    DisplayDisable();
}

/*********************************************************************
* Function: WORD GetPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: pixel color
*
* Side Effects: none
*
* Overview: returns pixel color at x,y position
*
* Note: none
*
*********************************************************************/
#ifndef __PIC24FJ256GB210__
#ifdef USE_16BIT_PMP

/* */
WORD GetPixel(SHORT x, SHORT y)
{
    DWORD   address;
    WORD    result;
        #if (DISP_ORIENTATION == 0)
    address = (long)LINE_MEM_PITCH * y + x;
        #else
    y = GetMaxY() - y;
    address = (long)LINE_MEM_PITCH * x + y;
        #endif
    DisplayEnable();

    SetAddress(address);

    // Temporary change wait cycles for reading (250ns = 4 cycles)
        #if defined(__C30__)
    PMMODEbits.WAITM = 4;
        #elif defined(__PIC32MX__)
    PMMODEbits.WAITM = 8;
        #else
            #error Need wait states for the device
        #endif
    DisplaySetData();
```

```
    // First RD cycle to move data from GRAM to Read Data Latch
    result = PMDIN1;

    while(PMMODEbits.BUSY);

    // Second RD cycle to get data from Read Data Latch
    result = PMDIN1;

    while(PMMODEbits.BUSY);

    // Disable LCD
    DisplayDisable();

    // Disable PMP
    PMCONbits.PMPEN = 1;

    // Read result
    result = PMDIN1;

    // Restore wait cycles for writing (60ns)
        #if defined(__dsPIC33F__) || defined(__PIC24H__)
    PMMODEbits.WAITM = 2;
        #else
    PMMODEbits.WAITM = 1;
        #endif

    // Enable PMP
    PMCONbits.PMPEN = 1;

    return (result);
}

#else

/* */
WORD GetPixel(SHORT x, SHORT y)
{
    DWORD       address;
    WORD_VAL    result;

        #if (DISP_ORIENTATION == 0)
    address = (long)LINE_MEM_PITCH * y + x;
        #else
    y = GetMaxY() - y;
    address = (long)LINE_MEM_PITCH * x + y;
        #endif
    DisplayEnable();

    SetAddress(address);

    // Temporary change wait cycles for reading (250ns = 4 cycles)
        #if defined(__C30__)
    PMMODEbits.WAITM = 4;
        #elif defined(__PIC32MX__)
    PMMODEbits.WAITM = 8;
        #else
            #error Need wait states for the device
        #endif
    DisplaySetData();

    // First RD cycle to move data from GRAM to Read Data Latch
    result.v[1] = PMDIN1;

    while(PMMODEbits.BUSY);

        #if defined(GFX_USE_DISPLAY_CONTROLLER_ILI9320)
    DelayForSync();
        #endif

    // Second RD cycle to move data from GRAM to Read Data Latch
    result.v[1] = PMDIN1;
```

```
    while(PMMODEbits.BUSY);
        #if defined (GFX_USE_DISPLAY_CONTROLLER_ILI9320)
    DelayForSync();
        #endif

    // First RD cycle to get data from Read Data Latch
    // Read previous dummy value
    result.v[1] = PMDIN1;

    while(PMMODEbits.BUSY);
        #if defined (GFX_USE_DISPLAY_CONTROLLER_ILI9320)
    DelayForSync();
        #endif

    // Second RD cycle to get data from Read Data Latch
    // Read MSB
    result.v[1] = PMDIN1;

    while(PMMODEbits.BUSY);
        #if defined (GFX_USE_DISPLAY_CONTROLLER_ILI9320)
    DelayForSync();
        #endif

    // Disable LCD
    DisplayDisable();

    // Disable PMP
    PMCONbits.PMPEN = 1;

    // Read LSB
    result.v[0] = PMDIN1;
        #if defined (GFX_USE_DISPLAY_CONTROLLER_ILI9320)
    DelayForSync();
        #endif

    // Restore wait cycles for writing (60ns)
        #if defined(__dsPIC33F__) || defined(__PIC24H__)
    PMMODEbits.WAITM = 2;
        #else
    PMMODEbits.WAITM = 1;
        #endif

    // Enable PMP
    PMCONbits.PMPEN = 1;

    return (result.Val);
}

#endif
#endif

/*****************************************************************************
* Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* PreCondition: none
*
* Input: left,top - top left corner coordinates,
*        right,bottom - bottom right corner coordinates
*
* Output: 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.
*
* Side Effects: none
*
* Overview: draws rectangle filled with current color
*
* Note: none
*
*****************************************************************************/
```

```
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    DWORD           address;
    register SHORT  x, y;

    #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
    #else
    if(IsDeviceBusy() != 0)
        return (0);
    #endif
    if(_clipRgn)
    {
        if(left < _clipLeft)
            left = _clipLeft;
        if(right > _clipRight)
            right = _clipRight;
        if(top < _clipTop)
            top = _clipTop;
        if(bottom > _clipBottom)
            bottom = _clipBottom;
    }

    #if (DISP_ORIENTATION == 0)
    address = (DWORD) LINE_MEM_PITCH * top + left;

    DisplayEnable();
    for(y = top; y < bottom + 1; y++)
    {
        SetAddress(address);
        for(x = left; x < right + 1; x++)
        {
            WritePixel(_color);
        }

        address += LINE_MEM_PITCH;
    }

    DisplayDisable();

    #else
    top = GetMaxY() - top;
    bottom = GetMaxY() - bottom;
    address = (DWORD) LINE_MEM_PITCH * left + top;

    DisplayEnable();
    for(y = bottom; y < top + 1; y++)
    {
        SetAddress(address);
        for(x = left; x < right + 1; x++)
        {
            WritePixel(_color);
        }

        address -= 1;
    }

    DisplayDisable();
    #endif
    return (1);
}

/*********************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
```

```
*
* Side Effects: none
*
* Overview: clears screen with current color
*
* Note: none
*
*********************************************************************/
void ClearDevice(void)
{
    DWORD   counter;

    DisplayEnable();
    SetAddress(0);
    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1); counter++)
    {
        WritePixel(_color);
    }

    DisplayDisable();
}

#ifndef USE_TRANSPARENT_COLOR

    #ifdef USE_BITMAP_FLASH

/*********************************************************************
* Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner,
*        image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp = 0;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;
    WORD                pallete[2];
    BYTE                mask;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
```

```
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[0] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[1] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
                address -= 1;
            #endif
        }
    }

    DisplayDisable();
}

/*****************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 16 color image starting from left,top coordinates
```

```
 *
 * Note: image must be located in flash
 *
 ***********************************************************************/
void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    BYTE                temp = 0;
    register BYTE       stretchX, stretchY;
    WORD                pallete[16];
    WORD                counter;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(pallete[temp >> 4]);
                }
                else
                {
                    temp = *flashAddress;
                    flashAddress++;

                    // first pixel in byte
                    SetColor(pallete[temp & 0x0f]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
```

```
                {
                    WritePixel(_color);
                }

                // Shift to the next pixel
                //temp >>= 4;
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
                address -= 1;
            #endif
        }
    }

    DisplayDisable();
}

/*****************************************************************************
* Function: void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 256 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                temp;
    BYTE                stretchX, stretchY;
    WORD                pallete[256];
    WORD                counter;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
            #if (DISP_ORIENTATION == 0)
    address = (long)LINE_MEM_PITCH * top + left;
            #else
    address = (long)LINE_MEM_PITCH * left + top;
            #endif

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
```

```
        flashAddress += 2;
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

                    #if (DISP_ORIENTATION == 0)
            address += LINE_MEM_PITCH;
                    #else
            address -= 1;
                    #endif
        }
    }

    DisplayDisable();
}

/*****************************************************************************
* Function: void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs hicolor image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_WORD *flashAddress;
    register FLASH_WORD *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                temp;
    register BYTE       stretchX, stretchY;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Move pointer to size information
    flashAddress = (FLASH_WORD *)image + 1;
```

```
    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
                address -= 1;
            #endif
        }
    }

    DisplayDisable();
}

    #endif // USE_BITMAP_FLASH
    #ifdef USE_BITMAP_EXTERNAL

/*********************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*********************************************************************/
void PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
```

```
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[2];
    BYTE            lineBuffer[((GetMaxX() + 1) / 8) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp = 0;
    BYTE            mask;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;


    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *pData++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
```

```
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
                address -= 1;
            #endif
        }

        DisplayDisable();
    }
}

/*****************************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************************/
void PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[16];
    BYTE            lineBuffer[((GetMaxX() + 1) / 2) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp = 0;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
```

```
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(pallete[temp >> 4]);
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    SetColor(pallete[temp & 0x0f]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
                address -= 1;
            #endif
        }

        DisplayDisable();
    }
}

/*********************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
```

```
 * Side Effects: none
 *
 * Overview: outputs monochrome image starting from left,top coordinates
 *
 * Note: image must be located in external memory
 *
 ********************************************************************/
void PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[256];
    BYTE            lineBuffer[(GetMaxX() + 1)];
    BYTE            *pData;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
        memOffset += sizeX;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
```

```
                address -= 1;
            #endif
        }

        DisplayDisable();
    }
}

/*****************************************************************************
* Function: void PutImage16BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************************/
void PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            lineBuffer[(GetMaxX() + 1)];
    WORD            *pData;
    WORD            byteWidth;

    WORD            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    #if (DISP_ORIENTATION == 90)
        top = GetMaxY() - top;
    #endif

    // Set start address
    #if (DISP_ORIENTATION == 0)
        address = (long)LINE_MEM_PITCH * top + left;
    #else
        address = (long)LINE_MEM_PITCH * left + top;
    #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    byteWidth = sizeX << 1;

    for(y = 0; y < sizeY; y++)
    {
        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);
```

```
            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            #if (DISP_ORIENTATION == 0)
                address += LINE_MEM_PITCH;
            #else
                address -= 1;
            #endif
        }

        DisplayDisable();
    }
}

    #endif //USE_BITMAP_EXTERNAL
#endif // #ifndef USE_TRANSPARENT_COLOR


/*****************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*****************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
void SetClip(BYTE control)
{
```

```
    _clipRgn=control;
}

/*****************************************************************
* Function: IsDeviceBusy()
*
* Overview: Returns non-zero if LCD controller is busy
*           (previous drawing operation is not completed).
*
* PreCondition: none
*
* Input: none
*
* Output: Busy status.
*
* Side Effects: none
*
*****************************************************************/
WORD IsDeviceBusy(void)
{
    return (0);
}

#endif // #if defined (GFX_USE_DISPLAY_CONTROLLER_S6D0129)  || defined
(GFX_USE_DISPLAY_CONTROLLER_S6D0139) ||  ...
```

14.3.2 drvTFT001.h

This is file drvTFT001.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  LCD controller driver
 *  LG LGDP4531
 *  Renesas R61505
 *  Renesas R61580
 *  Samsung S6D0129
 *  Samsung S6D0139
 *  Orise Tech. SPFD5408
 *  Ilitek ILI9320
 *****************************************************************************
 * FileName:        drvTFT001.h
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 01/29/08
 * 01/31/08     combined portrait and landscape, PIC32 support
 * 06/26/09     16-bit PMP support
 * 03/11/11     Changes for Graphics Library Version 3.00
 ***************************************************************************/
#ifndef _DRVTFT001_H
    #define _DRVTFT001_H

    #include "HardwareProfile.h"
    #include "GraphicsConfig.h"
    #include "GenericTypeDefs.h"

/*********************************************************************
* Error Checking
*********************************************************************/
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif
    #ifndef COLOR_DEPTH
        #error COLOR_DEPTH must be defined in GraphicsConfig.h
    #endif

/*********************************************************************
* Overview: Horizontal and vertical screen size.
*********************************************************************/
    #if (DISP_HOR_RESOLUTION != 240)
        #error This driver doesn't supports this resolution. Horisontal resolution must be
240 pixels.
    #endif
    #if (DISP_VER_RESOLUTION != 320)
        #error This driver doesn't supports this resolution. Vertical resolution must be
320 pixels.
    #endif

/*********************************************************************
* Overview: Display orientation.
*********************************************************************/
    #if (DISP_ORIENTATION != 0) && (DISP_ORIENTATION != 90)
        #error This driver doesn't support this orientation.
    #endif

/*********************************************************************
* Overview: Color depth.
*********************************************************************/
    #if (COLOR_DEPTH != 16)
        #error This driver support 16 BPP color depth only.
    #endif

/*********************************************************************
* Function:  void SetReg(WORD index, WORD value);
*
* PreCondition: none
*
* Input: index - register number
*        value - data to be written to the register
*
* Output: none
*
```

```
* Side Effects: none
*
* Overview: Writes the given value to the register defined by index.
*
* Note: none
*
*********************************************************************/
void            SetReg(WORD index, WORD value);

/*********************************************************************
* Function:  void DisplayBrightness(WORD level)
*
* PreCondition: none
*
* Input: level - Brightness level. Valid values are 0 to 100.
*           - 0: brightness level is zero or display is turned off
*           - 1: brightness level is maximum
*
* Output: none
*
* Side Effects: none
*
* Overview: Sets the brightness of the display.
*
* Note: none
*
*********************************************************************/
void            DisplayBrightness(WORD level);


// Memory pitch for line
    #define LINE_MEM_PITCH  0x100

/*********************************************************************
* Function:  void ResetDevice()
*
* Overview: Initializes LCD module.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void    ResetDevice(void);

/*********************************************************************
* Macros: SetPalette(colorNum, color)
*
* Overview:  Sets palette register.
*
* PreCondition: none
*
* Input: colorNum - Register number.
*        color - Color.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define SetPalette(colorNum, color)

#endif // _DRVTFT001_H
```

14.3.3 HIT1270.c

This is file HIT1270.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Densitron HIT1270 controller driver
 *  Landscape orientation
 *****************************************************************************
 * FileName:        HIT1270.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07         Version 1.0 release
 * 04/12/11         Support for Graphics Library Version 3.00
  ****************************************************************************/
#include "HardwareProfile.h"

#if defined(GFX_USE_DISPLAY_CONTROLLER_HIT1270)

#include "Compiler.h"
#include "GenericTypeDefs.h"
#include "TimeDelay.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/HIT1270.h"
#include "Graphics/Primitive.h"


#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_EPMP)
    #include "Graphics/gfxepmp.h"
#endif

// Unsupported Graphics Library Features
#ifdef USE_TRANSPARENT_COLOR
```

```
    #warning "This driver does not support the transparent feature on PutImage(). Build
will use the PutImage() functions defind in the Primitive.c"
#endif

#define SETCOLOR_LOWBYTE(color) (((WORD_VAL) color).v[0] & 0x1c)
#define SETCOLOR_HIBYTE(color)  (((WORD_VAL) color).v[1] & 0x73)

// Color
GFX_COLOR   _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT       _clipRgn;

// Clipping region borders
SHORT       _clipLeft;
SHORT       _clipTop;
SHORT       _clipRight;
SHORT       _clipBottom;

//////////////////////// LOCAL FUNCTIONS PROTOTYPES ////////////////////////////
#ifndef USE_TRANSPARENT_COLOR
void        PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);

void        PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
#endif

/*********************************************************************
* Function:  SetAddress(addr2,addr1,addr0)
*
* PreCondition: none
*
* Input: addr0,addr1,addr2 - address bytes
*
* Output: none
*
* Side Effects: none
*
* Overview: writes S6D0129 address pointer
*
* Note: none
*
*********************************************************************/
inline void SetAddress(DWORD address)
{
    DisplaySetCommand();
    DeviceWrite(SET_DATA_POINTER);
    DisplaySetData();
    DeviceWrite(((DWORD_VAL)address).v[0]);
    DeviceWrite(((DWORD_VAL)address).v[1]);
    DeviceWrite(((DWORD_VAL)address).v[2]);
}

/*********************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
```

```
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*********************************************************************/
void ResetDevice(void)
{
    WORD counter;

    // Initialize the device
    DriverInterfaceInit();

    // Enable LCD
    DisplayEnable();

    DelayMs(10);

    // Clear text layer
    SetAddress(0);
    for(counter = 0; counter < 0x4b0; counter++)
    {
        DeviceWrite(0x07);
        DeviceWrite(0x07);
    }

    // Disable LCD
    DisplayDisable();

    DelayMs(20);
}

#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel
*
* Note: none
*
*********************************************************************/
void PutPixel(SHORT x, SHORT y)
```

```
{
    DWORD   address;
    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    DisplayEnable();
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * y + x;
    SetAddress(address);
    DeviceWrite(SETCOLOR_LOWBYTE(_color));
    DeviceWrite(SETCOLOR_HIBYTE(_color));
    DisplayDisable();
}

/*********************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*********************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClip(BYTE control)
{
    _clipRgn=control;
}

/*********************************************************************
* Function: IsDeviceBusy()
*
```

```
 * Overview: Returns non-zero if LCD controller is busy
 *           (previous drawing operation is not completed).
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: Busy status.
 *
 * Side Effects: none
 *
 ********************************************************************/
WORD IsDeviceBusy(void)
{
    return (0);
}

/********************************************************************
 * Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
 *
 * PreCondition: none
 *
 * Input: left,top - top left corner coordinates,
 *        right,bottom - bottom right corner coordinates
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 * Overview: draws rectangle filled with current color
 *
 * Note: none
 *
 ********************************************************************/
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    DWORD       address;
    register SHORT  x, y;

    #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
    #else
    if(IsDeviceBusy() != 0)
        return (0);
    #endif
    if(_clipRgn)
    {
        if(left < _clipLeft)
            left = _clipLeft;
        if(right > _clipRight)
            right = _clipRight;
        if(top < _clipTop)
            top = _clipTop;
        if(bottom > _clipBottom)
            bottom = _clipBottom;
    }

    address = BUF_MEM_OFFSET + (DWORD) LINE_MEM_PITCH * top + left;

    DisplayEnable();

    for(y = top; y < bottom + 1; y++)
    {
        SetAddress(address);
        for(x = left; x < right + 1; x++)
        {
```

```
            DeviceWrite(SETCOLOR_HIBYTE(_color));
            DeviceWrite(SETCOLOR_LOWBYTE(_color));
        }

        address += LINE_MEM_PITCH;
    }

    DisplayDisable();
    return (1);
}

/*********************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: clears screen with current color
*
* Note: none
*
*********************************************************************/
void ClearDevice(void)
{
    DWORD   counter;

    DisplayEnable();
    SetAddress(0x018000);
    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1); counter++)
    {
        DeviceWrite(SETCOLOR_HIBYTE(_color));
        DeviceWrite(SETCOLOR_LOWBYTE(_color));
    }

    DisplayDisable();
    MoveTo(0, 0);
}

#ifndef USE_TRANSPARENT_COLOR

#ifdef USE_BITMAP_FLASH

/*********************************************************************
* Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    register DWORD  address;
    BYTE                temp;
    register WORD       sizeX, sizeY;
    WORD                x, y;
```

```
    register BYTE       stretchX, stretchY;
    WORD                pallete[2];
    BYTE                mask;

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[0] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[1] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            address += LINE_MEM_PITCH;
        }
    }

    DisplayDisable();
}

/*****************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
```

```
 *
 * Input: left,top - left top image corner, image - image pointer,
 *        stretch - image stretch factor
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: outputs 16 color image starting from left,top coordinates
 *
 * Note: image must be located in flash
 *
 ********************************************************************/
void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD  address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    BYTE                temp;
    register BYTE       stretchX, stretchY;
    WORD                pallete[16];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }


    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            DisplayEnable();
            flashAddress = tempFlashAddress;
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if((x & 0x0001) == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                }

                // Set color
                SetColor(pallete[temp & 0x0f]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }
```

```
                // Shift to the next pixel
                temp >>= 4;
            }

            address += LINE_MEM_PITCH;
            DisplayDisable();
        }
    }


}

/*****************************************************************************
* Function: void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 256 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD  address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    BYTE                temp;
    register BYTE       stretchX, stretchY;
    WORD                pallete[256];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {
```

```
                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));

                }
            }

            address += LINE_MEM_PITCH;
        }
    }

    DisplayDisable();
}

/*********************************************************************
* Function: void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs hicolor image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_WORD *flashAddress;
    register FLASH_WORD *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                temp;
    register BYTE       stretchX, stretchY;

    // Move pointer to size information
    flashAddress = (FLASH_WORD *)image + 1;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {
```

```
                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }
            }

            address += LINE_MEM_PITCH;
        }
    }

    DisplayDisable();
}

#endif //USE_BITMAP_FLASH
#ifdef USE_BITMAP_EXTERNAL

/*********************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD      address;
    register DWORD      memOffset;
    BITMAP_HEADER       bmp;
    WORD                pallete[2];
    BYTE                lineBuffer[320 / 8];
    BYTE                *pData;
    SHORT               byteWidth;

    BYTE                temp;
    BYTE                mask;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);

    // Line width in bytes
```

```
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *pData++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            address += LINE_MEM_PITCH;
        }
    }

    DisplayDisable();
}

/*********************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
```

```
 *
 * Note: image must be located in flash
 *
 ********************************************************************/
void PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD      address;
    register DWORD      memOffset;
    BITMAP_HEADER       bmp;
    WORD                pallete[16];
    BYTE                lineBuffer[320 / 2];
    BYTE                *pData;
    SHORT               byteWidth;

    BYTE                temp;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(pallete[temp & 0x0f]);
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    SetColor(pallete[temp >> 4]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
```

```
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }
            }

            address += LINE_MEM_PITCH;
        }

        DisplayDisable();
    }
}

/*********************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD      address;
    register DWORD      memOffset;
    BITMAP_HEADER       bmp;
    WORD                pallete[256];
    BYTE                lineBuffer[320];
    BYTE                *pData;

    BYTE                temp;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
        memOffset += sizeX;

        DisplayEnable();

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
```

```
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }
            }

            address += LINE_MEM_PITCH;
        }

        DisplayDisable();
    }
}

/*********************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD      memOffset;
    BITMAP_HEADER       bmp;
    WORD                lineBuffer[320];
    WORD                *pData;
    WORD                byteWidth;

    WORD                temp;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;

    // Set start address
    address = BUF_MEM_OFFSET + (long)LINE_MEM_PITCH * top + left;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    byteWidth = sizeX << 1;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
```

```
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;

        DisplayEnable();

        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(SETCOLOR_HIBYTE(_color));
                    DeviceWrite(SETCOLOR_LOWBYTE(_color));
                }
            }

            address += LINE_MEM_PITCH;
        }

        DisplayDisable();
    }
}

#endif //USE_BITMAP_EXTERNAL

#endif //#ifndef USE_TRANSPARENT_COLOR

#endif //GFX_USE_DISPLAY_CONTROLLER_HIT1270
```

14.3.4 HIT1270.h

This is file HIT1270.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Densitron HIT1270 LCD controller driver
 *  Landscape orientation only
 *****************************************************************************
 * FileName:        HIT1270.h
 * Dependencies:    p24Fxxxx.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 04/12/11     Support for Graphics Library Version 3.00
 ***************************************************************************/
#ifndef _HIT1270L_H
    #define _HIT1270L_H

    #include "HardwareProfile.h"
    #include "GraphicsConfig.h"
    #include "GenericTypeDefs.h"

    #ifdef USE_16BIT_PMP
        #error This driver doesn't support 16-bit PMP (USE_8BIT_PMP instead in
HardwareProfile.h)
    #endif
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef COLOR_DEPTH
        #error COLOR_DEPTH must be defined in GraphicsConfig.h
    #endif
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif

// Horizontal and vertical screen size
    #if (DISP_HOR_RESOLUTION != 320)
        #error This driver doesn't supports this resolution. Horisontal resolution must be
320 pixels.
    #endif
    #if (DISP_VER_RESOLUTION > 234)
        #error This driver doesn't supports this resolution. Vertical resolution must be
234 pixels or less.
    #endif

// Display orientation
    #if (DISP_ORIENTATION != 0)
        #error This driver doesn't support this orientation.
    #endif

// Color depth
    #if (COLOR_DEPTH != 16)
        #error This driver doesn't support this color depth. It should be 16.
    #endif

// Memory pitch for line
    #define LINE_MEM_PITCH  320

// Video buffer offset
    #define BUF_MEM_OFFSET  (DWORD) 0x18000

// HIT1270 commands
    #define SET_DATA_POINTER    0x50
    #define SRAM_WRITE          0x51
    #define STAMP_ICON          0x52
    #define FILL_BLOCK          0x53
```

```
    #define FLASH_WRITE         0x60
    #define FLASH_READ          0x61


/*********************************************************************
* Macros: SetPalette(colorNum, color)
*
* PreCondition: none
*
* Input: colorNum - register number, color - color
*
* Output: none
*
* Side Effects: none
*
* Overview:  sets palette register
*
* Note: S6D0129 has no palette
*
*********************************************************************/
    #define SetPalette(colorNum, color)

#endif // HIT1270L.h
```

14.3.5 HX8347.c

This is file HX8347.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  HIMAX HX8347 controller driver
 *****************************************************************************
 * FileName:        HX8347.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Linker:          MPLAB LINK30, MPLAB LINK32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2009 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 05/26/09     ...
```

```
 * 04/12/11     Graphics Library Version 3.00 Support
 ****************************************************************************/
#include "HardwareProfile.h"

#if defined (GFX_USE_DISPLAY_CONTROLLER_HX8347A) || defined
(GFX_USE_DISPLAY_CONTROLLER_HX8347D)

#include "Compiler.h"
#include "TimeDelay.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/HX8347.h"
#include "Graphics/Primitive.h"

#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_EPMP)
    #include "Graphics/gfxepmp.h"
#endif

// Unsupported Graphics Library Features
#ifdef USE_TRANSPARENT_COLOR
    #warning "This driver does not support the transparent feature on PutImage(). Build
will use the PutImage() functions defind in the Primitive.c"
#endif

// Color
GFX_COLOR   _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT   _clipRgn;

// Clipping region borders
SHORT   _clipLeft;
SHORT   _clipTop;
SHORT   _clipRight;
SHORT   _clipBottom;

//////////////////////// LOCAL FUNCTIONS PROTOTYPES ////////////////////////////
void    SetReg(BYTE index, BYTE value);
#ifndef USE_TRANSPARENT_COLOR
void    PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void    PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void    PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void    PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);

void    PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void    PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void    PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void    PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
#endif

/*******************************************************************
* Macros:  WritePixel(data)
*
* Overview: Writes data
*
* PreCondition: none
*
* Input: data
*
* Output: none
*
* Side Effects: none
*
*******************************************************************/

#ifdef USE_16BIT_PMP
#define WritePixel(data)  DeviceWrite(data)
```

```
#else
#define WritePixel(data) \
    DisplaySetData(); \
    DeviceWrite(((WORD_VAL)data).v[1]);\
    DeviceWrite(((WORD_VAL)data).v[0]);
#endif

/*********************************************************************
* Function:  void SetAddress(SHORT x, SHORT y)
*
* Overview: Writes address pointer.
*
* PreCondition: none
*
* Input: x - horizontal position
*        y - vertical position
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
inline void SetAddress(SHORT x, SHORT y)
{
    DisplaySetCommand();
    DeviceWrite(0x02);
    DisplaySetData();
    DeviceWrite(((WORD_VAL) (WORD) x).v[1]);
    DisplaySetCommand();
    DeviceWrite(0x03);
    DisplaySetData();
    DeviceWrite(((WORD_VAL) (WORD) x).v[0]);
    DisplaySetCommand();
    DeviceWrite(0x06);
    DisplaySetData();
    DeviceWrite(((WORD_VAL) (WORD) y).v[1]);
    DisplaySetCommand();
    DeviceWrite(0x07);
    DisplaySetData();
    DeviceWrite(((WORD_VAL) (WORD) y).v[0]);
    DisplaySetCommand();
    DeviceWrite(0x22);
}

/*********************************************************************
* Function:  void  SetReg(BYTE index, BYTE value)
*
* PreCondition: none
*
* Input: index - register number
*        value - value to be set
*
* Output: none
*
* Side Effects: none
*
* Overview: sets graphics controller register
*
* Note: none
*
*********************************************************************/
void SetReg(BYTE index, BYTE value)
{
    DisplayEnable();
    DisplaySetCommand();
    DeviceWrite(index);
    DisplaySetData();
    DeviceWrite(value);
    DisplayDisable();
}

/*********************************************************************
```

```
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*********************************************************************/
void ResetDevice(void)
{
    // Initialize the device
    DriverInterfaceInit();
    DelayMs(1);             // Delay to let controller ready for next SetReg command

#if defined (GFX_USE_DISPLAY_CONTROLLER_HX8347A)
    // Gamma for CMO 2.8
    SetReg(0x46, 0x95);     //
    SetReg(0x47, 0x51);     //
    SetReg(0x48, 0x00);     //
    SetReg(0x49, 0x36);     //
    SetReg(0x4A, 0x11);     //
    SetReg(0x4B, 0x66);     //
    SetReg(0x4C, 0x14);     //
    SetReg(0x4D, 0x77);     //
    SetReg(0x4E, 0x13);     //
    SetReg(0x4F, 0x4C);     //
    SetReg(0x50, 0x46);     //
    SetReg(0x51, 0x46);     //

    // Display
    SetReg(0x90, 0x7F);     // SAP=0111 1111
    SetReg(0x01, 0x06);     // IDMON=0, INVON=1, NORON=1, PTLON=0
    #if (DISP_ORIENTATION == 0)
    SetReg(0x16, 0xC8);     // MY=1, MX=1, MV=0, BGR=1
    #else
    SetReg(0x16, 0xA8);     // MY=1, MX=0, MV=1, BGR=1
    #endif
    SetReg(0x23, 0x95);     // N_DC=1001 0101
    SetReg(0x24, 0x95);     // P_DC=1001 0101
    SetReg(0x25, 0xFF);     // I_DC=1111 1111
    SetReg(0x27, 0x06);     // N_BP=0000 0110
    SetReg(0x28, 0x06);     // N_FP=0000 0110
    SetReg(0x29, 0x06);     // P_BP=0000 0110
    SetReg(0x2A, 0x06);     // P_FP=0000 0110
    SetReg(0x2C, 0x06);     // I_BP=0000 0110
    SetReg(0x2D, 0x06);     // I_FP=0000 0110
    SetReg(0x3A, 0x01);     // N_RTN=0000, N_NW=001
    SetReg(0x3B, 0x01);     // P_RTN=0000, P_NW=001
    SetReg(0x3C, 0xF0);     // I_RTN=1111, I_NW=000
    SetReg(0x3D, 0x00);     // DIV=00
    DelayMs(20);

    SetReg(0x10, 0xA6);     // SS=0,GS=0 CSEL=110

    // Power Supply Setting
    SetReg(0x19, 0x49);     // OSCADJ=10 0000, OSD_EN=1 //60Hz
    SetReg(0x93, 0x0C);     // RADJ=1100,
    DelayMs(10);

    SetReg(0x20, 0x40);     // BT=0100
    SetReg(0x1D, 0x07);     // VC1=111
    SetReg(0x1E, 0x00);     // VC3=000
    SetReg(0x1F, 0x04);     // VRH=0100          4.12V
    SetReg(0x44, 0x4D);     // VCM=101 0000   3.21V
    SetReg(0x45, 0x11);     // VDV=1 0001            -1.19V
```

```
    DelayMs(10);

    SetReg(0x1C, 0x04);     // AP=100
    DelayMs(20);
    SetReg(0x43, 0x80);     //set VCOMG=1
    DelayMs(5);
    SetReg(0x1B, 0x18);     // GASENB=0, PON=1, DK=1, XDK=0, DDVDH_TRI=0, STB=0
    DelayMs(40);

    SetReg(0x1B, 0x10);     // GASENB=0, PON=1, DK=0, XDK=0, DDVDH_TRI=0, STB=0
    DelayMs(40);

    // Display ON Setting
    SetReg(0x26, 0x04);     // GON=0, DTE=0, D=01
    DelayMs(40);
    SetReg(0x26, 0x24);     // GON=1, DTE=0, D=01
    SetReg(0x26, 0x2C);     // GON=1, DTE=0, D=11
    DelayMs(40);

    SetReg(0x26, 0x3C);     // GON=1, DTE=1, D=11
    SetReg(0x35, 0x38);     // EQS=38h
    SetReg(0x36, 0x78);     // EQP=78h
    SetReg(0x3E, 0x38);     // SON=38h
    SetReg(0x40, 0x0F);     // GDON=0Fh
    SetReg(0x41, 0xF0);     // GDOFF

    // Set spulse & rpulse
    SetReg(0x57, 0x02);     // Test mode='1'
    SetReg(0x56, 0x84);     // set Rpulse='1000',spulse='0100'
    SetReg(0x57, 0x00);     // Test mode= '0'
    #if (DISP_ORIENTATION == 0)
    SetReg(0x04, 0x00);
    SetReg(0x05, 0xEF);
    SetReg(0x08, 0x01);
    SetReg(0x09, 0x3F);
    #else
    SetReg(0x04, 0x01);
    SetReg(0x05, 0x3F);
    SetReg(0x08, 0x00);
    SetReg(0x09, 0xEF);
    #endif
    DelayMs(20);

#elif defined (GFX_USE_DISPLAY_CONTROLLER_HX8347D)

    // Driving ability setting
    SetReg(0xEA,0x00);   // PTBA[15:8]
    SetReg(0xEB,0x20);   // PTBA[7:0]
    SetReg(0xEC,0x0C);   // STBA[15:8]
    SetReg(0xED,0xC4);   // STBA[7:0]
    SetReg(0xE8,0x40);   // OPON[7:0]
    SetReg(0xE9,0x38);   // OPON1[7:0]
    SetReg(0xF1,0x01);   // OTPS1B
    SetReg(0xF2,0x10);   // GEN
    SetReg(0x27,0xA3);   //

    // Gamma 2.8 setting
    SetReg(0x40,0x00);   //
    SetReg(0x41,0x00);   //
    SetReg(0x42,0x01);   //
    SetReg(0x43,0x13);   //
    SetReg(0x44,0x10);   //
    SetReg(0x45,0x26);   //
    SetReg(0x46,0x08);   //
    SetReg(0x47,0x51);   //
    SetReg(0x48,0x02);   //
    SetReg(0x49,0x12);   //
    SetReg(0x4A,0x18);   //
    SetReg(0x4B,0x19);   //
    SetReg(0x4C,0x14);   //

    SetReg(0x50,0x19);   //
```

```
    SetReg(0x51,0x2F);   //
    SetReg(0x52,0x2C);   //
    SetReg(0x53,0x3E);   //
    SetReg(0x54,0x3F);   //
    SetReg(0x55,0x3F);   //
    SetReg(0x56,0x2E);   //
    SetReg(0x57,0x77);   //
    SetReg(0x58,0x0B);   //
    SetReg(0x59,0x06);   //
    SetReg(0x5A,0x07);   //
    SetReg(0x5B,0x0D);   //
    SetReg(0x5C,0x1D);   //
    SetReg(0x5D,0xCC);   //

    // Window setting
#if (DISP_ORIENTATION == 0)
    SetReg(0x04,0x00);
    SetReg(0x05,0xEF);
    SetReg(0x08,0x01);
    SetReg(0x09,0x3F);
#else
    SetReg(0x04,0x01);
    SetReg(0x05,0x3F);
    SetReg(0x08,0x00);
    SetReg(0x09,0xEF);
#endif

    // Display Setting
    //SetReg(0x01,0x06);   // IDMON=0, INVON=1, NORON=1, PTLON=0

#if (DISP_ORIENTATION == 0)
    SetReg(0x16,0x08);   // MY=0, MX=0, MV=0, BGR=1
#else
    SetReg(0x16,0x68);   // MY=0, MX=1, MV=1, BGR=1
#endif


    // Power Voltage Setting
    SetReg(0x1B,0x1B);   // VRH = 4.65
    SetReg(0x1A,0x01);   // BT
    SetReg(0x24,0x2F);   // VMH
    SetReg(0x25,0x57);   // VML

    // Vcom offset
    SetReg(0x23,0x8D);   // FLICKER ADJUST

    // Power ON Setting
    SetReg(0x18,0x36);   //
    SetReg(0x19,0x01);   //
    SetReg(0x01,0x00);   //
    SetReg(0x1F,0x88);   //
    DelayMs(5);
    SetReg(0x1F,0x80);   //
    DelayMs(5);
    SetReg(0x1F,0x90);   //
    DelayMs(5);
    SetReg(0x1F,0xD0);   //
    DelayMs(5);

    // 65K Color Selection
    SetReg(0x17,0x05);   //

    // Set Panel
    SetReg(0x36,0x00);   //

    // Display ON Setting
    SetReg(0x28,0x38);   //
    DelayMs( 40);
    SetReg(0x28,0x3C);   //
#endif

}
```

```
#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel
*
* Note: none
*
*********************************************************************/
void PutPixel(SHORT x, SHORT y)
{
    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    DisplayEnable();
    SetAddress(x, y);
    WritePixel(_color);
    DisplayDisable();
}

/*********************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
```

```
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*********************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClip(BYTE control)
{
    _clipRgn=control;
}

/*********************************************************************
* Function: IsDeviceBusy()
*
* Overview: Returns non-zero if LCD controller is busy
*           (previous drawing operation is not completed).
*
* PreCondition: none
*
* Input: none
*
* Output: Busy status.
*
* Side Effects: none
*
*********************************************************************/
WORD IsDeviceBusy(void)
{
    return (0);
}

/*********************************************************************
* Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* PreCondition: none
*
* Input: left,top - top left corner coordinates,
*        right,bottom - bottom right corner coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: draws rectangle filled with current color
*
* Note: none
*
*********************************************************************/
```

```
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    register SHORT  x, y;

    #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
    #else
    if(IsDeviceBusy() != 0)
        return (0);
    #endif
    if(_clipRgn)
    {
        if(left < _clipLeft)
            left = _clipLeft;
        if(right > _clipRight)
            right = _clipRight;
        if(top < _clipTop)
            top = _clipTop;
        if(bottom > _clipBottom)
            bottom = _clipBottom;
    }

    DisplayEnable();
    for(y = top; y < bottom + 1; y++)
    {
        SetAddress(left, y);
        for(x = left; x < right + 1; x++)
        {
            WritePixel(_color);
        }
    }

    DisplayDisable();
    return (1);
}

/*****************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: clears screen with current color
*
* Note: none
*
*****************************************************************/
void ClearDevice(void)
{
    DWORD   counter;

    DisplayEnable();
    SetAddress(0, 0);
    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1); counter++)
    {
        WritePixel(_color);
    }

    DisplayDisable();
}

#ifndef USE_TRANSPARENT_COLOR

    #ifdef USE_BITMAP_FLASH
```

```c
/*********************************************************************
* Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner,
*        image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp = 0;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;
    WORD                pallete[2];
    BYTE                mask;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[0] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[1] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress((WORD) left, (WORD) top);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }
```

```
                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            top++;
        }
    }

    DisplayDisable();
}

/*****************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 16 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************/
void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    BYTE                temp = 0;
    register BYTE       stretchX, stretchY;
    WORD                pallete[16];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top);
            for(x = 0; x < sizeX; x++)
            {
```

```
                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(pallete[temp >> 4]);
                }
                else
                {
                    temp = *flashAddress;
                    flashAddress++;

                    // first pixel in byte
                    SetColor(pallete[temp & 0x0f]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            top++;
        }
    }

    DisplayDisable();
}

/*****************************************************************************
* Function: void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 256 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                temp;
    BYTE                stretchX, stretchY;
    WORD                pallete[256];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
```

```
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top);
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            top++;
        }
    }

    DisplayDisable();
}

/***************************************************************************
* Function: void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs hicolor image starting from left,top coordinates
*
* Note: image must be located in flash
*
***************************************************************************/
void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_WORD *flashAddress;
    register FLASH_WORD *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                temp;
    register BYTE       stretchX, stretchY;

    // Move pointer to size information
    flashAddress = (FLASH_WORD *)image + 1;

    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
```

```
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top);
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }

            top++;
        }
    }

    DisplayDisable();
}

    #endif // USE_BITMAP_FLASH
    #ifdef USE_BITMAP_EXTERNAL

/*********************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*********************************************************************/
void PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[2];
    BYTE            lineBuffer[((GetMaxX() + 1) / 8) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp;
    BYTE            mask;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);
```

```
    // Line width in bytes
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *pData++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            top++;
        }

        DisplayDisable();
    }
}

/*****************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
```

```
 *
 * Note: image must be located in external memory
 *
 ***********************************************************************/
void PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[16];
    BYTE            lineBuffer[((GetMaxX() + 1) / 2) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top);

            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(pallete[temp >> 4]);
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    SetColor(pallete[temp & 0x0f]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
                }
            }
```

```
            top++;
        }

        DisplayDisable();
    }
}

/*********************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*********************************************************************/
void PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[256];
    BYTE            lineBuffer[(GetMaxX() + 1)];
    BYTE            *pData;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
        memOffset += sizeX;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    WritePixel(_color);
```

```
                }
            }

            top++;
        }

        DisplayDisable();
    }
}

/*****************************************************************
* Function: void PutImage16BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************/
void PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            lineBuffer[(GetMaxX() + 1)];
    WORD            *pData;
    WORD            byteWidth;

    WORD            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    byteWidth = sizeX << 1;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
```

```
                    WritePixel(_color);
                }
            }

            top++;
        }

        DisplayDisable();
    }
}

    #endif //USE_BITMAP_EXTERNAL
#endif //#ifndef USE_TRANSPARENT_COLOR

#endif // USE_DRV_PUTIMAGE
```

14.3.6 HX8347.h

This is file HX8347.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  HIMAX HX8347 LCD controllers driver
 *****************************************************************************
 * FileName:        HX8347.h
 * Dependencies:    p24Fxxxx.h or plib.h
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Linker:          MPLAB LINK30, MPLAB LINK32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2009 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 05/26/09     ...
 * 04/12/11     Graphics Library Version 3.00 Support
 *****************************************************************************/
#ifndef _HX8347_H
    #define _HX8347_H

    #include "HardwareProfile.h"
    #include "GraphicsConfig.h"
    #include "GenericTypeDefs.h"
```

```
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef COLOR_DEPTH
        #error COLOR_DEPTH must be defined in GraphicsConfig.h
    #endif
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* Overview: Horizontal and vertical screen size.
*********************************************************************/
    #if (DISP_HOR_RESOLUTION != 240)
        #error This driver doesn't supports this resolution. Horisontal resolution must be
240 pixels.
    #endif
    #if (DISP_VER_RESOLUTION != 320)
        #error This driver doesn't supports this resolution. Vertical resolution must be
320 pixels.
    #endif

/*********************************************************************
* Overview: Display orientation.
*********************************************************************/
    #if (DISP_ORIENTATION != 0) && (DISP_ORIENTATION != 90)
        #error This driver doesn't support this orientation.
    #endif

/*********************************************************************
* Overview: Color depth.
*********************************************************************/
    #if (COLOR_DEPTH != 16)
        #error This driver doesn't support this color depth. It should be 16.
    #endif

/*********************************************************************
* Macros: SetPalette(colorNum, color)
*
* Overview:  Sets palette register.
*
* PreCondition: none
*
* Input: colorNum - Register number.
*        color - Color.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define SetPalette(colorNum, color)


#endif // _HX8347_H
```

14.3.7 SH1101A_SSD1303.c

This is file SH1101A_SSD1303.c.

Body Source

```
/****************************************************************************
 *  Module for Microchip Graphics Library
```

```
 *  Sino Wealth Microelectronic SH1101A OLED controller driver
 *  Solomon Systech SSD1303 LCD controller driver
 ****************************************************************************
 * FileName:        SH1101A_SSD1303.c
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 03/10/07     Original
 * 12/20/07     Ported to PIC24 Kit
 * 03/26/10     Fixed error on PutPixel() timing (PMP timing).
 * 03/22/11     - Modified for Graphics Library Version 3.00
 *              - Replace color data type from WORD_VAL to GFX_COLOR
 *              - Replace DeviceInit() to DriverInterfaceInit()
 ****************************************************************************/
#include "HardwareProfile.h"

#if defined(USE_GFX_DISPLAY_CONTROLLER_SH1101A)

#include "Compiler.h"
#include "TimeDelay.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/SH1101A_SSD1303.h"
#include "Graphics/gfxtcon.h"
#include "Graphics/Primitive.h"

#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_PMP)
    #include "Graphics/gfxepmp.h"
#endif

// Color
GFX_COLOR   _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT   _clipRgn;

// Clipping region borders
SHORT   _clipLeft;
SHORT   _clipTop;
SHORT   _clipRight;
```

```
SHORT   _clipBottom;


/*****************************************************************
* Macros:  SetAddress(lowerAddr,higherAddr)
*
* Overview: Sets the page and the lower and higher address pointer
*           of the display buffer. All parameters should be pre-calculated.
*
* PreCondition: none
*
* Input: page - Page value for the address.
*        lowerAddr - Lower column address.
*        higherAddr - Higher column address.
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
#define SetAddress(page, lowerAddr, higherAddr) \
    DisplaySetCommand(); \
    DeviceWrite(page); \
    DeviceWrite(lowerAddr); \
    DeviceWrite(higherAddr); \
    DisplaySetData();


/*****************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*****************************************************************/
void ResetDevice(void)
{
    // Initialize the device
    DriverInterfaceInit();

    DisplayEnable();
    DisplaySetCommand();

    #if (DISPLAY_CONTROLLER == SH1101A)

    // Setup Display
    DeviceWrite(0xAE);              // turn off the display (AF=ON, AE=OFF)
    DeviceWrite(0xDB);              // set  VCOMH
    DeviceWrite(0x23);
    DeviceWrite(0xD9);              // set  VP
    DeviceWrite(0x22);

    // Re-map
    DeviceWrite(0xA1);              // [A0]:column address 0 is map to SEG0

    // [A1]:column address 131 is map to SEG0
    // COM Output Scan Direction
    DeviceWrite(0xC8);              // C0 is COM0 to COMn, C8 is COMn to COM0

    // COM Pins Hardware Configuration
    DeviceWrite(0xDA);              // set pins hardware configuration
    DeviceWrite(0x12);
```

```
    // Multiplex Ratio
    DeviceWrite(0xA8);              // set multiplex ratio
    DeviceWrite(0x3F);              // set to 64 mux

    // Display Clock Divide
    DeviceWrite(0xD5);              // set display clock divide
    DeviceWrite(0xA0);              // set to 100Hz

    // Contrast Control Register
    DeviceWrite(0x81);              // Set contrast control
    DeviceWrite(0x60);              // display 0 ~ 127; 2C

    // Display Offset
    DeviceWrite(0xD3);              // set display offset
    DeviceWrite(0x00);              // no offset

    //Normal or Inverse Display
    DeviceWrite(0xA6);              // Normal display
    DeviceWrite(0xAD);              // Set DC-DC
    DeviceWrite(0x8B);              // 8B=ON, 8A=OFF

    // Display ON/OFF
    DeviceWrite(0xAF);              // AF=ON, AE=OFF
    DelayMs(150);

    // Entire Display ON/OFF
    DeviceWrite(0xA4);              // A4=ON

    // Display Start Line
    DeviceWrite(0x40);              // Set display start line

    // Lower Column Address
    DeviceWrite(0x00 + OFFSET);     // Set lower column address

    // Higher Column Address
    DeviceWrite(0x10);              // Set higher column address
    DelayMs(1);
    #elif (DISPLAY_CONTROLLER == SSD1303)

    // Setup Display
    DeviceWrite(0xAE);              // turn off the display (AF=ON, AE=OFF)
    DeviceWrite(0xDB);              // set  VCOMH
    DeviceWrite(0x23);
    DeviceWrite(0xD9);              // set  VP
    DeviceWrite(0x22);
    DeviceWrite(0xAD);              // Set DC-DC
    DeviceWrite(0x8B);              // 8B=ON, 8A=OFF
    DelayMs(1);

    // Display ON/OFF
    DeviceWrite(0xAF);              // AF=ON, AE=OFF

    // Init OLED display using SSD1303 driver
    // Display Clock Divide
    DeviceWrite(0xD5);              // set display clock divide
    DeviceWrite(0xA0);              // set to 100Hz

    // Display Offset
    DeviceWrite(0xD3);              // set display offset
    DeviceWrite(0x00);              // no offset

    // Multiplex Ratio
    DeviceWrite(0xA8);              // set multiplex ratio
    DeviceWrite(0x3F);              // set to 64 mux

    // Display Start Line
    DeviceWrite(0x40);              // Set display start line

    // Re-map
    DeviceWrite(0xA0);              // [A0]:column address 0 is map to SEG0

    // [A1]:column address 131 is map to SEG0
```

```
    // COM Output Scan Direction
    DeviceWrite(0xC8);              // C0 is COM0 to COMn, C8 is COMn to COM0

    // COM Pins Hardware Configuration
    DeviceWrite(0xDA);              // set pins hardware configuration
    DeviceWrite(0x12);

    // Contrast Control Register
    DeviceWrite(0x81);              // Set contrast control
    DeviceWrite(0x60);              // display 0 ~ 127; 2C

    // Entire Display ON/OFF
    DeviceWrite(0xA4);              // A4=ON

    //Normal or Inverse Display
    DeviceWrite(0xA6);              // Normal display

#ifdef DISABLE_DC_DC_CONVERTER
    DeviceWrite(0x8A);
#else
    DeviceWrite(0x8B);              // 8B=ON, 8A=OFF
#endif

    // Lower Column Address
    DeviceWrite(0x00 + OFFSET);     // Set lower column address

    // Higher Column Address
    DeviceWrite(0x10);              // Set higher column address
    DelayMs(1);
    #else
        #error The controller is not supported.
    #endif
    DisplayDisable();
    DisplaySetData();
}

#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel
*
```

```
 * Note: none
 *
 ***************************************************************/
void PutPixel(SHORT x, SHORT y)
{
    BYTE    page, add, lAddr, hAddr;
    BYTE    mask, display;

    // check if point is in clipping region
    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    // Assign a page address
    if(y < 8)
        page = 0xB0;
    else if(y < 16)
        page = 0xB1;
    else if(y < 24)
        page = 0xB2;
    else if(y < 32)
        page = 0xB3;
    else if(y < 40)
        page = 0xB4;
    else if(y < 48)
        page = 0xB5;
    else if(y < 56)
        page = 0xB6;
    else
        page = 0xB7;

    add = x + OFFSET;
    lAddr = 0x0F & add;             // Low address
    hAddr = 0x10 | (add >> 4);      // High address

    // Calculate mask from rows basically do a y%8 and remainder is bit position
    add = y >> 3;                   // Divide by 8
    add <<= 3;                      // Multiply by 8
    add = y - add;                  // Calculate bit position
    mask = 1 << add;                // Left shift 1 by bit position
    DisplayEnable();

    SetAddress(page, lAddr, hAddr); // Set the address (sets the page,


    // lower and higher column address pointers)
    display = SingleDeviceRead();   // Read to initiate Read transaction on PMP and dummy
read
                                    // (requirement for data synchronization in the
controller)
    display = SingleDeviceRead();   // Read again as a requirement for data synchronization
in the display controller
    display = SingleDeviceRead();   // Read actual data from from display buffer

    if(_color > 0)                  // If non-zero for pixel on
        display |= mask;            // or in mask
    else
        // If 0 for pixel off
        display &= ~mask;           // and with inverted mask
    SetAddress(page, lAddr, hAddr); // Set the address (sets the page,

    // lower and higher column address pointers)
    DeviceWrite(display);             // restore the byte with manipulated bit
    DisplayDisable();
```

```
}

/*********************************************************************
* Function: BYTE GetPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: pixel color
*
* Side Effects: none
*
* Overview: return pixel color at x,y position
*
* Note: none
*
*********************************************************************/
BYTE GetPixel(SHORT x, SHORT y)
{
    BYTE    page, add, lAddr, hAddr;
    BYTE    mask, temp, display;

    // check if point is in clipping region
    if(_clipRgn)
    {
        if(x < _clipLeft)
            return (0);
        if(x > _clipRight)
            return (0);
        if(y < _clipTop)
            return (0);
        if(y > _clipBottom)
            return (0);
    }

    // Assign a page address
    if(y < 8)
        page = 0xB0;
    else if(y < 16)
        page = 0xB1;
    else if(y < 24)
        page = 0xB2;
    else if(y < 32)
        page = 0xB3;
    else if(y < 40)
        page = 0xB4;
    else if(y < 48)
        page = 0xB5;
    else if(y < 56)
        page = 0xB6;
    else
        page = 0xB7;

    add = x + OFFSET;
    lAddr = 0x0F & add;             // Low address
    hAddr = 0x10 | (add >> 4);      // High address

    // Calculate mask from rows basically do a y%8 and remainder is bit position
    temp = y >> 3;                  // Divide by 8
    temp <<= 3;                     // Multiply by 8
    temp = y - temp;                // Calculate bit position
    mask = 1 << temp;               // Left shift 1 by bit position
    DisplayEnable();

    SetAddress(page, lAddr, hAddr); // Set the address (sets the page,

    // lower and higher column address pointers)
    display = SingleDeviceRead();   // Read to initiate Read transaction on PMP
                                    // Dummy Read (requirement for data synchronization in
the controller)
    display = DeviceRead();         // Read data from display buffer
```

```
    DisplayDisable();

    return (display & mask);        // mask all other bits and return the result
}

/*********************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*********************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClip(BYTE control)
{
    _clipRgn=control;
}

/*********************************************************************
* Function: IsDeviceBusy()
*
* Overview: Returns non-zero if LCD controller is busy
*           (previous drawing operation is not completed).
*
* PreCondition: none
*
* Input: none
*
* Output: Busy status.
*
* Side Effects: none
*
*********************************************************************/
WORD IsDeviceBusy(void)
{
    return (0);
}
```

```
/*****************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: clears screen with current color
*
* Note: none
*
*****************************************************************/
void ClearDevice(void)
{
    BYTE    i, j;
    DisplayEnable();
    for(i = 0xB0; i < 0xB8; i++)
    {       // Go through all 8 pages
        SetAddress(i, 0x00, 0x10);
        for(j = 0; j < 132; j++)
        {   // Write to all 132 bytes
            DeviceWrite(_color);
        }
    }

    DisplayDisable();
}


#endif // #if defined(USE_GFX_DISPLAY_CONTROLLER_SH1101A)
```

14.3.8 SH1101A_SSD1303.h

Macros

Name	Description
SetPalette (see page 396)	Sets palette register.

Description

This is file SH1101A_SSD1303.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Sino Wealth Microelectronic SH1101A OLED controller driver
 *  Solomon Systech SSD1303 LCD controller driver
  ****************************************************************************
 * FileName:        SH1101A_SSD1303.h
 * Dependencies:    p24Fxxxx.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 03/10/07     Original
 * 12/20/07     Ported to PIC24 Kit
 * 03/22/11     Modified for Graphics Library Version 3.00
 ****************************************************************************/
#ifndef _SH1101A_SSD1303_OLED_H
    #define _SH1101A_SSD1303_OLED_H

    #include "HardwareProfile.h"
    #include "GenericTypeDefs.h"
    #include "GraphicsConfig.h"

#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_PMP)
    #include "Graphics/gfxepmp.h"
#endif

    #ifdef USE_16BIT_PMP
        #error This driver doesn't support 16-bit PMP (remove USE_16BIT_PMP option from
HardwareProfile.h)
    #endif
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef COLOR_DEPTH
        #error COLOR_DEPTH must be defined in GraphicsConfig.h
    #endif
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* Overview: Horizontal and vertical screen size.
*********************************************************************/
    #if (DISP_HOR_RESOLUTION != 128)
        #error This driver doesn't supports this resolution. Horisontal resolution must be
128 pixels.
    #endif
    #if (DISP_VER_RESOLUTION != 64)
        #error This driver doesn't supports this resolution. Vertical resolution must be 64
pixels.
    #endif

/*********************************************************************
* Overview: Display orientation.
*********************************************************************/
    #if (DISP_ORIENTATION != 0)
        #error This driver doesn't support this orientation.
    #endif
```

```
/*********************************************************************
* Overview: Defines the display offset in x direction. Dependent on the display
* used and how it is connected.
*********************************************************************/
    #define OFFSET  2

/*********************************************************************
* Overview: Clipping region control codes to be used with SetClip(...)
*           function.
*********************************************************************/
    #define CLIP_DISABLE    0   // Disables clipping.
    #define CLIP_ENABLE     1   // Enables clipping.

/*********************************************************************
* Overview: Screen Saver parameters.
*           - SSON - Means that screen saver will be enabled when
*                    ScreenSaver(SSON) function is called with SSON as
*                    parameter.
*           - SSOFF - Means that screen saver will be disbled when
*                    ScreenSaver(SSOFF) function is called with SSOFF as
*                    parameter.
*
*********************************************************************/
    #define SSON    1           // screen saver is turned on
    #define SSOFF   0           // screen saver is turned off

    // Memory pitch for line
    #define LINE_MEM_PITCH  0x100

/*********************************************************************
* Macros: SetPalette(colorNum, color)
*
* Overview:  Sets palette register.
*
* PreCondition: none
*
* Input: colorNum - Register number.
*        color - Color.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define SetPalette(colorNum, color)

#endif // _SH1101A_SSD1303_OLED_H
```

14.3.9 SSD1339.c

This is file SSD1339.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Solomon Systech SSD1339 OLED display controller driver
 ******************************************************************************
 * FileName:        SSD1339.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/12/07     Version 1.0 release
 * 04/11/11     - Modified for Graphics Library Version 3.00
 *              - Replace color data type from WORD_VAL to GFX_COLOR
 *              - Replace DeviceInit() to DriverInterfaceInit()
 ***************************************************************************/
#include "HardwareProfile.h"

#if defined(GFX_USE_DISPLAY_CONTROLLER_SSD1339)

#include "Compiler.h"
#include "GenericTypeDefs.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/SSD1339.h"
#include "Graphics/Primitive.h"

#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_PMP)
    #include "Graphics/gfxepmp.h"
#endif

// Unsupported Graphics Library Features
#ifdef USE_TRANSPARENT_COLOR
    #warning "This driver does not support the transparent feature on PutImage(). Build
will use the PutImage() functions defind in the Primitive.c"
#endif

// Color
GFX_COLOR    _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT       _clipRgn;

// Clipping region borders
SHORT       _clipLeft;
SHORT       _clipTop;
SHORT       _clipRight;
SHORT       _clipBottom;

////////////////////// LOCAL FUNCTIONS PROTOTYPES ///////////////////////////
#ifndef USE_TRANSPARENT_COLOR
void        PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void        PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
```

```
void        PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
#endif

/*****************************************************************
* Macros:  SetAddress(x, y)
*
* PreCondition: none
*
* Input: x,y - start x,y position
*
* Output: none
*
* Side Effects: none
*
* Overview: sets pointer to to write/read operations
*
* Note: none
*
*****************************************************************/
#define SetAddress(x, y) \
    DisplaySetCommand(); \
    DeviceWrite(CMD_COL); \
    DisplaySetData(); \
    DeviceWrite(x); \
    DisplaySetCommand(); \
    DeviceWrite(CMD_ROW); \
    DisplaySetData(); \
    DeviceWrite(y);

/*****************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*****************************************************************/
void ResetDevice(void)
{
    // Initialize device
    DriverInterfaceInit();

    // Enable LCD
    DisplayEnable();

    // Setup display
    DisplaySetCommand();
    DeviceWrite(CMD_DISPON);
    DeviceWrite(CMD_MODE);
    DisplaySetData();
    DeviceWrite(0x74);
    DisplaySetCommand();
    DeviceWrite(CMD_SRTLINE);
    DisplaySetData();
    DeviceWrite(128);

    // Disable LCD
    DisplayDisable();
}
```

```
#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel
*
* Note: none
*
*********************************************************************/
void PutPixel(SHORT x, SHORT y)
{
    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    DisplayEnable();
    SetAddress(x, y);
    DisplaySetCommand();
    DeviceWrite(CMD_WRITE);
    DisplaySetData();
    DeviceWrite(((WORD_VAL)_color).v[1]);
    DeviceWrite(((WORD_VAL)_color).v[0]);
    DisplayDisable();
}

/*********************************************************************
* Function: WORD GetPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: pixel color
*
```

```
 * Side Effects: none
 *
 * Overview: return pixel color at x,y position
 *
 * Note: none
 *
 ********************************************************************/
WORD GetPixel(SHORT x, SHORT y)
{
    WORD_VAL    result;

    DisplayEnable();

    SetAddress(x, y);

    DisplaySetCommand();
    DeviceWrite(CMD_READ);
    DisplaySetData();

    result.v[0] = DeviceRead();
    result.v[1] = DeviceRead();

    DisplayDisable();

    return (result.Val);
}

/********************************************************************
 * Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
 *
 * PreCondition: none
 *
 * Input: left,top - top left corner coordinates,
 *        right,bottom - bottom right corner coordinates
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 * Overview: draws rectangle filled with current color
 *
 * Note: none
 *
 ********************************************************************/
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    SHORT   x, y;

    #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
    #else
    if(IsDeviceBusy() != 0)
        return (0);
    #endif
    if(_clipRgn)
    {
        if(left < _clipLeft)
            left = _clipLeft;
        if(right > _clipRight)
            right = _clipRight;
        if(top < _clipTop)
            top = _clipTop;
        if(bottom > _clipBottom)
            bottom = _clipBottom;
    }
```

```
    DisplayEnable();

    for(y = top; y < bottom + 1; y++)
    {
        SetAddress(left, top);
        DisplaySetCommand();
        DeviceWrite(CMD_WRITE);
        DisplaySetData();
        for(x = left; x < right + 1; x++)
        {
            DeviceWrite(((WORD_VAL)_color).v[1]);
            DeviceWrite(((WORD_VAL)_color).v[0]);
        }
    }

    DisplayDisable();
    return (1);
}

/*********************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*********************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClip(BYTE control)
{
    _clipRgn=control;
}

/*********************************************************************
* Function: IsDeviceBusy()
*
* Overview: Returns non-zero if LCD controller is busy
*           (previous drawing operation is not completed).
*
```

```
 * PreCondition: none
 *
 * Input: none
 *
 * Output: Busy status.
 *
 * Side Effects: none
 *
 ********************************************************************/
WORD IsDeviceBusy(void)
{
    return (0);
}

/********************************************************************
 * Function: void ClearDevice(void)
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: clears screen with current color
 *
 * Note: none
 *
 ********************************************************************/
void ClearDevice(void)
{
    DWORD   counter;

    DisplayEnable();
    SetAddress(0, 0);
    DisplaySetCommand();
    DeviceWrite(CMD_WRITE);
    DisplaySetData();
    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1); counter++)
    {
        DeviceWrite(((WORD_VAL)_color).v[1]);
        DeviceWrite(((WORD_VAL)_color).v[0]);
    }

    DisplayDisable();
}

#ifndef USE_TRANSPARENT_COLOR

#ifdef USE_BITMAP_FLASH

/********************************************************************
 * Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
 *
 * PreCondition: none
 *
 * Input: left,top - left top image corner, image - image pointer,
 *        stretch - image stretch factor
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: outputs monochrome image starting from left,top coordinates
 *
 * Note: image must be located in flash
 *
 ********************************************************************/
void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
```

```
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                stretchX, stretchY;
    WORD                pallete[2];
    BYTE                mask;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[0] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[1] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }

                // Shift to the next pixel
                mask >>= 1;
            }
        }
    }

    DisplayDisable();
}

/*****************************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
```

```
 *
 * PreCondition: none
 *
 * Input: left,top - left top image corner, image - image pointer,
 *        stretch - image stretch factor
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: outputs 16 color image starting from left,top coordinates
 *
 * Note: image must be located in flash
 *
 ********************************************************************/
void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    BYTE                temp;
    register BYTE       stretchX, stretchY;
    WORD                pallete[16];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if((x & 0x0001) == 0)
                {

                    // Set color
                    SetColor(pallete[temp >> 4]);
                }
                else
                {
                    temp = *flashAddress;
                    flashAddress++;

                    // Set color
                    SetColor(pallete[temp & 0x0f]);
                }
```

```
                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }

                // Shift to the next pixel
                temp >>= 4;
            }
        }
    }

    DisplayDisable();
}

/*****************************************************************
* Function: void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 256 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************/
void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    WORD                x, y;
    BYTE                temp;
    BYTE                stretchX, stretchY;
    WORD                pallete[256];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            for(x = 0; x < sizeX; x++)
```

```
            {

                // Read pixels from flash
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(pallete[temp]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }
            }
        }
    }

    DisplayDisable();
}

/*********************************************************************
* Function: void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs hicolor image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_WORD *flashAddress;
    register FLASH_WORD *tempFlashAddress;
    WORD                sizeX, sizeY;
    register WORD       x, y;
    WORD                temp;
    register BYTE       stretchX, stretchY;

    // Move pointer to size information
    flashAddress = (FLASH_WORD *)image + 1;

    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;

    DisplayEnable();
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
```

```
                temp = *flashAddress;
                flashAddress++;

                // Set color
                SetColor(temp);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }
            }
        }
    }

    DisplayDisable();
}

#endif //USE_BITMAP_FLASH
#ifdef USE_BITMAP_EXTERNAL

/*********************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[2];
    BYTE            lineBuffer[((GetMaxX() + 1) / 8) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp;
    BYTE            mask;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;
```

```
    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *pData++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(pallete[1]);
                }
                else
                {
                    SetColor(pallete[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }

                // Shift to the next pixel
                mask >>= 1;
            }
        }

        DisplayDisable();
    }
}

/*************************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*************************************************************************/
void PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
```

```
    WORD            pallete[16];
    BYTE            lineBuffer[((GetMaxX() + 1) / 2) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(pallete[temp >> 4]);
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    SetColor(pallete[temp & 0x0f]);
                }

                // Set color
                SetColor(pallete[temp & 0x0f]);

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }

                // Shift to the next pixel
                temp >>= 4;
```

```
            }
        }

        DisplayDisable();
    }
}

/*********************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[256];
    BYTE            lineBuffer[(GetMaxX() + 1)];
    BYTE            *pData;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
        memOffset += sizeX;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            for(x = 0; x < sizeX; x++)
            {

                // Read pixels from flash
                temp = *pData++;

                // Set color
                SetColor(pallete[temp]);
```

```
                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    DeviceWrite(((WORD_VAL)_color).v[1]);
                    DeviceWrite(((WORD_VAL)_color).v[0]);
                }
            }
        }

        DisplayDisable();
    }
}

/*****************************************************************
* Function: void PutImage16BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************/
void PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            lineBuffer[(GetMaxX() + 1)];
    WORD            *pData;
    WORD            byteWidth;

    WORD            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    byteWidth = sizeX << 1;

    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        DisplayEnable();
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            SetAddress(left, top + y);
            DisplaySetCommand();
            DeviceWrite(CMD_WRITE);
            DisplaySetData();
            for(x = 0; x < sizeX; x++)
            {
```

```
                    // Read pixels from flash
                    temp = *pData++;

                    // Set color
                    SetColor(temp);

                    // Write pixel to screen
                    for(stretchX = 0; stretchX < stretch; stretchX++)
                    {
                        DeviceWrite(((WORD_VAL)_color).v[1]);
                        DeviceWrite(((WORD_VAL)_color).v[0]);
                    }
                }
            }

            DisplayDisable();
        }
    }

    #endif // USE_BITMAP_EXTERNAL
    #endif //#ifndef USE_TRANSPARENT_COLOR
    #endif //if defined(GFX_USE_DISPLAY_CONTROLLER_SSD1339)
```

14.3.10 SSD1339.h

This is file SSD1339.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Solomon Systech. SSD1339 OLED display controller driver
 *****************************************************************************
 * FileName:        SSD1339.h
 * Dependencies:    p24Fxxxx.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
 * 11/12/07      Version 1.0 release
 * 04/11/11      - Modified for Graphics Library Version 3.00
 ***************************************************************************/
#ifndef _SSD1339_H
    #define _SSD1339_H

    #include "HardwareProfile.h"
    #include "GenericTypeDefs.h"
    #include "GraphicsConfig.h"

#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_PMP)
    #include "Graphics/gfxepmp.h"
#endif

    #ifdef USE_16BIT_PMP
        #error This driver doesn't support 16-bit PMP (remove USE_16BIT_PMP option from
HardwareProfile.h)
    #endif
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef COLOR_DEPTH
        #error COLOR_DEPTH must be defined in GraphicsConfig.h
    #endif
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* Overview: Horizontal and vertical screen size.
*********************************************************************/
    #if (DISP_HOR_RESOLUTION != 128)
        #error This driver doesn't supports this resolution. Horisontal resolution must be
128 pixels.
    #endif
    #if (DISP_VER_RESOLUTION != 128)
        #error This driver doesn't supports this resolution. Vertical resolution must be
128 pixels.
    #endif

/*********************************************************************
* Overview: Display orientation.
*********************************************************************/
    #if (DISP_ORIENTATION != 0)
        #error This driver doesn't support this orientation.
    #endif

/*********************************************************************
* Overview: Color depth.
*********************************************************************/
    #if (COLOR_DEPTH != 16)
        #error This driver doesn't support this color depth. It should be 16.
    #endif

// Clipping region control codes
    #define CLIP_DISABLE    0
    #define CLIP_ENABLE     1

// Display commands
    #define CMD_COL         0x15
    #define CMD_ROW         0x75
    #define CMD_WRITE       0x5C
    #define CMD_READ        0x5D
    #define CMD_MODE        0xA0
    #define CMD_SRTLINE     0xA1
    #define CMD_DISPON      0xAF
    #define CMD_DISPOFF     0xAE
```

```
    #define CMD_FILLMODE    0x92
    #define CMD_RECT        0x84


/*********************************************************************
* Macros: SetPalette(colorNum, color)
*
* PreCondition: none
*
* Input: colorNum - register number, color - color
*
* Output: none
*
* Side Effects: none
*
* Overview:  sets palette register
*
* Note: SSD1339 has no palette
*
*********************************************************************/
    #define SetPalette(colorNum, color)


#endif // _SSD1339_H
```

14.3.11 SSD1926.c

This is file SSD1926.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Solomon Systech. SSD1926 LCD controller driver
 *  to be used with GFX 3 PICtail board
 *****************************************************************************
 * FileName:        SSD1926.c
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright ï¿½ 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 ï¿½AS ISï¿½ 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 08/27/08
```

```
 * 06/25/09     dsPIC & PIC24H support
 * 07/30/09     Added Palette Support
 * 02/05/10     Fixed GetPixel() bug
 * 03/03/10     Fixed Circle() bug
 * 03/26/10     Fixed Line2D() bug
 * 07/28/10     Add USE_DRV_PUTIMAGE label to PutImage() related functions
 *              to fix build error when driver PutImage() is not used.
 * 01/27/11     Fixed Bar() bug when clipping region is enabled.
 * 02/11/11     Added backlight enabling after display controller and
 *              Timing Controller (TCON) initialization.
 * 02/14/11     - Removed USE_DRV_xxxx switches. This is not needed anymore
 *                since Primitive Layer implements weak attributes on
 *                Primitive Routines that can be implemented in hardware.
 *              - Removed Circle Function implementation (SSD1926 bug)
 *              - Replace color data type from WORD_VAL to GFX_COLOR
 *              - Replace DeviceInit() to DriverInterfaceInit()
 *              - Optimized PutImageXBPP() & PutImageXBPPExt() for performance.
 *              - Added transparent color feature in PutImageXBPP() and
 *                PutImageXBPPExt().
 *              - Added DisplayBrightness() to control the backlight.
 *              - Added GFX_LCD_TYPE to select type of display
 ****************************************************************************/
#include "HardwareProfile.h"

#if defined(GFX_USE_DISPLAY_CONTROLLER_SSD1926)

#include "Compiler.h"
#include "TimeDelay.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/SSD1926.h"
#include "Graphics/gfxtcon.h"
#include "Graphics/Primitive.h"


#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_EPMP)
    #include "Graphics/gfxepmp.h"
#endif

// Color
GFX_COLOR   _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT       _clipRgn;

// Clipping region borders
SHORT       _clipLeft;
SHORT       _clipTop;
SHORT       _clipRight;
SHORT       _clipBottom;

#define RED8(color16)   (BYTE) ((color16 & 0xF800) >> 8)
#define GREEN8(color16) (BYTE) ((color16 & 0x07E0) >> 3)
#define BLUE8(color16)  (BYTE) ((color16 & 0x001F) << 3)

////////////////////// LOCAL FUNCTIONS PROTOTYPES ///////////////////////////
void        SetAddress(DWORD address);
BYTE        GetReg(WORD index);

void        PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *bitmap, BYTE stretch);
void        PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *bitmap, BYTE stretch);
void        PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *bitmap, BYTE stretch);
void        PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *bitmap, BYTE stretch);

void        PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void        PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
```

```
void        PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);

#ifdef USE_PALETTE
extern void *_palette;
static BYTE PaletteBpp = 16;
extern BYTE blPaletteChangeError;
extern void *pPendingPalette;
extern WORD PendingStartEntry;
extern WORD PendingLength;
#endif

/*********************************************************************
* Macro:  WritePixel(color)
*
* PreCondition: none
*
* Input: color
*
* Output: none
*
* Side Effects: none
*
* Overview: writes pixel at the current address
*
* Note: chip select should be enabled
*
*********************************************************************/
#ifdef USE_16BIT_PMP
#define WritePixel(color)   DeviceWrite(color)
#else
#ifdef USE_PALETTE
#define WritePixel(color)   DeviceWrite(color)
#else
#define WritePixel(color)   { DeviceWrite(((WORD_VAL)color).v[1]);
DeviceWrite(((WORD_VAL)color).v[0]);}
#endif
#endif

/*********************************************************************
* Function:  void SetAddress(DWORD address)
*
* PreCondition: none
*
* Input: address - address
*
* Output: none
*
* Side Effects: none
*
* Overview: sets the address for read/write operations
*
* Note: chip select should be enabled
*
*********************************************************************/
void SetAddress(DWORD address)
{
    #ifdef USE_16BIT_PMP
    WORD_VAL    temp;

    DisplaySetCommand(); // set RS line to low for command

    temp.v[0] = ((DWORD_VAL) address).v[1];
    temp.v[1] = ((DWORD_VAL) address).v[2] | 0x80;
    DeviceWrite(temp.Val);
    temp.v[0] = 0x01;
    temp.v[1] = ((DWORD_VAL) address).v[0];
    DeviceWrite(temp.Val);

    DisplaySetData();    // set RS line to high for data

    #else
```

```
    DisplaySetCommand(); // set RS line to low for command

    DeviceWrite(((DWORD_VAL) address).v[2] | 0x80);
    DeviceWrite(((DWORD_VAL) address).v[1]);
    DeviceWrite(((DWORD_VAL) address).v[0]);

    DisplaySetData();    // set RS line to high for data

    #endif
}

/*********************************************************************
* Function:  void  SetReg(WORD index, BYTE value)
*
* PreCondition: none
*
* Input: index - register number
*        value - value to be set
*
* Output: none
*
* Side Effects: none
*
* Overview: sets graphics controller register (byte access)
*
* Note: none
*
*********************************************************************/
void SetReg(WORD index, BYTE value)
{
    #ifdef USE_16BIT_PMP

    DisplaySetCommand(); // set RS line to low for command
    DisplayEnable();     // enable SSD1926

    DeviceWrite(((WORD_VAL) index).v[1]);
    DeviceWrite(index << 8);

    DisplaySetData();    // set RS line to high for data

    if(index & 0x0001)
        DeviceWrite(value);
    else
        DeviceWrite(value << 8);

    DisplayDisable();   // disable SSD1926

    #else

    DisplaySetCommand(); // set RS line to low for command
    DisplayEnable();     // enable SSD1926

    DeviceWrite(0x00);    // register access
    DeviceWrite(((WORD_VAL) index).v[1]);
    DeviceWrite(((WORD_VAL) index).v[0]);

    DisplaySetData();    // set RS line to high for data
    DeviceWrite(value);

    DisplayDisable();   // disable SSD1926
    #endif
}

/*********************************************************************
* Function:  BYTE GetReg(WORD index)
*
* PreCondition: none
*
* Input: index - register number
*
* Output: none
*
```

```
* Side Effects: none
*
* Overview: returns graphics controller register value (byte access)
*
* Note: none
*
*********************************************************************/
BYTE GetReg(WORD index)
{
#ifdef USE_16BIT_PMP

    WORD    value;

    DisplaySetCommand(); // set RS line to low for command
    DisplayEnable();     // enable SSD1926

    DeviceWrite(((WORD_VAL) index).v[1]);
    DeviceWrite(index << 8);

    DisplaySetData();    // set RS line to high for data

    value = DeviceRead();
    value = DeviceRead();

    DisplayDisable();   // disable SSD1926

    if(index & 0x0001)
        value &= 0x00ff;
    else
        value = (value >> 8) & 0x00ff;

#else
    BYTE   value;

    DisplaySetCommand(); // set RS line to low for command
    DisplayEnable();     // enable SSD1926

    DeviceWrite(0x00);    // register access
    DeviceWrite(((WORD_VAL) index).v[1]);
    DeviceWrite(((WORD_VAL) index).v[0]);

    DisplaySetData();    // set RS line to high for data

    value = DeviceRead();
    value = DeviceRead();

    DisplayDisable();   // disable SSD1926
#endif

    return (value);
}

/*********************************************************************
* Function:  void DisplayBrightness(WORD level)
*
* PreCondition: none
*
* Input: level - Brightness level. Valid values are 0 to 100.
*           -   0: brightness level is zero or display is turned off
*           - 100: brightness level is maximum
*
* Output: none
*
* Side Effects: none
*
* Overview: Sets the brightness of the display.
*
* Note: none
*
*********************************************************************/
void DisplayBrightness(WORD level)
{
```

```
    // If the brightness can be controlled (for example through PWM)
    // add code that will control the PWM here.

    if (level > 0)
    {
        DisplayBacklightOn();
    }
    else if (level == 0)
    {
        DisplayBacklightOff();
    }

}

/*********************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*********************************************************************/
void ResetDevice(void)
{
    /*
        This is the reset routine of the display controller.
        ResetDevice() recommended sequence:
        1. Initialize the interface to the controller  - DriverInterfaceInit()
        2. Initialize the registers of the controller make the enabling of
           the display control last. In other words, the sync signals
           are enabled only after all the registers that needs to be set
           for the synchronization control are initialized.
        3. Before, the display is turned on, initialize the display buffer
           to a single color so you will not see a snowy image on the screen.
           This is done here using:
              SetColor(0);
              ClearDevice();
           where 0 is the black color.
        4. Enable the display controller refresh of the screen
           This is done here using:
              SetReg(REG_POWER_SAVE_CONFIG, 0x00);    //  wake up
        5. Initialize the Timing Controller (TCON) if needed.
        6. Enable the Backlight

        Most display panels will have the power-up sequence:
        1. Enable Analog Power Supply
        2. Enable Digital Power Supply - to start synchronization signals
        3. Enable Backlight - to show the contents of the display buffer
        For power-down, the reverse is followed.

    */

    ////////////////////////////////////////////////////////////////////
    // Initialize the device
    ////////////////////////////////////////////////////////////////////
    DriverInterfaceInit();

    // give time for the controller to power up
    DelayMs(250);

    ////////////////////////////////////////////////////////////////////
    // PLL SETUP
    // Crystal frequency x M / N = 80 MHz
    // for 4 MHz crystal:
```

```
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_PLL_CONFIG_0, 0x0a);         // set N = 10
    SetReg(REG_PLL_CONFIG_1, 0xc8);         // set M = 200
    SetReg(REG_PLL_CONFIG_2, 0xae);         // must be programmed to 0xAE
    SetReg(REG_PLL_CONFIG_0, 0x8a);         // enable PLL

    ////////////////////////////////////////////////////////////////////
    // VIDEO BUFFER MEMORY CLOCK SETUP
    // Memory frequency = PLL frequency / (MCLK + 1)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_MEMCLK_CONFIG, 0x00);        // set MCLK = 0 (80 MHz)

    ////////////////////////////////////////////////////////////////////

    // PIXEL OUTPUT CLOCK SETUP (LCD_SHIFT SIGNAL)
    // Pixel clock = Memory frequency * (PCLK + 1) / 0x100000
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_PCLK_FREQ_RATIO_0, 0x00);    // set PCLK = 0x020000
    SetReg(REG_PCLK_FREQ_RATIO_1, 0x00);    // Pixel clock = 10 MHz
    SetReg(REG_PCLK_FREQ_RATIO_2, 0x02);

    ////////////////////////////////////////////////////////////////////
    // Panel Configuration (reg 10h)
    // TFT display with 18 bit or 24-bit RGB parallel interface.
    ////////////////////////////////////////////////////////////////////
    BYTE panelType  = 0;
    BYTE panelWidth = 0;

    #if (DISP_DATA_WIDTH == 18)
        panelWidth |= 0x20;
    #elif (DISP_DATA_WIDTH == 24)
        panelWidth |= 0x30;
    #else
       #error "Define DISP_DATA_WIDTH in HardwareProfile.h (valid values: 18, 24)"
    #endif

    #if (GFX_LCD_TYPE == GFX_LCD_TFT)
        panelType |= 0x41;
    #elif (GFX_LCD_TYPE == GFX_LCD_CSTN)
        panelType |= 0x40;
    #elif (GFX_LCD_TYPE == GFX_LCD_MSTN)
        panelType |= 0x00;
    #else
       #error "Define GFX_LCD_TYPE in HardwareProfile.h (valid values: GFX_LCD_TFT,
GFX_LCD_CSTN, GFX_LCD_MSTN)"
    #endif

    SetReg(REG_PANEL_TYPE, panelType|panelWidth);

    ////////////////////////////////////////////////////////////////////
    // Horizontal total HT (reg 12h)
    ////////////////////////////////////////////////////////////////////
    #define HT  (DISP_HOR_PULSE_WIDTH + DISP_HOR_BACK_PORCH + DISP_HOR_FRONT_PORCH +
DISP_HOR_RESOLUTION)
    SetReg(REG_HORIZ_TOTAL_0, HT / 8);
    SetReg(REG_HORIZ_TOTAL_1, HT % 8);

    ////////////////////////////////////////////////////////////////////
    // Horizontal display period HDP (reg 14h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_HDP, DISP_HOR_RESOLUTION / 8 - 1);

    ////////////////////////////////////////////////////////////////////
    // Horizontal display period start HDPS (regs 16h, 17h)
    ////////////////////////////////////////////////////////////////////
    #define HDPS (DISP_HOR_PULSE_WIDTH + DISP_HOR_BACK_PORCH)
    SetReg(REG_HDP_START_POS0, HDPS & 0x00FF);
    SetReg(REG_HDP_START_POS1, (HDPS >> 8) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Horizontal syncronization pulse width HPW (reg 20h)
    ////////////////////////////////////////////////////////////////////
```

```
    SetReg(REG_HSYNC_PULSE_WIDTH, DISP_HOR_PULSE_WIDTH - 1);

    ////////////////////////////////////////////////////////////////////
    // Vertical total VT (regs 18h, 19h)
    ////////////////////////////////////////////////////////////////////
    #define VT  (DISP_VER_PULSE_WIDTH + DISP_VER_BACK_PORCH + DISP_VER_FRONT_PORCH +
DISP_VER_RESOLUTION)
    SetReg(REG_VERT_TOTAL0, VT & 0x00FF);
    SetReg(REG_VERT_TOTAL1, (VT >> 8) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Vertical display period VDP (regs 1ch, 1dh)
    ////////////////////////////////////////////////////////////////////
    #define VDP (DISP_VER_RESOLUTION - 1)
    SetReg(REG_VDP0, VDP & 0x00FF);
    SetReg(REG_VDP1, (VDP >> 8) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Vertical display period start VDPS (regs 1eh, 1fh)
    ////////////////////////////////////////////////////////////////////
    #define VDPS  (DISP_VER_PULSE_WIDTH + DISP_VER_BACK_PORCH)
    SetReg(REG_VDP_START_POS0, VDPS & 0x00FF);
    SetReg(REG_VDP_START_POS1, (VDPS >> 8) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Vertical syncronization pulse width VPW (reg 24h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_VSYNC_PULSE_WIDTH, DISP_VER_PULSE_WIDTH - 1);

    ////////////////////////////////////////////////////////////////////
    // PALETTE INIT
    #ifdef USE_PALETTE
    PaletteInit();
    #endif

    ////////////////////////////////////////////////////////////////////
    // ROTATION MODE
    #if (DISP_ORIENTATION == 0)
        #define WIN_START_ADDR  0ul
        #define ROTATION        0

    #elif (DISP_ORIENTATION == 90)
        #ifndef USE_PALETTE
            #define WIN_START_ADDR  ((((DWORD) GetMaxX() + 1) >> 1) - 1)
        #else
            #define WIN_START_ADDR  (((((DWORD) GetMaxX() + 1) * PaletteBpp) >> 5) - 1)
        #endif
        #define ROTATION    1

    #elif (DISP_ORIENTATION == 180)
        #ifndef USE_PALETTE
            #define WIN_START_ADDR  (((((DWORD) GetMaxX() + 1) * (GetMaxY() + 1)) >> 1) - 1)
        #else
            #define WIN_START_ADDR  (((((DWORD) GetMaxX() + 1) * (GetMaxY() + 1) *
PaletteBpp) >> 5) - 1)
        #endif
        #define ROTATION    2

    #elif (DISP_ORIENTATION == 270)
        #ifndef USE_PALETTE
            #define WIN_START_ADDR  ((((DWORD) GetMaxX() + 1) * GetMaxY())) >> 1)
        #else
            #define WIN_START_ADDR  ((((DWORD) GetMaxX() + 1) * GetMaxY() * PaletteBpp) >>
5)
        #endif
        #define ROTATION    3
    #endif

    ////////////////////////////////////////////////////////////////////
    // Special Effects Register (reg 71h)
    ////////////////////////////////////////////////////////////////////
    #ifndef USE_PALETTE
```

```
    SetReg(REG_SPECIAL_EFFECTS, 0x40 | ROTATION);
    #else
    SetReg(REG_SPECIAL_EFFECTS, 0x00 | ROTATION);
    #endif

    ////////////////////////////////////////////////////////////////////
    // Main Window Display Start Address (regs 74h, 75h, 76h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_MAIN_WIN_DISP_START_ADDR0, WIN_START_ADDR & 0x00FF);
    SetReg(REG_MAIN_WIN_DISP_START_ADDR1, (WIN_START_ADDR >> 8) & 0x00FF);
    SetReg(REG_MAIN_WIN_DISP_START_ADDR2, (WIN_START_ADDR >> 16) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Main Window Display Offset (regs 78h, 79h)
    ////////////////////////////////////////////////////////////////////
    #ifndef USE_PALETTE
        #define WIN_OFFSET  ((GetMaxX() + 1) >> 1)
    #else
        #define WIN_OFFSET  (((GetMaxX() + 1) * PaletteBpp) >> 5)
    #endif
    SetReg(REG_MAIN_WIN_ADDR_OFFSET0, WIN_OFFSET & 0x00FF);
    SetReg(REG_MAIN_WIN_ADDR_OFFSET1, (WIN_OFFSET >> 8) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Display Mode Register (reg 70h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_DISPLAY_MODE, 0x04);         // 16 BPP, enable RAM content to screen

    ////////////////////////////////////////////////////////////////////
    // RGB Settings Register (reg 1a4h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_RGB_SETTING, 0xc0);          // RGB format

    ////////////////////////////////////////////////////////////////////
    // LSHIFT Polarity Register (reg 38h)
    ////////////////////////////////////////////////////////////////////
    #ifdef DISP_INV_LSHIFT
    SetReg(REG_LSHIFT_POLARITY, 0x01);      // 1 means falling trigger
    #endif

    ////////////////////////////////////////////////////////////////////
    // Clear the display buffer with all zeros so the display will not
    // show garbage data when initially powered up
    ////////////////////////////////////////////////////////////////////
    SetColor(0);
    ClearDevice();

    ////////////////////////////////////////////////////////////////////
    // Power Saving Configuration Register (reg a0h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_POWER_SAVE_CONFIG, 0x00);    //  wake up

    // check until the controller is really awake
    while(GetReg(REG_POWER_SAVE_CONFIG) == 0x00);

    ////////////////////////////////////////////////////////////////////
    // LCD Power Control Register (reg adh)
    // If LCD_POWER is connected to the glass DISPON or RESET
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_GPIO_STATUS_CONTROL1, 0x80); // release the glass from reset

    ////////////////////////////////////////////////////////////////////
    // Panel TCON Programming - set up the TCON first before turning
    // on the display.
    ////////////////////////////////////////////////////////////////////
#ifdef USE_TCON_MODULE
    GfxTconInit();
#endif
    ////////////////////////////////////////////////////////////////////
    // Turn on the backlight
    ////////////////////////////////////////////////////////////////////
    DisplayBacklightOn();
```

```
}

/*****************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel
*
* Note: none
*
*****************************************************************/
void PutPixel(SHORT x, SHORT y)
{
    DWORD   address;

    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    #ifndef USE_PALETTE
    address = (((DWORD) (GetMaxX() + 1)) * y + x) << 1;
    #else
    address = ((((DWORD) (GetMaxX() + 1)) * y + x) * PaletteBpp) >> 3;
    #endif
    DisplayEnable();      // enable SSD1926
    SetAddress(address);
    WritePixel(_color);
    DisplayDisable();    // disable SSD1926
}

/*****************************************************************
* Function: WORD GetPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
* Output: pixel color
*
* Side Effects: none
*
* Overview: returns pixel color at x,y position
*
* Note: none
*
*****************************************************************/
WORD GetPixel(SHORT x, SHORT y)
{
    DWORD   address;

    address = (((DWORD) (GetMaxX() + 1)) * y + x) << 1;

    #ifdef USE_16BIT_PMP

    WORD    value;
```

```
    DisplayEnable();

    SetAddress(address);
    value = DeviceRead();
    value = DeviceRead();

    DisplayDisable();

    return (value);
    #else

    WORD_VAL    value;

    DisplayEnable();

    SetAddress(address);

    #if defined(USE_GFX_PMP)
        // this first two reads are a dummy reads
        value.Val = SingleDeviceRead();
        value.Val = SingleDeviceRead();
        // these are the reads that will get the actual pixel data
        value.v[1] = SingleDeviceRead();
        value.v[0] = SingleDeviceRead();
    #endif

    #if defined(USE_GFX_EPMP)
        value.Val = DeviceRead();
        value.v[1] = DeviceRead();
        value.v[0] = DeviceRead();
    #endif
    DisplayDisable();

    return (value.Val);
    #endif
}

#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

#ifndef USE_PALETTE
    // "In SSD1926 2D-Acceleration is not supported in Palette mode. Use Line function of
Primitive layer"

/*********************************************************************
* Function: WORD Line2D(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
*
* PreCondition: none
*
* Input: x1,y1 - starting coordinates, x2,y2 - ending coordinates
*
* Output: 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.
*
* Side Effects: none
*
* Overview: draws solid line
*
* Note: none
*
*********************************************************************/
static WORD Line2D(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
{
        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
        #else
    if(IsDeviceBusy() != 0)
        return (0);
        #endif

    /* Line Boundaries */
    SetReg(REG_2D_1e4, x1 & 0xFF);
    SetReg(REG_2D_1e5, (x1 >> 8) & 0xFF);
    SetReg(REG_2D_1e8, y1 & 0xFF);
    SetReg(REG_2D_1e9, (y1 >> 8) & 0xFF);
    SetReg(REG_2D_1ec, x2 & 0xFF);
    SetReg(REG_2D_1ed, (x2 >> 8) & 0xFF);
    SetReg(REG_2D_1f0, y2 & 0xFF);
    SetReg(REG_2D_1f1, (y2 >> 8) & 0xFF);

    /* Source & Destination Window Start Addresses */
    SetReg(REG_2D_1d4, 0);
    SetReg(REG_2D_1d5, 0);
    SetReg(REG_2D_1d6, 0);
    SetReg(REG_2D_1f4, 0);
    SetReg(REG_2D_1f5, 0);
    SetReg(REG_2D_1f6, 0);

    /* Display width */
    SetReg(REG_2D_1f8, (GetMaxX() + 1) & 0xFF);
    SetReg(REG_2D_1f9, ((GetMaxX() + 1) >> 8) & 0xFF);

    /* Display 2d width */
    SetReg(REG_2D_1d8, (GetMaxY() + 1) & 0xFF);
    SetReg(REG_2D_1d9, ((GetMaxY() + 1) >> 8) & 0xFF);

    /* Set Color */
    SetReg(REG_2D_1fe, RED8(_color));
    SetReg(REG_2D_1fd, GREEN8(_color));
    SetReg(REG_2D_1fc, BLUE8(_color));

    /* 16bpp */
    SetReg(REG_2D_1dd, 0x00);

    /* Line command */
    SetReg(REG_2D_1d1, 0x01);

    /* Draw2d command */
    SetReg(REG_2D_1d2, 0x01);

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
        #endif
    return (1);
}

/*********************************************************************
```

```
 * Function: WORD Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
 *
 * PreCondition: none
 *
 * Input: x1,y1 - starting coordinates, x2,y2 - ending coordinates
 *
 * Output: 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.
 *
 * Side Effects: none
 *
 * Overview: draws line
 *
 * Note: none
 *
 ********************************************************************/
WORD Line(SHORT x1, SHORT y1, SHORT x2, SHORT y2)
{

    SHORT   deltaX, deltaY;
    SHORT   error, stepErrorLT, stepErrorGE;
    SHORT   stepX, stepY;
    SHORT   steep;
    SHORT   temp;
    SHORT   style, type;

    stepX = 0;
    deltaX = x2 - x1;
    if(deltaX < 0)
    {
        deltaX = -deltaX;
        --stepX;
    }
    else
    {
        ++stepX;
    }

    stepY = 0;
    deltaY = y2 - y1;
    if(deltaY < 0)
    {
        deltaY = -deltaY;
        --stepY;
    }
    else
    {
        ++stepY;
    }

    steep = 0;
    if(deltaX < deltaY)
    {
        ++steep;
    }

        #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
        #else
    if(IsDeviceBusy() != 0)
        return (0);
        #endif
    if(_lineType == 0)
    {
        if(!Line2D(x1, y1, x2, y2))
            return (0);
        if(_lineThickness)
```

```
        {
            if(steep)
            {
                while(!Line2D(x1 + 1, y1, x2 + 1, y2));
                while(!Line2D(x1 - 1, y1, x2 - 1, y2));
            }
            else
            {
                while(!Line2D(x1, y1 + 1, x2, y2 + 1));
                while(!Line2D(x1, y1 - 1, x2, y2 - 1));
            }
        }

        // Move cursor
        MoveTo(x2, y2);
        return (1);
    }

    // Move cursor
    MoveTo(x2, y2);

    if(x1 == x2)
    {
        if(y1 > y2)
        {
            temp = y1;
            y1 = y2;
            y2 = temp;
        }

        style = 0;
        type = 1;
        for(temp = y1; temp < y2 + 1; temp++)
        {
            if((++style) == _lineType)
            {
                type ^= 1;
                style = 0;
            }

            if(type)
            {
                PutPixel(x1, temp);
                if(_lineThickness)
                {
                    PutPixel(x1 + 1, temp);
                    PutPixel(x1 - 1, temp);
                }
            }
        }

        return (1);
    }

    if(y1 == y2)
    {
        if(x1 > x2)
        {
            temp = x1;
            x1 = x2;
            x2 = temp;
        }

        style = 0;
        type = 1;
        for(temp = x1; temp < x2 + 1; temp++)
        {
            if((++style) == _lineType)
            {
                type ^= 1;
                style = 0;
            }
```

```
            if(type)
            {
                PutPixel(temp, y1);
                if(_lineThickness)
                {
                    PutPixel(temp, y1 + 1);
                    PutPixel(temp, y1 - 1);
                }
            }
        }

        return (1);
    }

    if(deltaX < deltaY)
    {
        temp = deltaX;
        deltaX = deltaY;
        deltaY = temp;
        temp = x1;
        x1 = y1;
        y1 = temp;
        temp = stepX;
        stepX = stepY;
        stepY = temp;
        PutPixel(y1, x1);
    }
    else
    {
        PutPixel(x1, y1);
    }

    // If the current error greater or equal zero
    stepErrorGE = deltaX << 1;

    // If the current error less than zero
    stepErrorLT = deltaY << 1;

    // Error for the first pixel
    error = stepErrorLT - deltaX;

    style = 0;
    type = 1;

    while(--deltaX >= 0)
    {
        if(error >= 0)
        {
            y1 += stepY;
            error -= stepErrorGE;
        }

        x1 += stepX;
        error += stepErrorLT;

        if((++style) == _lineType)
        {
            type ^= 1;
            style = 0;
        }

        if(type)
        {
            if(steep)
            {
                PutPixel(y1, x1);
                if(_lineThickness)
                {
                    PutPixel(y1 + 1, x1);
                    PutPixel(y1 - 1, x1);
                }
```

```
            }
            else
            {
                PutPixel(x1, y1);
                if(_lineThickness)
                {
                    PutPixel(x1, y1 + 1);
                    PutPixel(x1, y1 - 1);
                }
            }
        }
    }   // end of while

    return (1);
}

#endif // #ifndef USE_PALETTE

/*****************************************************************
* Function: WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
*
* PreCondition: none
*
* Input: left,top - top left corner coordinates,
*        right,bottom - bottom right corner coordinates
*
* Output: 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.
*
* Side Effects: none
*
* Overview: draws rectangle filled with current color
*
* Note: none
*
*****************************************************************/
WORD Bar(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
        #ifdef USE_PALETTE

    DWORD           address;
    register SHORT  x, y;

    if(_clipRgn)
    {
        left   = (left   < _clipLeft)   ? _clipLeft   : ((left   > _clipRight)  ?
_clipRight  : left);
        right  = (right  > _clipRight)  ? _clipRight  : ((right  < _clipLeft)   ?
_clipLeft   : right);
        top    = (top    < _clipTop)    ? _clipTop    : ((top    > _clipBottom) ?
_clipBottom : top);
        bottom = (bottom > _clipBottom) ? _clipBottom : ((bottom < _clipTop)    ?
_clipTop    : bottom);
    }

            #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
            #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
            #endif
    DisplayEnable();
    for(y = top; y < bottom + 1; y++)
    {
        SetAddress(address);
        for(x = left; x < right + 1; x++)
        {
            WritePixel(_color);
        }
```

```
                #ifndef USE_PALETTE
        address += (GetMaxX() + 1) << 1;
                #else
        address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
                #endif
    }

    DisplayDisable();
    return (1);

        #else

    DWORD   address;
    SHORT   width, height;

            #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
            #else
    if(IsDeviceBusy() != 0)
        return (0);
            #endif
    if(left > right)
    {
        return (1); /* Don't draw but return 1 */
    }

    if(top > bottom)
    {
        return (1); /* Don't draw but return 1 */
    }

    if(_clipRgn)
    {
        left   = (left   < _clipLeft)   ? _clipLeft   : ((left   > _clipRight)  ?
_clipRight  : left);
        right  = (right  > _clipRight)  ? _clipRight  : ((right  < _clipLeft)   ?
_clipLeft   : right);
        top    = (top    < _clipTop)    ? _clipTop    : ((top    > _clipBottom) ?
_clipBottom : top);
        bottom = (bottom > _clipBottom) ? _clipBottom : ((bottom < _clipTop)    ?
_clipTop    : bottom);
    }

    width = right - left + 1;
    height = bottom - top + 1;

    address = top * (GetMaxX() + (DWORD) 1) + left;

    PutPixel(left, top);

    /* Source, Destination & Brush Window Start Addresses */
    SetReg(REG_2D_1d4, address & 0xFF);
    SetReg(REG_2D_1d5, (address >> 8) & 0xFF);
    SetReg(REG_2D_1d6, (address >> 16) & 0xFF);
    SetReg(REG_2D_1f4, address & 0xFF);
    SetReg(REG_2D_1f5, (address >> 8) & 0xFF);
    SetReg(REG_2D_1f6, (address >> 16) & 0xFF);
    SetReg(REG_2D_204, address & 0xFF);
    SetReg(REG_2D_205, (address >> 8) & 0xFF);
    SetReg(REG_2D_206, (address >> 16) & 0xFF);

    /* Source & Destination Window Width */
    SetReg(REG_2D_1ec, width & 0xFF);
    SetReg(REG_2D_1ed, (width >> 8) & 0xFF);
    SetReg(REG_2D_1e4, width & 0xFF);
    SetReg(REG_2D_1e5, (width >> 8) & 0xFF);

    /* Source & Destination Window Height */
    SetReg(REG_2D_1f0, height & 0xFF);
    SetReg(REG_2D_1f1, (height >> 8) & 0xFF);
```

```
    SetReg(REG_2D_1e8, height & 0xFF);
    SetReg(REG_2D_1e9, (height >> 8) & 0xFF);

    /* Brush width */
    SetReg(REG_2D_214, 1);
    SetReg(REG_2D_215, 0);

    /* Brush height */
    SetReg(REG_2D_218, 1);
    SetReg(REG_2D_219, 0);

    /* Display width */
    SetReg(REG_2D_1f8, (GetMaxX() + 1) & 0xFF);
    SetReg(REG_2D_1f9, ((GetMaxX() + 1) >> 8) & 0xFF);

    /* Display 2d width */
    SetReg(REG_2D_1d8, (GetMaxX() + 1) & 0xFF);
    SetReg(REG_2D_1d9, ((GetMaxX() + 1) >> 8) & 0xFF);

    /* ROP3 Command */
    SetReg(REG_2D_1fc, 0xF0);

    /* 16bpp */
    SetReg(REG_2D_1dd, 0x00);

    /* ROP command */
    SetReg(REG_2D_1d1, 0x09);

    /* Draw2d command */
    SetReg(REG_2D_1d2, 0x01);

            #ifndef USE_NONBLOCKING_CONFIG
    while(IsDeviceBusy() != 0);

    /* Ready */
            #endif
    return (1);
        #endif
}

/*********************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: clears screen with current color
*
* Note: none
*
*********************************************************************/
void ClearDevice(void)
{
        #ifdef USE_PALETTE

    DWORD   counter;

    DisplayEnable();
    SetAddress(0);
    SetAddress(0);
    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1); counter++)
    {
        WritePixel(_color);
    }

    DisplayDisable();
```

```
        #else
    while(GetReg(REG_2D_220) == 0);
    while(!Bar(0, 0, GetMaxX(), GetMaxY()));
    while(GetReg(REG_2D_220) == 0);

    /* Ready */
        #endif
}


#ifdef USE_BITMAP_FLASH
/*********************************************************************
* Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner,
*        image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*********************************************************************/
void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp = 0, stretchY, mask;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                pallete[2];

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[0] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    pallete[1] = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    DisplayEnable();      // enable SSD1926

   // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            SetAddress(address);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {
```

```
                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == 1) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(1);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[1]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(pallete[1]);
                    }
                }
                else
                {
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == 0) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(0);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[0]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(pallete[0]);
                    }
                }
                // shift to the next pixel
                mask >>= 1;
            }

            #ifndef USE_PALETTE
                address += (GetMaxX() + 1) << 1;
            #else
                address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
            #endif
        }
    }
    else
```

```
    {
        for(y = 0; y < sizeY; y++)
        {
            tempFlashAddress = flashAddress;
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                flashAddress = tempFlashAddress;
                SetAddress(address);
                mask = 0;
                for(x = 0; x < sizeX; x++)
                {
                    // Read 8 pixels from flash
                    if(mask == 0)
                    {
                        temp = *flashAddress;
                        flashAddress++;
                        mask = 0x80;
                    }

                    // Set color
                    if(mask & temp)
                    {
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == 1) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(1);
                                WritePixel(1);
                            }

                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == pallete[1]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(pallete[1]);
                                WritePixel(pallete[1]);
                            }
                        }
                    }
                    else
                    {
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == 0) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
```

```
                                {
                                    WritePixel(0);
                                    WritePixel(0);
                                }
                            }
                            else
                            #endif
                            {
                                #ifdef USE_TRANSPARENT_COLOR
                                if ((GetTransparentColor() == pallete[0]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                                {
                                    DeviceReadWord();
                                    DeviceReadWord();
                                }
                                else
                                #endif
                                {
                                    WritePixel(pallete[0]);
                                    WritePixel(pallete[0]);
                                }
                            }
                        }
                        // shift to the next pixel
                        mask >>= 1;
                    }
                    #ifndef USE_PALETTE
                        address += (GetMaxX() + 1) << 1;
                    #else
                        address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
                    #endif
                }
            }
        }
    DisplayDisable();
}

/*****************************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs 16 color image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************************/
void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp = 0, stretchY;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                pallete[16];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
```

```
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();      // enable SSD1926

   // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {
                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == (temp >> 4)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                        }
                        else
                        #endif
                            WritePixel(temp >> 4);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[temp >> 4]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                        }
                        else
                        #endif
                            WritePixel(pallete[temp >> 4]);
                    }
                }
                else
                {
                    temp = *flashAddress++;

                    // first pixel in byte
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == (temp & 0x0f)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        {
```

```
                            DeviceReadWord();
                        }
                        else
                        #endif
                            WritePixel(temp & 0x0f);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[temp & 0x0f]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                        }
                        else
                        #endif
                            WritePixel(pallete[temp & 0x0f]);
                    }
                }
            }
            #ifndef USE_PALETTE
                address += (GetMaxX() + 1) << 1;
            #else
                address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
            #endif
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            tempFlashAddress = flashAddress;
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                flashAddress = tempFlashAddress;
                SetAddress(address);

                for(x = 0; x < sizeX; x++)
                {
                    // Read 2 pixels from flash
                    if(x & 0x0001)
                    {

                        // second pixel in byte
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == (temp >> 4)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(temp >> 4);
                                WritePixel(temp >> 4);
                            }
                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == pallete[temp >> 4]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
```

```
                            else
                            #endif
                            {
                                WritePixel(pallete[temp >> 4]);
                                WritePixel(pallete[temp >> 4]);
                            }
                        }
                    }
                    else
                    {
                        temp = *flashAddress++;

                        // first pixel in byte
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == (temp & 0x0f)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(temp & 0x0f);
                                WritePixel(temp & 0x0f);
                            }
                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == pallete[temp & 0x0f]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                           }
                            else
                            #endif
                            {
                                WritePixel(pallete[temp & 0x0f]);
                                WritePixel(pallete[temp & 0x0f]);
                            }
                        }
                    }
                }
                #ifndef USE_PALETTE
                    address += (GetMaxX() + 1) << 1;
                #else
                    address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
                #endif
            }
        }
    }

    DisplayDisable();
}

/*****************************************************************************
* Function: void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
```

```
 *
 * Overview: outputs 256 color image starting from left,top coordinates
 *
 * Note: image must be located in flash
 *
 ********************************************************************/
void PutImage8BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp, stretchY;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                pallete[256];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 256; counter++)
    {
        pallete[counter] = *((FLASH_WORD *)flashAddress);
        flashAddress += 2;
    }

    DisplayEnable();      // enable SSD1926

    // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {
                temp = *flashAddress;
                flashAddress++;

                // Write pixel to screen
                #ifdef USE_PALETTE
                if(IsPaletteEnabled())
                {
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == temp) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                    {
                        DeviceReadWord();
                    }
                    else
                    #endif
                        WritePixel(temp);
                }
                else
                #endif
                {
```

```
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == pallete[temp]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                    {
                        DeviceReadWord();
                    }
                    else
                    #endif
                        WritePixel(pallete[temp]);
                }

            }
            #ifndef USE_PALETTE
                address += (GetMaxX() + 1) << 1;
            #else
                address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
            #endif
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            tempFlashAddress = flashAddress;
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                flashAddress = tempFlashAddress;
                SetAddress(address);

                for(x = 0; x < sizeX; x++)
                {
                    temp = *flashAddress;
                    flashAddress++;

                   // Write pixel to screen
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == temp) && (GetTransparentColorStatus()
== TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                            DeviceReadWord();
                        }
                        else
                        #endif
                        {
                            WritePixel(temp);
                            WritePixel(temp);
                        }
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[temp]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                            DeviceReadWord();
                        }
                        else
                        #endif
                        {
                            WritePixel(pallete[temp]);
                            WritePixel(pallete[temp]);
                        }
                    }

                }
                #ifndef USE_PALETTE
```

```
                    address += (GetMaxX() + 1) << 1;
                #else
                    address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
                #endif
            }
        }
    }

    DisplayDisable();
}

/*****************************************************************
* Function: void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs hicolor image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************/
void PutImage16BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register DWORD      address;
    register FLASH_WORD *flashAddress;
    register FLASH_WORD *tempFlashAddress;
    WORD                temp, stretchY;
    WORD                sizeX, sizeY;
    WORD                x, y;

    // Move pointer to size information
    flashAddress = (FLASH_WORD *)image + 1;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Read image size
    sizeY = *flashAddress;
    flashAddress++;
    sizeX = *flashAddress;
    flashAddress++;

    DisplayEnable();      // enable SSD1926

    // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {
            SetAddress(address);
            for(x = 0; x < sizeX; x++)
            {
                temp = *flashAddress;
                flashAddress++;

                // Write pixel to screen
                #ifdef USE_PALETTE
                if(IsPaletteEnabled())
                {
```

```
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == temp) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                    {
                        DeviceReadWord();
                    }
                    else
                    #endif
                        WritePixel(0); /* Paint first value of Palette instead of junk */
                }
                else
                #endif
                {
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == temp) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                    {
                        DeviceReadWord();
                    }
                    else
                    #endif
                        WritePixel(temp);
                }

            }
            #ifndef USE_PALETTE
                address += (GetMaxX() + 1) << 1;
            #else
                address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
            #endif
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            tempFlashAddress = flashAddress;
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                flashAddress = tempFlashAddress;
                SetAddress(address);

                for(x = 0; x < sizeX; x++)
                {
                    temp = *flashAddress;
                    flashAddress++;

                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == temp) && (GetTransparentColorStatus()
== TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                            DeviceReadWord();
                        }
                        else
                        #endif
                        {
                            WritePixel(0); /* Paint first value of Palette instead of junk
*/
                            WritePixel(0);
                        }
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == temp) && (GetTransparentColorStatus()
== TRANSPARENT_COLOR_ENABLE))
                        {
```

```
                            DeviceReadWord();
                            DeviceReadWord();
                        }
                        else
                        #endif
                        {
                            WritePixel(temp);
                            WritePixel(temp);
                        }
                    }
                }
                #ifndef USE_PALETTE
                    address += (GetMaxX() + 1) << 1;
                #else
                    address += ((GetMaxX() + 1) * PaletteBpp) >> 3;
                #endif
            }
        }
    }

    DisplayDisable();
}

#endif
#ifdef USE_BITMAP_EXTERNAL

/*********************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*********************************************************************/
void PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[2];
    BYTE            lineBuffer[((GetMaxX() + 1) / 8) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp = 0, stretchY;
    BYTE            mask;
    WORD            sizeX, sizeY;
    WORD            x, y;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), pallete);

    // Set offset to the image data
```

```
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {

            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;
            DisplayEnable();      // enable SSD1926

            pData = lineBuffer;
            SetAddress(address);
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *pData++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == 1) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(1);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[1]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(pallete[1]);
                    }
                }
                else
                {
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == 0) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
```

```
                        else
                        #endif
                            WritePixel(0);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[0]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(pallete[0]);
                    }
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            address += (GetMaxX() + 1) << 1;
            DisplayDisable();
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;
            DisplayEnable();      // enable SSD1926
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                pData = lineBuffer;
                SetAddress(address);
                mask = 0;
                for(x = 0; x < sizeX; x++)
                {

                    // Read 8 pixels from flash
                    if(mask == 0)
                    {
                        temp = *pData++;
                        mask = 0x80;
                    }

                    // Set color
                    if(mask & temp)
                    {
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == 1) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(1);
                                WritePixel(1);
                            }
                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
```

```
                            if ((GetTransparentColor() == pallete[1]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(pallete[1]);
                                WritePixel(pallete[1]);
                            }
                        }
                    }
                    else
                    {
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == 0) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(0);
                                WritePixel(0);
                            }
                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == pallete[0]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(pallete[0]);
                                WritePixel(pallete[0]);
                            }
                        }
                    }

                    // Shift to the next pixel
                    mask >>= 1;
                }
                address += (GetMaxX() + 1) << 1;
            }
            DisplayDisable();
        }
    }
}

/*****************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
```

```
 * Side Effects: none
 *
 * Overview: outputs monochrome image starting from left,top coordinates
 *
 * Note: image must be located in external memory
 *
 ********************************************************************/
void PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[16];
    BYTE            lineBuffer[((GetMaxX() + 1) / 2) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp = 0, stretchY;
    WORD            sizeX, sizeY;
    WORD            x, y;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {

            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;
            DisplayEnable();      // enable SSD1926

            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {
                // Read 2 pixels from flash
                if(x & 0x0001)
                {
                    // second pixel in byte
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
```

```
                        if ((GetTransparentColor() == (temp >> 4)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(temp >> 4);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[temp >> 4]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(pallete[temp >> 4]);
                    }
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == (temp & 0x0f)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(temp & 0x0f);
                    }
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[temp & 0x0f]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            DeviceReadWord();
                        else
                        #endif
                            WritePixel(pallete[temp & 0x0f]);
                    }
                }
            }

            address += (GetMaxX() + 1) << 1;
            DisplayDisable();
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;
            DisplayEnable();      // enable SSD1926
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                pData = lineBuffer;
                SetAddress(address);

                for(x = 0; x < sizeX; x++)
                {

                    // Read 2 pixels from flash
                    if(x & 0x0001)
                    {
```

```
                        // second pixel in byte
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == (temp >> 4)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(temp >> 4);
                                WritePixel(temp >> 4);
                            }
                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == pallete[temp >> 4]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(pallete[temp >> 4]);
                                WritePixel(pallete[temp >> 4]);
                            }
                        }
                    }
                    else
                    {
                        temp = *pData++;

                        // first pixel in byte
                        #ifdef USE_PALETTE
                        if(IsPaletteEnabled())
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == (temp & 0x0f)) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(temp & 0x0f);
                                WritePixel(temp & 0x0f);
                            }
                        }
                        else
                        #endif
                        {
                            #ifdef USE_TRANSPARENT_COLOR
                            if ((GetTransparentColor() == pallete[temp & 0x0f]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                            {
                                DeviceReadWord();
                                DeviceReadWord();
                            }
                            else
                            #endif
                            {
                                WritePixel(pallete[temp & 0x0f]);
```

```
                                WritePixel(pallete[temp & 0x0f]);
                            }
                        }
                    }
                }
                address += (GetMaxX() + 1) << 1;
            }
            DisplayDisable();
        }
    }
}

/*****************************************************************************
* Function: void PutImage8BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************************/
void PutImage8BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[256];
    BYTE            lineBuffer[(GetMaxX() + 1)];
    BYTE            *pData;

    BYTE            temp, stretchY;
    WORD            sizeX, sizeY;
    WORD            x, y;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (256 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 256 * sizeof(WORD), pallete);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 256 * sizeof(WORD);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {

            // Get line
            ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
```

```
            memOffset += sizeX;
            DisplayEnable();      // enable SSD1926

            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;

                #ifdef USE_PALETTE
                if(IsPaletteEnabled())
                {
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == temp) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                        DeviceReadWord();
                    else
                    #endif
                        WritePixel(temp);
                }
                else
                #endif
                {
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == pallete[temp]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        DeviceReadWord();
                    else
                    #endif
                        WritePixel(pallete[temp]);
                }
            }

            address += (GetMaxX() + 1) << 1;
            DisplayDisable();
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, sizeX, lineBuffer);
            memOffset += sizeX;
            DisplayEnable();      // enable SSD1926
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                pData = lineBuffer;
                SetAddress(address);

                for(x = 0; x < sizeX; x++)
                {
                    temp = *pData++;

                    #ifdef USE_PALETTE
                    if(IsPaletteEnabled())
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == temp) && (GetTransparentColorStatus()
== TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                            DeviceReadWord();
                        }
                        else
                        #endif
                        {
                            WritePixel(temp);
                            WritePixel(temp);
                        }
                    }
```

```
                    else
                    #endif
                    {
                        #ifdef USE_TRANSPARENT_COLOR
                        if ((GetTransparentColor() == pallete[temp]) &&
(GetTransparentColorStatus() == TRANSPARENT_COLOR_ENABLE))
                        {
                            DeviceReadWord();
                            DeviceReadWord();
                        }
                        else
                        #endif
                        {
                            WritePixel(pallete[temp]);
                            WritePixel(pallete[temp]);
                        }
                    }

                }
                address += (GetMaxX() + 1) << 1;
            }
            DisplayDisable();
        }
    }
}

/*****************************************************************
* Function: void PutImage16BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************/
void PutImage16BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  address;
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            lineBuffer[(GetMaxX() + 1)];
    WORD            *pData;
    WORD            byteWidth;

    WORD            temp, stretchY;
    WORD            sizeX, sizeY;
    WORD            x, y;

    // Set start address
        #ifndef USE_PALETTE
    address = ((DWORD) (GetMaxX() + 1) * top + left) << 1;
        #else
    address = (((DWORD) (GetMaxX() + 1) * top + left) * PaletteBpp) >> 3;
        #endif

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER);

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;
```

```
    byteWidth = sizeX << 1;

    // Note: For speed the code for loops are repeated. A small code size increase for
performance

    if (stretch == IMAGE_NORMAL)
    {
        for(y = 0; y < sizeY; y++)
        {

            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;
            DisplayEnable();      // enable SSD1926

            pData = lineBuffer;
            SetAddress(address);

            for(x = 0; x < sizeX; x++)
            {
                temp = *pData++;

                // Write pixel to screen
                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == temp) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                        DeviceReadWord();
                    else
                    #endif
                        WritePixel(temp);

            }

            address += (GetMaxX() + 1) << 1;
            DisplayDisable();
        }
    }
    else
    {
        for(y = 0; y < sizeY; y++)
        {
            // Get line
            ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
            memOffset += byteWidth;
            DisplayEnable();      // enable SSD1926
            for(stretchY = 0; stretchY < stretch; stretchY++)
            {
                pData = lineBuffer;
                SetAddress(address);

                for(x = 0; x < sizeX; x++)
                {
                    temp = *pData++;

                    #ifdef USE_TRANSPARENT_COLOR
                    if ((GetTransparentColor() == temp) && (GetTransparentColorStatus() ==
TRANSPARENT_COLOR_ENABLE))
                    {
                        DeviceReadWord();
                        DeviceReadWord();
                    }
                    else
                    #endif
                    {
                        WritePixel(temp);
                        WritePixel(temp);
                    }
                }
                address += (GetMaxX() + 1) << 1;
            }
            DisplayDisable();
```

```
        }
    }
}
#endif

/*****************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*****************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*****************************************************************/
void SetClip(BYTE control)
{
    _clipRgn=control;
}

/*****************************************************************
* Function: IsDeviceBusy()
*
* Overview: Returns non-zero if LCD controller is busy
*           (previous drawing operation is not completed).
*
* PreCondition: none
*
* Input: none
*
* Output: Busy status.
*
* Side Effects: none
*
*****************************************************************/
WORD IsDeviceBusy(void)
{
    return ((GetReg(REG_2D_220) & 0x01) == 0);
}
```

```
#ifdef USE_PALETTE

/*********************************************************************
* Function: void PaletteInit(void)
*
* Overview: Initializes the CLUT.
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: Drawing mode will change to support palettes
*
*********************************************************************/
void PaletteInit(void)
{
    PaletteBpp = 16;
    blPaletteChangeError = 0;
    pPendingPalette = NULL;
    PendingStartEntry = 0;
    PendingLength = 0;
}

/*********************************************************************
* Function: BYTE SetPaletteBpp(BYTE bpp)
*
* Overview: Sets the CLUT's number of valid entries.
*
* PreCondition: PaletteInit() must be called before.
*
* Input: bpp -> Bits per pixel
*
* Output: Status: Zero -> Success, Non-zero -> Error.
*
* Side Effects: Drawing mode will change to support palettes
*
*********************************************************************/
BYTE SetPaletteBpp(BYTE bpp)
{
    switch(bpp)
    {
        /*case 1: SetReg(REG_DISPLAY_MODE,0x00);
                break;

        case 2: SetReg(REG_DISPLAY_MODE,0x01);
                break;

        case 4: SetReg(REG_DISPLAY_MODE,0x02);
                break;*/
        case 8:     SetReg(REG_DISPLAY_MODE, 0x03); break;
        case 16:    SetReg(REG_DISPLAY_MODE, 0x04); break;
        default:    SetReg(REG_DISPLAY_MODE, 0x04); PaletteBpp = 16; return (-1);
    }

    PaletteBpp = bpp;

    ////////////////////////////////////////////////////////////////////
    // ROTATION MODE
        ////////////////////////////////////////////////////////////////////
        #if (DISP_ORIENTATION == 0)
            #define WIN_START_ADDR  0ul
            #define ROTATION        0

        #elif (DISP_ORIENTATION == 90)
            #ifndef USE_PALETTE
                #define WIN_START_ADDR  ((((DWORD) GetMaxX() + 1) >> 1) - 1)
            #else
                #define WIN_START_ADDR  (((((DWORD) GetMaxX() + 1) * PaletteBpp) >> 5) - 1)
            #endif
            #define ROTATION    1
```

```
        #elif (DISP_ORIENTATION == 180)
            #ifndef USE_PALETTE
                #define WIN_START_ADDR  (((((DWORD) GetMaxX() + 1) * (GetMaxY() + 1)) >> 1)
- 1)
            #else
                #define WIN_START_ADDR  (((((DWORD) GetMaxX() + 1) * (GetMaxY() + 1) *
PaletteBpp) >> 5) - 1)
            #endif
            #define ROTATION    2

        #elif (DISP_ORIENTATION == 270)
            #ifndef USE_PALETTE
                #define WIN_START_ADDR  ((((DWORD) GetMaxX() + 1) * GetMaxY()) >> 1)
            #else
                #define WIN_START_ADDR  ((((DWORD) GetMaxX() + 1) * GetMaxY() * PaletteBpp)
>> 5)
            #endif
            #define ROTATION    3
        #endif

    ////////////////////////////////////////////////////////////////////
    // Special Effects Register (reg 71h)
    ////////////////////////////////////////////////////////////////////
        #ifndef USE_PALETTE
    SetReg(REG_SPECIAL_EFFECTS, 0x40 | ROTATION);
        #else
    SetReg(REG_SPECIAL_EFFECTS, 0x00 | ROTATION);
        #endif

    ////////////////////////////////////////////////////////////////////
    // Main Window Display Start Address (regs 74h, 75h, 76h)
    ////////////////////////////////////////////////////////////////////
    SetReg(REG_MAIN_WIN_DISP_START_ADDR0, WIN_START_ADDR & 0x00FF);
    SetReg(REG_MAIN_WIN_DISP_START_ADDR1, (WIN_START_ADDR >> 8) & 0x00FF);
    SetReg(REG_MAIN_WIN_DISP_START_ADDR2, (WIN_START_ADDR >> 16) & 0x00FF);

    ////////////////////////////////////////////////////////////////////
    // Main Window Display Offset (regs 78h, 79h)
    ////////////////////////////////////////////////////////////////////
        #ifndef USE_PALETTE
            #define WIN_OFFSET  ((GetMaxX() + 1) >> 1)
        #else
            #define WIN_OFFSET  (((GetMaxX() + 1) * PaletteBpp) >> 5)
        #endif
    SetReg(REG_MAIN_WIN_ADDR_OFFSET0, WIN_OFFSET & 0x00FF);
    SetReg(REG_MAIN_WIN_ADDR_OFFSET1, (WIN_OFFSET >> 8) & 0x00FF);

    return (0);
}

/*******************************************************************
* Function: void EnablePalette(void)
*
* Overview: Enables the Palette mode
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects:
*
*******************************************************************/
void EnablePalette(void)
{
    SetPaletteBpp(PaletteBpp);
}

/*******************************************************************
* Function: void DisablePalette(void)
```

```
 *
 * Overview: Disables the Palette mode
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: none
 *
 * Side Effects:
 *
 ********************************************************************/
void DisablePalette(void)
{
    SetPaletteBpp(16);
}

/********************************************************************
 * Function: BYTE IsPaletteEnabled(void)
 *
 * Overview: Returns if the Palette mode is enabled or not
 *
 * PreCondition: none
 *
 * Input: none
 *
 * Output: Enabled -> 1, Disabled -> 0
 *
 * Side Effects:
 *
 ********************************************************************/
BYTE IsPaletteEnabled(void)
{
    return ((PaletteBpp == 16) ? 0 : 1);
}

/********************************************************************
 * Function: void StartVBlankInterrupt(void)
 *
 * Overview: Sets up the Vertical Blanking Interrupt
 *
 * PreCondition: Interrupts must be enabled
 *
 * Input: none
 *
 * Output: none
 *
 * Side Effects: none
 *
 ********************************************************************/
void StartVBlankInterrupt(void)
{

    /* We don't use Vertical Blanking Interrupt in SSD1926 */
    if(pPendingPalette != NULL)
    {
        blPaletteChangeError = SetPalette(pPendingPalette, PendingStartEntry,
PendingLength);
        if(!blPaletteChangeError)
        {
            _palette = pPendingPalette;
        }
    }
}

/********************************************************************
 * Function: BYTE SetPaletteFlash(PALETTE_ENTRY *pPaletteEntry, WORD startEntry, WORD length)
 *
 * Overview: Loads the palettes from the flash immediately.
 *
 * PreCondition: PaletteInit() must be called before.
 *
```

```
* Input: pPaletteEntry   - Pointer to the palette table in ROM
*        startEntry      - Start entry to load (inclusive)
*        length          - Number of entries
*
* Output: Status: Zero -> Success, Non-zero -> Error.
*
* 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)
{
    WORD    counter;

    if((pPaletteEntry == NULL) || ((startEntry + length) > 256))
    {
        return (-1);
    }

    for(counter = 0; counter < length; counter++)
    {
        SetReg(REG_LUT_RED_WRITE_DATA, RED8(pPaletteEntry[counter].value));
        SetReg(REG_LUT_GREEN_WRITE_DATA, GREEN8(pPaletteEntry[counter].value));
        SetReg(REG_LUT_BLUE_WRITE_DATA, BLUE8(pPaletteEntry[counter].value));
        SetReg(REG_LUT_WRITE_ADDR, startEntry + counter);
    }

    return (0);
}

#endif

#endif // #if (GFX_USE_DISPLAY_CONTROLLER_SSD1926)
```

14.3.12 SSD1926.h

Functions

	Name	Description
	DisplayBrightness (see page 383)	Sets the brightness of the display.

Macros

Name	Description
GFX_LCD_CSTN (see page 39)	Type Color STN Display
GFX_LCD_MSTN (see page 39)	Type Mono STN Display
GFX_LCD_TFT (see page 39)	Type TFT Display

Description

This is file SSD1926.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Solomon Systech. SSD1926 LCD controllers driver
 *  to be used with GFX 3 PICtail board
 *****************************************************************************
 * FileName:        SSD1926.h
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 08/27/08     ...
 * 30/07/09     Added Palette Support
 * 03/03/10     Enabled accelerated Circle() function.
 * 02/14/11     - for Graphics Library Version 3.00
 *              - Removed USE_DRV_xxxx switches. This is not needed anymore
 *                since Primitive Lyer implements weak attributes on
 *                Primitive Routines that can be implemented in hardware.
 *              - Removed Circle Function implementation (SSD1926 bug)
 *              - Moved common APIs to DisplayDriver.h. DisplayDriver.h
 *                is now a common file that should be included in the
 *                the driver source code.
 *              - Added #define GFX_LCD_TFT  0x01 // Type TFT Display
 *                      #define GFX_LCD_CSTN 0x03 // Type Color STN Display
 *                      #define GFX_LCD_MSTN 0x02 // Type Mono STN Display
 *                to select type of display panel.
 ****************************************************************************/

#ifndef _SSD1926_H
    #define _SSD1926_H

    #include "HardwareProfile.h"
    #include "GenericTypeDefs.h"

    #ifdef USE_PALETTE
        #include "Graphics/Palette.h"
    #endif

/*********************************************************************
* Error Checking
*********************************************************************/
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* Overview: Horizontal synchronization timing in pixels
*                   (from the glass datasheet).
*********************************************************************/

    #ifndef DISP_HOR_PULSE_WIDTH
        #error DISP_HOR_PULSE_WIDTH must be defined in HardwareProfile.h
    #endif
```

```
    #ifndef DISP_HOR_BACK_PORCH
        #error DISP_HOR_BACK_PORCH must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_HOR_FRONT_PORCH
        #error DISP_HOR_FRONT_PORCH must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* Overview: Vertical synchronization timing in lines
*                   (from the glass datasheet).
*********************************************************************/
    #ifndef DISP_VER_PULSE_WIDTH
        #error DISP_VER_PULSE_WIDTH must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_BACK_PORCH
        #error DISP_VER_BACK_PORCH must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_FRONT_PORCH
        #error DISP_VER_FRONT_PORCH must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* PARAMETERS VALIDATION
*********************************************************************/
    #if COLOR_DEPTH != 16
        #error This driver supports 16 BPP only.
    #endif
    #if (DISP_HOR_RESOLUTION % 8) != 0
        #error Horizontal resolution must be divisible by 8.
    #endif
    #if (DISP_ORIENTATION != 0) && (DISP_ORIENTATION != 180) && (DISP_ORIENTATION != 90) &&
(DISP_ORIENTATION != 270)
        #error The display orientation selected is not supported. It can be only 0,90,180
or 270.
    #endif

/*********************************************************************
* Function:  void SetReg(WORD index, BYTE value);
*
* PreCondition: none
*
* Input: index - register number
*        value - data to be written to the register
*
* Output: none
*
* Side Effects: none
*
* Overview: Writes the given value to the register defined by index.
*
* Note: none
*
*********************************************************************/
void            SetReg(WORD index, BYTE value);


/*********************************************************************
* Function:  BYTE GetReg(WORD index)
*
* PreCondition: none
*
* Input: index - register number
*
* Output: none
*
* Side Effects: none
*
* Overview: returns graphics controller register value (byte access)
*
* Note: none
*
*********************************************************************/
```

```
BYTE            GetReg(WORD index);

/*****************************************************************
* Function:  void DisplayBrightness(WORD level)
*
* PreCondition: none
*
* Input: level - Brightness level. Valid values are 0 to 100.
*           - 0: brightness level is zero or display is turned off
*           - 1: brightness level is maximum
*
* Output: none
*
* Side Effects: none
*
* Overview: Sets the brightness of the display.
*
* Note: none
*
*****************************************************************/
void            DisplayBrightness(WORD level);

/*****************************************************************
* Overview: SSD1926 registers definitions.
*****************************************************************/
    #define REG_PLL_CONFIG_0                    0x126
    #define REG_PLL_CONFIG_1                    0x127
    #define REG_PLL_CONFIG_2                    0x12b

    #define REG_DISP_BUFFER_SIZE                0x01
    #define REG_CONFIG_READBACK                 0x02
    #define REG_REVISION_CODE                   0x03
    #define REG_MEMCLK_CONFIG                   0x04
    #define REG_PCLK_CONFIG                     0x05
    #define REG_VCLK_CONFIG_0                   0x06
    #define REG_VCLK_CONFIG_1                   0x07
    #define REG_LUT_BLUE_WRITE_DATA             0x08
    #define REG_LUT_GREEN_WRITE_DATA            0x09
    #define REG_LUT_RED_WRITE_DATA              0x0a
    #define REG_LUT_WRITE_ADDR                  0x0b
    #define REG_LUT_BLUE_READ_DATA              0x0c
    #define REG_LUT_GREEN_READ_DATA             0x0d
    #define REG_LUT_RED_READ_DATA               0x0e
    #define REG_LUT_READ_ADDR                   0x0f
    #define REG_PANEL_TYPE                      0x10
    #define REG_MOD_RATE                        0x11
    #define REG_HORIZ_TOTAL_0                   0x12
    #define REG_HORIZ_TOTAL_1                   0x13
    #define REG_HDP                             0x14
    #define REG_HDP_START_POS0                  0x16
    #define REG_HDP_START_POS1                  0x17
    #define REG_VERT_TOTAL0                     0x18
    #define REG_VERT_TOTAL1                     0x19
    #define REG_VDP0                            0x1c
    #define REG_VDP1                            0x1d
    #define REG_VDP_START_POS0                  0x1e
    #define REG_VDP_START_POS1                  0x1f
    #define REG_HSYNC_PULSE_WIDTH               0x20
    #define REG_LLINE_PULSE_START_SUBPIXEL_POS  0x21
    #define REG_HSYNC_PULSE_START_POS0          0x22
    #define REG_HSYNC_PULSE_START_POS1          0x23
    #define REG_VSYNC_PULSE_WIDTH               0x24
    #define REG_VSYNC_PULSE_START_POS0          0x26
    #define REG_VSYNC_PULSE_START_POS1          0x27
    #define REG_POST_PROCESSING_SATURATION      0x2c
    #define REG_POST_PROCESSING_BRIGHTNESS      0x2d
    #define REG_POST_PROCESSING_CONTRAST        0x2e
    #define REG_POST_PROCESSING_CTRL            0x2f

    #define REG_FPFRAME_START_OFFSET0           0x30
    #define REG_FPFRAME_START_OFFSET1           0x31
    #define REG_FPFRAME_STOP_OFFSET0            0x34
```

```
#define REG_FPFRAME_STOP_OFFSET1            0x35
#define REG_LSHIFT_POLARITY                 0x38
#define REG_GPIO1_PULSE_START               0x3c
#define REG_GPIO1_PULSE_STOP                0x3e
#define REG_GPIO2_PULSE_DELAY               0x40
#define REG_LCD_SUBPIXEL_ALIGNMENT          0x42
#define REG_STN_COLOR_DEPTH                 0x45

#define REG_INTERRUPT_FLAG                  0x48
#define REG_INTERRUPT_ENABLE                0x4A

#define REG_DYN_DITHER_CONTROL              0x50

#define REG_DISPLAY_MODE                    0x70
#define REG_SPECIAL_EFFECTS                 0x71
#define REG_RGB_SETTING                     0x1a4

#define REG_MAIN_WIN_DISP_START_ADDR0       0x74
#define REG_MAIN_WIN_DISP_START_ADDR1       0x75
#define REG_MAIN_WIN_DISP_START_ADDR2       0x76
#define REG_MAIN_WIN_ADDR_OFFSET0           0x78
#define REG_MAIN_WIN_ADDR_OFFSET1           0x79
#define REG_FLOAT_WIN_DISP_START_ADDR0      0x7c
#define REG_FLOAT_WIN_DISP_START_ADDR1      0x7d
#define REG_FLOAT_WIN_DISP_START_ADDR2      0x7e
#define REG_FLOAT_WIN_ADDR_OFFSET0          0x80
#define REG_FLOAT_WIN_ADDR_OFFSET1          0x81
#define REG_FLOAT_WIN_X_START_POS0          0x84
#define REG_FLOAT_WIN_X_START_POS1          0x85
#define REG_FLOAT_WIN_Y_START_POS0          0x88
#define REG_FLOAT_WIN_Y_START_POS1          0x89
#define REG_FLOAT_WIN_X_END_POS0            0x8c
#define REG_FLOAT_WIN_X_END_POS1            0x8d
#define REG_FLOAT_WIN_Y_END_POS0            0x90
#define REG_FLOAT_WIN_Y_END_POS1            0x91
#define REG_POWER_SAVE_CONFIG               0xa0
#define REG_SOFTWARE_RESET                  0xa2
#define REG_SCRATCH_PAD0                    0xa4
#define REG_SCRATCH_PAD1                    0xa5
#define REG_GPIO_CONFIG0                    0xa8
#define REG_GPIO_CONFIG1                    0xa9
#define REG_GPIO_STATUS_CONTROL0            0xac
#define REG_GPIO_STATUS_CONTROL1            0xad
#define REG_PWM_CV_CLOCK_CONTROL            0xb0
#define REG_PWM_CV_CLOCK_CONFIG             0xb1
#define REG_CV_CLOCK_BURST_LENGTH           0xb2
#define REG_PWM_CLOCK_DUTY_CYCLE            0xb3

#define REG_PWM1_CLOCK_CONTROL              0xb4
#define REG_PWM1_CLOCK_CONFIG               0xb5
#define REG_PWM1_CLOCK_DUTY_CYCLE           0xb7

#define REG_PWM2_CLOCK_CONTROL              0xb8
#define REG_PWM2_CLOCK_CONFIG               0xb9
#define REG_PWM2_CLOCK_DUTY_CYCLE           0xbb

#define REG_PWM3_CLOCK_CONTROL              0xbc
#define REG_PWM3_CLOCK_CONFIG               0xbd
#define REG_PWM3_CLOCK_DUTY_CYCLE           0xbf

#define REG_CURSOR_FEATURE                  0xc0
#define REG_CURSOR1_BLINK_TOTAL0            0xc4
#define REG_CURSOR1_BLINK_TOTAL1            0xc5
#define REG_CURSOR1_BLINK_ON0               0xc8
#define REG_CURSOR1_BLINK_ON1               0xc9
#define REG_CURSOR1_MEM_START0              0xcc
#define REG_CURSOR1_MEM_START1              0xcd
#define REG_CURSOR1_MEM_START2              0xce
#define REG_CURSOR1_POSX0                   0xd0
#define REG_CURSOR1_POSX1                   0xd1
#define REG_CURSOR1_POSY0                   0xd4
#define REG_CURSOR1_POSY1                   0xd5
```

```
#define REG_CURSOR1_HORIZ_SIZE_0            0xd8
#define REG_CURSOR1_HORIZ_SIZE_1            0xd9
#define REG_CURSOR1_VERT_SIZE_0             0xdc
#define REG_CURSOR1_VERT_SIZE_1             0xdd
#define REG_CURSOR1_COL_IND1_0              0xe0
#define REG_CURSOR1_COL_IND1_1              0xe1
#define REG_CURSOR1_COL_IND1_2              0xe2
#define REG_CURSOR1_COL_IND1_3              0xe3
#define REG_CURSOR1_COL_IND2_0              0xe4
#define REG_CURSOR1_COL_IND2_1              0xe5
#define REG_CURSOR1_COL_IND2_2              0xe6
#define REG_CURSOR1_COL_IND2_3              0xe7
#define REG_CURSOR1_COL_IND3_0              0xe8
#define REG_CURSOR1_COL_IND3_1              0xe9
#define REG_CURSOR1_COL_IND3_2              0xea
#define REG_CURSOR1_COL_IND3_3              0xeb
#define REG_CURSOR2_BLINK_TOTAL0            0xec
#define REG_CURSOR2_BLINK_TOTAL1            0xed
#define REG_CURSOR2_BLINK_ON0               0xf0
#define REG_CURSOR2_BLINK_ON1               0xf1
#define REG_CURSOR2_MEM_START0              0xf4
#define REG_CURSOR2_MEM_START1              0xf5
#define REG_CURSOR2_MEM_START2              0xf6
#define REG_CURSOR2_POSX0                   0xf8
#define REG_CURSOR2_POSX1                   0xf9
#define REG_CURSOR2_POSY0                   0xfc
#define REG_CURSOR2_POSY1                   0xfd
#define REG_CURSOR2_HORIZ_SIZE_0            0x100
#define REG_CURSOR2_HORIZ_SIZE_1            0x101
#define REG_CURSOR2_VERT_SIZE_0             0x104
#define REG_CURSOR2_VERT_SIZE_1             0x105
#define REG_CURSOR2_COL_IND1_0              0x108
#define REG_CURSOR2_COL_IND1_1              0x109
#define REG_CURSOR2_COL_IND1_2              0x10a
#define REG_CURSOR2_COL_IND1_3              0x10b
#define REG_CURSOR2_COL_IND2_0              0x10c
#define REG_CURSOR2_COL_IND2_1              0x10d
#define REG_CURSOR2_COL_IND2_2              0x10e
#define REG_CURSOR2_COL_IND2_3              0x10f
#define REG_CURSOR2_COL_IND3_0              0x110
#define REG_CURSOR2_COL_IND3_1              0x111
#define REG_CURSOR2_COL_IND3_2              0x112
#define REG_CURSOR2_COL_IND3_3              0x113

#define REG_MAIN_REFLESH                    0x12c

#define REG_PCLK_FREQ_RATIO_0               0x158
#define REG_PCLK_FREQ_RATIO_1               0x159
#define REG_PCLK_FREQ_RATIO_2               0x15a

#define REG_DV_OP_MODE                      0x160
#define REG_DV_FRAME_SAMPLING               0x161

#define REG_DV_NFRAME_POS_0                 0x162
#define REG_DV_NFRAME_POS_1                 0x163
#define REG_DV_JHORI_SIZE_0                 0x164
#define REG_DV_JHORI_SIZE_1                 0x165
#define REG_DV_JVERT_SIZE_0                 0x168
#define REG_DV_JVERT_SIZE_1                 0x169

#define REG_DV_JMEM_STR_0                   0x16c
#define REG_DV_JMEM_STR_1                   0x16d
#define REG_DV_JMEM_STR_2                   0x16e
#define REG_DV_JMEM_STR_3                   0x16f

#define REG_DV_VHDEC_RATIO                  0x170
#define REG_DV_VVDEC_RATIO                  0x171
#define REG_DV_JHDEC_RATIO                  0x172
#define REG_DV_JVDEC_RATIO                  0x173

#define REG_DV_HORI_PERIOD_0                0x174
#define REG_DV_HORI_PERIOD_1                0x175
```

```
#define REG_DV_HORI_MAX_0                   0x17c
#define REG_DV_HORI_MAX_1                   0x17d

#define REG_DV_VERT_MAX_0                   0x180
#define REG_DV_VERT_MAX_1                   0x181

#define REG_DV_HCROP_STR_0                  0x184   //x
#define REG_DV_HCROP_STR_1                  0x185   //x
#define REG_DV_VCROP_STR_0                  0x188   //x
#define REG_DV_VCROP_STR_1                  0x189   //x
#define REG_DV_HCROP_SIZE_0                 0x18c
#define REG_DV_HCROP_SIZE_1                 0x18d

#define REG_DV_VCROP_SIZE_0                 0x190
#define REG_DV_VCROP_SIZE_1                 0x191

#define REG_DV_FRAME_PULSE_WIDTH            0x194
#define REG_DV_VMEM_STR_ADDR1_0             0x19c
#define REG_DV_VMEM_STR_ADDR1_1             0x19d
#define REG_DV_VMEM_STR_ADDR1_2             0x19e
#define REG_DV_VMEM_STR_ADDR1_3             0x19f

#define REG_DV_VMEM_STR_ADDR2_0             0x1a0
#define REG_DV_VMEM_STR_ADDR2_1             0x1a1
#define REG_DV_VMEM_STR_ADDR2_2             0x1a2
#define REG_DV_VMEM_STR_ADDR2_3             0x1a3

#define REG_DV_OFORMAT                      0x1a4
#define REG_DV_ALPHA                        0x1a5
#define REG_DV_SUBPIXEL_MODE                0x1a6
#define REG_DV_CSC_MODE                     0x1a8
#define REG_DV_Y                            0x1a9
#define REG_DV_CB                           0x1aa
#define REG_DV_CR                           0x1ab

#define REG_DV_TV_0                         0x1ac
#define REG_DV_TV_1                         0x1ad
#define REG_DV_TV_2                         0x1ae

#define REG_DV_ENB                          0x1af

#define REG_DV_DV0_START_ADDR_0             0x1b0
#define REG_DV_DV0_START_ADDR_1             0x1b1
#define REG_DV_DV0_START_ADDR_2             0x1b2

#define REG_DV_DV1_START_ADDR_0             0x1b4
#define REG_DV_DV1_START_ADDR_1             0x1b5
#define REG_DV_DV1_START_ADDR_2             0x1b6

#define REG_2D_1d0                      0x1d0
#define REG_2D_1d1                      0x1d1
#define REG_2D_1d2                      0x1d2
#define REG_2D_1d4                      0x1d4
#define REG_2D_1d5                      0x1d5
#define REG_2D_1d6                      0x1d6
#define REG_2D_1d8                      0x1d8
#define REG_2D_1d9                      0x1d9
#define REG_2D_1dc                      0x1dc
#define REG_2D_1dd                      0x1dd
#define REG_2D_1de                      0x1de
#define REG_2D_1e4                      0x1e4
#define REG_2D_1e5                      0x1e5
#define REG_2D_1e8                      0x1e8
#define REG_2D_1e9                      0x1e9
#define REG_2D_1ec                      0x1ec
#define REG_2D_1ed                      0x1ed
#define REG_2D_1f0                      0x1f0
#define REG_2D_1f1                      0x1f1
#define REG_2D_1f4                      0x1f4
#define REG_2D_1f5                      0x1f5
#define REG_2D_1f6                      0x1f6
```

```
#define REG_2D_1f8                      0x1f8
#define REG_2D_1f9                      0x1f9
#define REG_2D_1fc                      0x1fc
#define REG_2D_1fd                      0x1fd
#define REG_2D_1fe                      0x1fe
#define REG_2D_204                      0x204
#define REG_2D_205                      0x205
#define REG_2D_206                      0x206
#define REG_2D_208                      0x208
#define REG_2D_209                      0x209
#define REG_2D_214                      0x214
#define REG_2D_215                      0x215
#define REG_2D_218                      0x218
#define REG_2D_219                      0x219
#define REG_2D_220                      0x220

#define REG_2D_CMD_1                    0x1d0
#define REG_2D_CMD_2                    0x1d1
#define REG_2D_CMD_FIFO_STATUS          0x1d2
#define REG_2D_SRC_WND_START_ADDR0      0x1d4
#define REG_2D_SRC_WND_START_ADDR1      0x1d5
#define REG_2D_SRC_WND_START_ADDR2      0x1d6
#define REG_2D_SRC_WND_ADDR_OFFSET0     0x1d8
#define REG_2D_SRC_WND_ADDR_OFFSET1     0x1d9
#define REG_2D_SRC_WND_COLOR_MODE       0x1dc
#define REG_2D_DEST_WND_COLOR_MODE      0x1dd
#define REG_2D_SRC_WND_WIDTH0           0x1e4
#define REG_2D_SRC_WND_WIDTH1           0x1e5
#define REG_2D_SRC_WND_HEIGHT0          0x1e8
#define REG_2D_SRC_WND_HEIGHT1          0x1e9
#define REG_2D_DEST_WND_WIDTH0          0x1ec
#define REG_2D_DEST_WND_WIDTH1          0x1ed
#define REG_2D_DEST_WND_HEIGHT0         0x1f0
#define REG_2D_DEST_WND_HEIGHT1         0x1f1
#define REG_2D_DEST_WND_START_ADDR0     0x1f4
#define REG_2D_DEST_WND_START_ADDR1     0x1f5
#define REG_2D_DEST_WND_START_ADDR2     0x1f6
#define REG_2D_DEST_WND_ADDR_OFFSET0    0x1f8
#define REG_2D_DEST_WND_ADDR_OFFSET1    0x1f9
#define REG_2D_WRTIE_PATTERN0           0x1fc
#define REG_2D_WRTIE_PATTERN1           0x1fd
#define REG_2D_WRTIE_PATTERN2           0x1fe
#define REG_2D_BRUSH_WND_START_ADDR0    0x204
#define REG_2D_BRUSH_WND_START_ADDR1    0x205
#define REG_2D_BRUSH_WND_START_ADDR2    0x206
#define REG_2D_BRUSH_WND_ADDR_OFFSET0   0x208
#define REG_2D_BRUSH_WND_ADDR_OFFSET1   0x209
#define REG_2D_BRUSH_WND_WIDTH0         0x214
#define REG_2D_BRUSH_WND_WIDTH1         0x215
#define REG_2D_BRUSH_WND_HEIGHT0        0x218
#define REG_2D_BRUSH_WND_HEIGHT1        0x219
#define REG_2D_CMD_FIFO_INT_EN          0x21c
#define REG_2D_CMD_FIFO_INT_STATUS      0x21e
#define REG_2D_CMD_FIFO_FLAG            0x220
#define REG_2D_CMD_FIFO_IND             0x222

#define REG_I2C_DATA_OUT                0x230
#define REG_I2C_CTL                     0x231
#define REG_I2C_EN_OUT                  0x232
#define REG_I2C_BAUD                    0x233
#define REG_I2C_STATUS                  0x234
#define REG_I2C_INTERRUPT               0x235
#define REG_I2C_DATA_READY              0x236
#define REG_I2C_DATA_IN                 0x237

#define REG_Sub_mode                    0x250
#define REG_Sub_CLK_Divide              0x252
#define REG_Sub_Hori_Size               0x254
#define REG_Sub_Vert_Size               0x25c
#define REG_Sub_Type                    0x260
#define REG_Sub_Bpp                     0x261
#define REG_Sub_TFT_Start               0x262
```

```
#define REG_Sub_SCLK_Divide             0x263

#define REG_Sub_Ctl_0                   0x270
#define REG_Sub_Ctl_1                   0x271
#define REG_Sub_YVU                     0x268
#define REG_Sub_Y_Offset                0x269
#define REG_Sub_Cb_Offset               0x26a
#define REG_Sub_Cr_Offset               0x26b
#define REG_Sub_Data_0                  0x26c
#define REG_Sub_Data_1                  0x26d
#define REG_Sub_RS                      0x26e
#define REG_Sub_Start_Adr_0             0x274
#define REG_Sub_Start_Adr_1             0x275
#define REG_Sub_Start_Adr_2             0x276
#define REG_Sub_Offset_0                0x278
#define REG_Sub_Offset_1                0x279
#define REG_Sub_Ready                   0x27d

#define REG_LCD0_Rise_0                 0x2b0
#define REG_LCD0_Rise_1                 0x2b1
#define REG_LCD0_Fall_0                 0x2b4
#define REG_LCD0_Fall_1                 0x2b5
#define REG_LCD0_Period_0               0x2b8
#define REG_LCD0_Period_1               0x2b9
#define REG_LCD0_Ctl_0                  0x2bc
#define REG_LCD0_Ctl_1                  0x2bd
#define REG_LCD1_Rise_0                 0x2c0
#define REG_LCD1_Rise_1                 0x2c1
#define REG_LCD1_Fall_0                 0x2c4
#define REG_LCD1_Fall_1                 0x2c5
#define REG_LCD1_Period_0               0x2c8
#define REG_LCD1_Period_1               0x2c9
#define REG_LCD1_Ctl_0                  0x2cc
#define REG_LCD1_Ctl_1                  0x2cd
#define REG_LCD2_Rise_0                 0x2d0
#define REG_LCD2_Rise_1                 0x2d1
#define REG_LCD2_Fall_0                 0x2d4
#define REG_LCD2_Fall_1                 0x2d5
#define REG_LCD2_Period_0               0x2d8
#define REG_LCD2_Period_1               0x2d9
#define REG_LCD2_Ctl_0                  0x2dc
#define REG_LCD2_Ctl_1                  0x2dd
#define REG_LCD3_Rise_0                 0x2e0
#define REG_LCD3_Rise_1                 0x2e1
#define REG_LCD3_Fall_0                 0x2e4
#define REG_LCD3_Fall_1                 0x2e5
#define REG_LCD3_Period_0               0x2e8
#define REG_LCD3_Period_1               0x2e9
#define REG_LCD3_Ctl_0                  0x2ec
#define REG_LCD3_Ctl_1                  0x2ed
#define REG_LCD4_Rise_0                 0x2f0
#define REG_LCD4_Rise_1                 0x2f1
#define REG_LCD4_Fall_0                 0x2f4
#define REG_LCD4_Fall_1                 0x2f5
#define REG_LCD4_Period_0               0x2f8
#define REG_LCD4_Period_1               0x2f9
#define REG_LCD4_Ctl_0                  0x2fc
#define REG_LCD4_Ctl_1                  0x2fd
#define REG_LCD5_Rise_0                 0x300
#define REG_LCD5_Rise_1                 0x301
#define REG_LCD5_Fall_0                 0x304
#define REG_LCD5_Fall_1                 0x305
#define REG_LCD5_Period_0               0x308
#define REG_LCD5_Period_1               0x309
#define REG_LCD5_Ctl_0                  0x30c
#define REG_LCD5_Ctl_1                  0x30d
#define REG_LCD6_Rise_0                 0x310
#define REG_LCD6_Rise_1                 0x311
#define REG_LCD6_Fall_0                 0x314
#define REG_LCD6_Fall_1                 0x315
#define REG_LCD6_Period_0               0x318
#define REG_LCD6_Period_1               0x319
```

```
#define REG_LCD6_Ctl_0                  0x31c
#define REG_LCD6_Ctl_1                  0x31d
#define REG_LCD7_Rise_0                 0x320
#define REG_LCD7_Rise_1                 0x321
#define REG_LCD7_Fall_0                 0x324
#define REG_LCD7_Fall_1                 0x325
#define REG_LCD7_Period_0               0x328
#define REG_LCD7_Period_1               0x329
#define REG_LCD7_Ctl_0                  0x32c
#define REG_LCD7_Ctl_1                  0x32d

#define REG_FRC_FRAME_CTL               0x334
#define REG_FRC_ENABLE                  0x336

#define REG_JPEG_RESIZER_CTL            0x360
#define REG_JPEG_RESIZER_STARTX_0       0x364
#define REG_JPEG_RESIZER_STARTX_1       0x365
#define REG_JPEG_RESIZER_STARTY_0       0x366
#define REG_JPEG_RESIZER_STARTY_1       0x367
#define REG_JPEG_RESIZER_ENDX_0         0x368
#define REG_JPEG_RESIZER_ENDX_1         0x369
#define REG_JPEG_RESIZER_ENDY_0         0x36a
#define REG_JPEG_RESIZER_ENDY_1         0x36b
#define REG_JPEG_RESIZER_OP_0           0x36c
#define REG_JPEG_RESIZER_OP_1           0x36e

#define REG_JPEG_CTRL                   0x380
#define REG_JPEG_STATUS                 0x382
#define REG_JPEG_STATUS1                0x383

#define REG_JPEG_RAW_STATUS             0x384
#define REG_JPEG_RAW_STATUS1            0x385

#define REG_JPEG_INTR_CTL0              0x386
#define REG_JPEG_INTR_CTL1              0x387

#define REG_JPEG_START_STOP             0x38a

#define REG_JPEG_FIFO_CTL               0x3a0
#define REG_JPEG_FIFO_STATUS            0x3a2
#define REG_JPEG_FIFO_SIZE              0x3a4

#define REG_JPEG_ENCODE_SIZE_LIMIT_0    0x3B0
#define REG_JPEG_ENCODE_SIZE_LIMIT_1    0x3B1
#define REG_JPEG_ENCODE_SIZE_LIMIT_2    0x3B2

#define REG_JPEG_ENCODE_SIZE_RESULT_0   0x3b4
#define REG_JPEG_ENCODE_SIZE_RESULT_1   0x3b5
#define REG_JPEG_ENCODE_SIZE_RESULT_2   0x3b6

#define REG_JPEG_FILE_SIZE_0            0x3B8
#define REG_JPEG_FILE_SIZE_1            0x3B9
#define REG_JPEG_FILE_SIZE_2            0x3BA

#define REG_JPEG_DECODE_X_SIZE          0x3d8
#define REG_JPEG_DECODE_Y_SIZE          0x3dc

#define REG_JPEG_OP_MODE_ENC            0x400
#define REG_JPEG_OP_MODE                0x401

#define REG_JPEG_CMD                    0x402

#define REG_DRI_SETTING                 0x40a

#define REG_JPEG_DECODE_VALUE           0x404

#define REG_JPEG_Y_PIXEL_SIZE_0         0x40c
#define REG_JPEG_Y_PIXEL_SIZE_1         0x40d
#define REG_JPEG_X_PIXEL_SIZE_0         0x40e
#define REG_JPEG_X_PIXEL_SIZE_1         0x40f

#define REG_JPEG_SRC_START_ADDR_0       0x410
```

```
#define REG_JPEG_SRC_START_ADDR_1        0x411
#define REG_JPEG_SRC_START_ADDR_2        0x412
#define REG_JPEG_SRC_START_ADDR_3        0x413

#define REG_JPEG_DEST_START_ADDR_0       0x414
#define REG_JPEG_DEST_START_ADDR_1       0x415
#define REG_JPEG_DEST_START_ADDR_2       0x416
#define REG_JPEG_DEST_START_ADDR_3       0x417

#define REG_JPEG_RST_MARKER              0x41c

#define REG_JPEG_RST_MARKER_STATUS       0x41e

#define REG_JPEG_INSERT_MARKER00         0x420
#define REG_JPEG_INSERT_MARKER01         0x422
#define REG_JPEG_MARKER_LENGTH00         0x424
#define REG_JPEG_MARKER_LENGTH01         0x426

#define REG_JPEG_MARKER_DATA_00          0x428
#define REG_JPEG_MARKER_DATA_01          0x42a
#define REG_JPEG_MARKER_DATA_02          0x42c
#define REG_JPEG_MARKER_DATA_03          0x42e
#define REG_JPEG_MARKER_DATA_04          0x430
#define REG_JPEG_MARKER_DATA_05          0x432
#define REG_JPEG_MARKER_DATA_06          0x434
#define REG_JPEG_MARKER_DATA_07          0x436
#define REG_JPEG_MARKER_DATA_08          0x438
#define REG_JPEG_MARKER_DATA_09          0x43a
#define REG_JPEG_MARKER_DATA_10          0x43c
#define REG_JPEG_MARKER_DATA_11          0x43e
#define REG_JPEG_MARKER_DATA_12          0x440
#define REG_JPEG_MARKER_DATA_13          0x442
#define REG_JPEG_MARKER_DATA_14          0x444
#define REG_JPEG_MARKER_DATA_15          0x446
#define REG_JPEG_MARKER_DATA_16          0x448
#define REG_JPEG_MARKER_DATA_17          0x44a
#define REG_JPEG_MARKER_DATA_18          0x44c
#define REG_JPEG_MARKER_DATA_19          0x44e
#define REG_JPEG_MARKER_DATA_20          0x450
#define REG_JPEG_MARKER_DATA_21          0x452
#define REG_JPEG_MARKER_DATA_22          0x454
#define REG_JPEG_MARKER_DATA_23          0x456
#define REG_JPEG_MARKER_DATA_24          0x458
#define REG_JPEG_MARKER_DATA_25          0x45a
#define REG_JPEG_MARKER_DATA_26          0x45c
#define REG_JPEG_MARKER_DATA_27          0x45e
#define REG_JPEG_MARKER_DATA_28          0x460
#define REG_JPEG_MARKER_DATA_29          0x462
#define REG_JPEG_MARKER_DATA_30          0x464
#define REG_JPEG_MARKER_DATA_31          0x466

#define REG_JPEG_SOI_CONST_00            0x468
#define REG_JPEG_SOI_CONST_01            0x46a

#define REG_JPEG_JFIF_CONST_00           0x46c
#define REG_JPEG_JFIF_CONST_01           0x46e
#define REG_JPEG_JFIF_CONST_02           0x470
#define REG_JPEG_JFIF_CONST_03           0x472
#define REG_JPEG_JFIF_CONST_04           0x474
#define REG_JPEG_JFIF_CONST_05           0x476
#define REG_JPEG_JFIF_CONST_06           0x478
#define REG_JPEG_JFIF_CONST_07           0x47a
#define REG_JPEG_JFIF_CONST_08           0x47c
#define REG_JPEG_JFIF_CONST_09           0x47e
#define REG_JPEG_JFIF_CONST_10           0x480
#define REG_JPEG_JFIF_CONST_11           0x482
#define REG_JPEG_JFIF_CONST_12           0x484
#define REG_JPEG_JFIF_CONST_13           0x486
#define REG_JPEG_JFIF_CONST_14           0x488
#define REG_JPEG_JFIF_CONST_15           0x48a
#define REG_JPEG_JFIF_CONST_16           0x48c
#define REG_JPEG_JFIF_CONST_17           0x48e
```

```
#define REG_JPEG_LUM_DC_HT_CONST_00     0x490
#define REG_JPEG_LUM_DC_HT_CONST_01     0x492
#define REG_JPEG_LUM_DC_HT_CONST_02     0x494
#define REG_JPEG_LUM_DC_HT_CONST_03     0x496
#define REG_JPEG_LUM_DC_HT_CONST_04     0x498

#define REG_JPEG_CHR_DC_HT_CONST_00     0x4a0
#define REG_JPEG_CHR_DC_HT_CONST_01     0x4a2
#define REG_JPEG_CHR_DC_HT_CONST_02     0x4a4
#define REG_JPEG_CHR_DC_HT_CONST_03     0x4a6
#define REG_JPEG_CHR_DC_HT_CONST_04     0x4a8

#define REG_JPEG_LUM_AC_HT_CONST_00     0x4b0
#define REG_JPEG_LUM_AC_HT_CONST_01     0x4b2
#define REG_JPEG_LUM_AC_HT_CONST_02     0x4b4
#define REG_JPEG_LUM_AC_HT_CONST_03     0x4b6
#define REG_JPEG_LUM_AC_HT_CONST_04     0x4b8

#define REG_JPEG_CHR_AC_HT_CONST_00     0x4c0
#define REG_JPEG_CHR_AC_HT_CONST_01     0x4c2
#define REG_JPEG_CHR_AC_HT_CONST_02     0x4c4
#define REG_JPEG_CHR_AC_HT_CONST_03     0x4c6
#define REG_JPEG_CHR_AC_HT_CONST_04     0x4c8

#define REG_JPEG_LUM_QT_CONST_00        0x4d0
#define REG_JPEG_LUM_QT_CONST_01        0x4d2
#define REG_JPEG_LUM_QT_CONST_02        0x4d4
#define REG_JPEG_LUM_QT_CONST_03        0x4d6
#define REG_JPEG_LUM_QT_CONST_04        0x4d8

#define REG_JPEG_CHR_QT_CONST_00        0x4e0
#define REG_JPEG_CHR_QT_CONST_01        0x4e2
#define REG_JPEG_CHR_QT_CONST_02        0x4e4
#define REG_JPEG_CHR_QT_CONST_03        0x4e6
#define REG_JPEG_CHR_QT_CONST_04        0x4e8

#define REG_JPEG_SOF_CONST_00           0x4f0
#define REG_JPEG_SOF_CONST_01           0x4f2
#define REG_JPEG_SOF_CONST_02           0x4f4
#define REG_JPEG_SOF_CONST_03           0x4f6
#define REG_JPEG_SOF_CONST_04           0x4f8

#define REG_JPEG_QUANT_T0_00            0x500
#define REG_JPEG_QUANT_T0_01            0x502
#define REG_JPEG_QUANT_T0_02            0x504
#define REG_JPEG_QUANT_T0_03            0x506
#define REG_JPEG_QUANT_T0_04            0x508
#define REG_JPEG_QUANT_T0_05            0x50a
#define REG_JPEG_QUANT_T0_06            0x50c
#define REG_JPEG_QUANT_T0_07            0x50e
#define REG_JPEG_QUANT_T0_08            0x510
#define REG_JPEG_QUANT_T0_09            0x512
#define REG_JPEG_QUANT_T0_10            0x514
#define REG_JPEG_QUANT_T0_11            0x516
#define REG_JPEG_QUANT_T0_12            0x518
#define REG_JPEG_QUANT_T0_13            0x51a
#define REG_JPEG_QUANT_T0_14            0x51c
#define REG_JPEG_QUANT_T0_15            0x51e
#define REG_JPEG_QUANT_T0_16            0x520
#define REG_JPEG_QUANT_T0_17            0x522
#define REG_JPEG_QUANT_T0_18            0x524
#define REG_JPEG_QUANT_T0_19            0x526
#define REG_JPEG_QUANT_T0_20            0x528
#define REG_JPEG_QUANT_T0_21            0x52a
#define REG_JPEG_QUANT_T0_22            0x52c
#define REG_JPEG_QUANT_T0_23            0x52e
#define REG_JPEG_QUANT_T0_24            0x530
#define REG_JPEG_QUANT_T0_25            0x532
#define REG_JPEG_QUANT_T0_26            0x534
#define REG_JPEG_QUANT_T0_27            0x536
#define REG_JPEG_QUANT_T0_28            0x538
```

```
#define REG_JPEG_QUANT_T0_29            0x53a
#define REG_JPEG_QUANT_T0_30            0x53c
#define REG_JPEG_QUANT_T0_31            0x53e
#define REG_JPEG_QUANT_T0_32            0x540
#define REG_JPEG_QUANT_T0_33            0x542
#define REG_JPEG_QUANT_T0_34            0x544
#define REG_JPEG_QUANT_T0_35            0x546
#define REG_JPEG_QUANT_T0_36            0x548
#define REG_JPEG_QUANT_T0_37            0x54a
#define REG_JPEG_QUANT_T0_38            0x54c
#define REG_JPEG_QUANT_T0_39            0x54e
#define REG_JPEG_QUANT_T0_40            0x550
#define REG_JPEG_QUANT_T0_41            0x552
#define REG_JPEG_QUANT_T0_42            0x554
#define REG_JPEG_QUANT_T0_43            0x556
#define REG_JPEG_QUANT_T0_44            0x558
#define REG_JPEG_QUANT_T0_45            0x55a
#define REG_JPEG_QUANT_T0_46            0x55c
#define REG_JPEG_QUANT_T0_47            0x55e
#define REG_JPEG_QUANT_T0_48            0x560
#define REG_JPEG_QUANT_T0_49            0x562
#define REG_JPEG_QUANT_T0_50            0x564
#define REG_JPEG_QUANT_T0_51            0x566
#define REG_JPEG_QUANT_T0_52            0x568
#define REG_JPEG_QUANT_T0_53            0x56a
#define REG_JPEG_QUANT_T0_54            0x56c
#define REG_JPEG_QUANT_T0_55            0x56e
#define REG_JPEG_QUANT_T0_56            0x570
#define REG_JPEG_QUANT_T0_57            0x572
#define REG_JPEG_QUANT_T0_58            0x574
#define REG_JPEG_QUANT_T0_59            0x576
#define REG_JPEG_QUANT_T0_60            0x578
#define REG_JPEG_QUANT_T0_61            0x57a
#define REG_JPEG_QUANT_T0_62            0x57c
#define REG_JPEG_QUANT_T0_63            0x57e

#define REG_JPEG_QUANT_T1_00            0x580
#define REG_JPEG_QUANT_T1_01            0x582
#define REG_JPEG_QUANT_T1_02            0x584
#define REG_JPEG_QUANT_T1_03            0x586
#define REG_JPEG_QUANT_T1_04            0x588
#define REG_JPEG_QUANT_T1_05            0x58a
#define REG_JPEG_QUANT_T1_06            0x58c
#define REG_JPEG_QUANT_T1_07            0x58e
#define REG_JPEG_QUANT_T1_08            0x590
#define REG_JPEG_QUANT_T1_09            0x592
#define REG_JPEG_QUANT_T1_10            0x594
#define REG_JPEG_QUANT_T1_11            0x596
#define REG_JPEG_QUANT_T1_12            0x598
#define REG_JPEG_QUANT_T1_13            0x59a
#define REG_JPEG_QUANT_T1_14            0x59c
#define REG_JPEG_QUANT_T1_15            0x59e
#define REG_JPEG_QUANT_T1_16            0x5a0
#define REG_JPEG_QUANT_T1_17            0x5a2
#define REG_JPEG_QUANT_T1_18            0x5a4
#define REG_JPEG_QUANT_T1_19            0x5a6
#define REG_JPEG_QUANT_T1_20            0x5a8
#define REG_JPEG_QUANT_T1_21            0x5aa
#define REG_JPEG_QUANT_T1_22            0x5ac
#define REG_JPEG_QUANT_T1_23            0x5ae
#define REG_JPEG_QUANT_T1_24            0x5b0
#define REG_JPEG_QUANT_T1_25            0x5b2
#define REG_JPEG_QUANT_T1_26            0x5b4
#define REG_JPEG_QUANT_T1_27            0x5b6
#define REG_JPEG_QUANT_T1_28            0x5b8
#define REG_JPEG_QUANT_T1_29            0x5ba
#define REG_JPEG_QUANT_T1_30            0x5bc
#define REG_JPEG_QUANT_T1_31            0x5be
#define REG_JPEG_QUANT_T1_32            0x5c0
#define REG_JPEG_QUANT_T1_33            0x5c2
#define REG_JPEG_QUANT_T1_34            0x5c4
#define REG_JPEG_QUANT_T1_35            0x5c6
```

```
#define REG_JPEG_QUANT_T1_36            0x5c8
#define REG_JPEG_QUANT_T1_37            0x5ca
#define REG_JPEG_QUANT_T1_38            0x5cc
#define REG_JPEG_QUANT_T1_39            0x5ce
#define REG_JPEG_QUANT_T1_40            0x5d0
#define REG_JPEG_QUANT_T1_41            0x5d2
#define REG_JPEG_QUANT_T1_42            0x5d4
#define REG_JPEG_QUANT_T1_43            0x5d6
#define REG_JPEG_QUANT_T1_44            0x5d8
#define REG_JPEG_QUANT_T1_45            0x5da
#define REG_JPEG_QUANT_T1_46            0x5dc
#define REG_JPEG_QUANT_T1_47            0x5de
#define REG_JPEG_QUANT_T1_48            0x5e0
#define REG_JPEG_QUANT_T1_49            0x5e2
#define REG_JPEG_QUANT_T1_50            0x5e4
#define REG_JPEG_QUANT_T1_51            0x5e6
#define REG_JPEG_QUANT_T1_52            0x5e8
#define REG_JPEG_QUANT_T1_53            0x5ea
#define REG_JPEG_QUANT_T1_54            0x5ec
#define REG_JPEG_QUANT_T1_55            0x5ee
#define REG_JPEG_QUANT_T1_56            0x5f0
#define REG_JPEG_QUANT_T1_57            0x5f2
#define REG_JPEG_QUANT_T1_58            0x5f4
#define REG_JPEG_QUANT_T1_59            0x5f6
#define REG_JPEG_QUANT_T1_60            0x5f8
#define REG_JPEG_QUANT_T1_61            0x5fa
#define REG_JPEG_QUANT_T1_62            0x5fc
#define REG_JPEG_QUANT_T1_63            0x5fe

#define REG_JPEG_DC_T0_R0_00            0x600
#define REG_JPEG_DC_T0_R0_01            0x602
#define REG_JPEG_DC_T0_R0_02            0x604
#define REG_JPEG_DC_T0_R0_03            0x606
#define REG_JPEG_DC_T0_R0_04            0x608
#define REG_JPEG_DC_T0_R0_05            0x60a
#define REG_JPEG_DC_T0_R0_06            0x60c
#define REG_JPEG_DC_T0_R0_07            0x60e
#define REG_JPEG_DC_T0_R0_08            0x610
#define REG_JPEG_DC_T0_R0_09            0x612
#define REG_JPEG_DC_T0_R0_10            0x614
#define REG_JPEG_DC_T0_R0_11            0x616
#define REG_JPEG_DC_T0_R0_12            0x618
#define REG_JPEG_DC_T0_R0_13            0x61a
#define REG_JPEG_DC_T0_R0_14            0x61c
#define REG_JPEG_DC_T0_R0_15            0x61e

#define REG_JPEG_DC_T0_R1_00            0x620
#define REG_JPEG_DC_T0_R1_01            0x622
#define REG_JPEG_DC_T0_R1_02            0x624
#define REG_JPEG_DC_T0_R1_03            0x626
#define REG_JPEG_DC_T0_R1_04            0x628
#define REG_JPEG_DC_T0_R1_05            0x62a
#define REG_JPEG_DC_T0_R1_06            0x62c
#define REG_JPEG_DC_T0_R1_07            0x62e
#define REG_JPEG_DC_T0_R1_08            0x630
#define REG_JPEG_DC_T0_R1_09            0x632
#define REG_JPEG_DC_T0_R1_10            0x634
#define REG_JPEG_DC_T0_R1_11            0x636

#define REG_JPEG_AC_T0_R0_00            0x640
#define REG_JPEG_AC_T0_R0_01            0x642
#define REG_JPEG_AC_T0_R0_02            0x644
#define REG_JPEG_AC_T0_R0_03            0x646
#define REG_JPEG_AC_T0_R0_04            0x648
#define REG_JPEG_AC_T0_R0_05            0x64a
#define REG_JPEG_AC_T0_R0_06            0x64c
#define REG_JPEG_AC_T0_R0_07            0x64e
#define REG_JPEG_AC_T0_R0_08            0x650
#define REG_JPEG_AC_T0_R0_09            0x652
#define REG_JPEG_AC_T0_R0_10            0x654
#define REG_JPEG_AC_T0_R0_11            0x656
#define REG_JPEG_AC_T0_R0_12            0x658
```

```
#define REG_JPEG_AC_T0_R0_13            0x65a
#define REG_JPEG_AC_T0_R0_14            0x65c
#define REG_JPEG_AC_T0_R0_15            0x65e

#define REG_JPEG_AC_T0_R1_00            0x660
#define REG_JPEG_AC_T0_R1_01            0x662
#define REG_JPEG_AC_T0_R1_02            0x664
#define REG_JPEG_AC_T0_R1_03            0x666
#define REG_JPEG_AC_T0_R1_04            0x668
#define REG_JPEG_AC_T0_R1_05            0x66a
#define REG_JPEG_AC_T0_R1_06            0x66c
#define REG_JPEG_AC_T0_R1_07            0x66e
#define REG_JPEG_AC_T0_R1_08            0x670
#define REG_JPEG_AC_T0_R1_09            0x672
#define REG_JPEG_AC_T0_R1_10            0x674
#define REG_JPEG_AC_T0_R1_11            0x676
#define REG_JPEG_AC_T0_R1_12            0x678
#define REG_JPEG_AC_T0_R1_13            0x67a
#define REG_JPEG_AC_T0_R1_14            0x67c
#define REG_JPEG_AC_T0_R1_15            0x67e
#define REG_JPEG_AC_T0_R1_16            0x680
#define REG_JPEG_AC_T0_R1_17            0x682
#define REG_JPEG_AC_T0_R1_18            0x684
#define REG_JPEG_AC_T0_R1_19            0x686
#define REG_JPEG_AC_T0_R1_20            0x688
#define REG_JPEG_AC_T0_R1_21            0x68a
#define REG_JPEG_AC_T0_R1_22            0x68c
#define REG_JPEG_AC_T0_R1_23            0x68e
#define REG_JPEG_AC_T0_R1_24            0x690
#define REG_JPEG_AC_T0_R1_25            0x692
#define REG_JPEG_AC_T0_R1_26            0x694
#define REG_JPEG_AC_T0_R1_27            0x696
#define REG_JPEG_AC_T0_R1_28            0x698
#define REG_JPEG_AC_T0_R1_29            0x69a
#define REG_JPEG_AC_T0_R1_30            0x69c
#define REG_JPEG_AC_T0_R1_31            0x69e
#define REG_JPEG_AC_T0_R1_32            0x6a0
#define REG_JPEG_AC_T0_R1_33            0x6a2
#define REG_JPEG_AC_T0_R1_34            0x6a4
#define REG_JPEG_AC_T0_R1_35            0x6a6
#define REG_JPEG_AC_T0_R1_36            0x6a8
#define REG_JPEG_AC_T0_R1_37            0x6aa
#define REG_JPEG_AC_T0_R1_38            0x6ac
#define REG_JPEG_AC_T0_R1_39            0x6ae
#define REG_JPEG_AC_T0_R1_40            0x6b0
#define REG_JPEG_AC_T0_R1_41            0x6b2
#define REG_JPEG_AC_T0_R1_42            0x6b4
#define REG_JPEG_AC_T0_R1_43            0x6b6
#define REG_JPEG_AC_T0_R1_44            0x6b8
#define REG_JPEG_AC_T0_R1_45            0x6ba
#define REG_JPEG_AC_T0_R1_46            0x6bc
#define REG_JPEG_AC_T0_R1_47            0x6be
#define REG_JPEG_AC_T0_R1_48            0x6c0
#define REG_JPEG_AC_T0_R1_49            0x6c2
#define REG_JPEG_AC_T0_R1_50            0x6c4
#define REG_JPEG_AC_T0_R1_51            0x6c6
#define REG_JPEG_AC_T0_R1_52            0x6c8
#define REG_JPEG_AC_T0_R1_53            0x6ca
#define REG_JPEG_AC_T0_R1_54            0x6cc
#define REG_JPEG_AC_T0_R1_55            0x6ce
#define REG_JPEG_AC_T0_R1_56            0x6d0
#define REG_JPEG_AC_T0_R1_57            0x6d2
#define REG_JPEG_AC_T0_R1_58            0x6d4
#define REG_JPEG_AC_T0_R1_59            0x6d6
#define REG_JPEG_AC_T0_R1_60            0x6d8
#define REG_JPEG_AC_T0_R1_61            0x6da
#define REG_JPEG_AC_T0_R1_62            0x6dc
#define REG_JPEG_AC_T0_R1_63            0x6de
#define REG_JPEG_AC_T0_R1_64            0x6e0
#define REG_JPEG_AC_T0_R1_65            0x6e2
#define REG_JPEG_AC_T0_R1_66            0x6e4
#define REG_JPEG_AC_T0_R1_67            0x6e6
```

```
#define REG_JPEG_AC_T0_R1_68            0x6e8
#define REG_JPEG_AC_T0_R1_69            0x6ea
#define REG_JPEG_AC_T0_R1_70            0x6ec
#define REG_JPEG_AC_T0_R1_71            0x6ee
#define REG_JPEG_AC_T0_R1_72            0x6f0
#define REG_JPEG_AC_T0_R1_73            0x6f2
#define REG_JPEG_AC_T0_R1_74            0x6f4
#define REG_JPEG_AC_T0_R1_75            0x6f6
#define REG_JPEG_AC_T0_R1_76            0x6f8
#define REG_JPEG_AC_T0_R1_77            0x6fa
#define REG_JPEG_AC_T0_R1_78            0x6fc
#define REG_JPEG_AC_T0_R1_79            0x6fe
#define REG_JPEG_AC_T0_R1_80            0x700
#define REG_JPEG_AC_T0_R1_81            0x702
#define REG_JPEG_AC_T0_R1_82            0x704
#define REG_JPEG_AC_T0_R1_83            0x706
#define REG_JPEG_AC_T0_R1_84            0x708
#define REG_JPEG_AC_T0_R1_85            0x70a
#define REG_JPEG_AC_T0_R1_86            0x70c
#define REG_JPEG_AC_T0_R1_87            0x70e
#define REG_JPEG_AC_T0_R1_88            0x710
#define REG_JPEG_AC_T0_R1_89            0x712
#define REG_JPEG_AC_T0_R1_90            0x714
#define REG_JPEG_AC_T0_R1_91            0x716
#define REG_JPEG_AC_T0_R1_92            0x718
#define REG_JPEG_AC_T0_R1_93            0x71a
#define REG_JPEG_AC_T0_R1_94            0x71c
#define REG_JPEG_AC_T0_R1_95            0x71e
#define REG_JPEG_AC_T0_R1_96            0x720
#define REG_JPEG_AC_T0_R1_97            0x722
#define REG_JPEG_AC_T0_R1_98            0x724
#define REG_JPEG_AC_T0_R1_99            0x726
#define REG_JPEG_AC_T0_R1_100           0x728
#define REG_JPEG_AC_T0_R1_101           0x72a
#define REG_JPEG_AC_T0_R1_102           0x72c
#define REG_JPEG_AC_T0_R1_103           0x72e
#define REG_JPEG_AC_T0_R1_104           0x730
#define REG_JPEG_AC_T0_R1_105           0x732
#define REG_JPEG_AC_T0_R1_106           0x734
#define REG_JPEG_AC_T0_R1_107           0x736
#define REG_JPEG_AC_T0_R1_108           0x738
#define REG_JPEG_AC_T0_R1_109           0x73a
#define REG_JPEG_AC_T0_R1_110           0x73c
#define REG_JPEG_AC_T0_R1_111           0x73e
#define REG_JPEG_AC_T0_R1_112           0x740
#define REG_JPEG_AC_T0_R1_113           0x742
#define REG_JPEG_AC_T0_R1_114           0x744
#define REG_JPEG_AC_T0_R1_115           0x746
#define REG_JPEG_AC_T0_R1_116           0x748
#define REG_JPEG_AC_T0_R1_117           0x74a
#define REG_JPEG_AC_T0_R1_118           0x74c
#define REG_JPEG_AC_T0_R1_119           0x74e
#define REG_JPEG_AC_T0_R1_120           0x750
#define REG_JPEG_AC_T0_R1_121           0x752
#define REG_JPEG_AC_T0_R1_122           0x754
#define REG_JPEG_AC_T0_R1_123           0x756
#define REG_JPEG_AC_T0_R1_124           0x758
#define REG_JPEG_AC_T0_R1_125           0x75a
#define REG_JPEG_AC_T0_R1_126           0x75c
#define REG_JPEG_AC_T0_R1_127           0x75e
#define REG_JPEG_AC_T0_R1_128           0x760
#define REG_JPEG_AC_T0_R1_129           0x762
#define REG_JPEG_AC_T0_R1_130           0x764
#define REG_JPEG_AC_T0_R1_131           0x766
#define REG_JPEG_AC_T0_R1_132           0x768
#define REG_JPEG_AC_T0_R1_133           0x76a
#define REG_JPEG_AC_T0_R1_134           0x76c
#define REG_JPEG_AC_T0_R1_135           0x76e
#define REG_JPEG_AC_T0_R1_136           0x770
#define REG_JPEG_AC_T0_R1_137           0x772
#define REG_JPEG_AC_T0_R1_138           0x774
#define REG_JPEG_AC_T0_R1_139           0x776
```

```
#define REG_JPEG_AC_T0_R1_140          0x778
#define REG_JPEG_AC_T0_R1_141          0x77a
#define REG_JPEG_AC_T0_R1_142          0x77c
#define REG_JPEG_AC_T0_R1_143          0x77e
#define REG_JPEG_AC_T0_R1_144          0x780
#define REG_JPEG_AC_T0_R1_145          0x782
#define REG_JPEG_AC_T0_R1_146          0x784
#define REG_JPEG_AC_T0_R1_147          0x786
#define REG_JPEG_AC_T0_R1_148          0x788
#define REG_JPEG_AC_T0_R1_149          0x78a
#define REG_JPEG_AC_T0_R1_150          0x78c
#define REG_JPEG_AC_T0_R1_151          0x78e
#define REG_JPEG_AC_T0_R1_152          0x790
#define REG_JPEG_AC_T0_R1_153          0x792
#define REG_JPEG_AC_T0_R1_154          0x794
#define REG_JPEG_AC_T0_R1_155          0x796
#define REG_JPEG_AC_T0_R1_156          0x798
#define REG_JPEG_AC_T0_R1_157          0x79a
#define REG_JPEG_AC_T0_R1_158          0x79c
#define REG_JPEG_AC_T0_R1_159          0x79e
#define REG_JPEG_AC_T0_R1_160          0x7a0
#define REG_JPEG_AC_T0_R1_161          0x7a2

#define REG_JPEG_DC_T1_R0_00           0x800
#define REG_JPEG_DC_T1_R0_01           0x802
#define REG_JPEG_DC_T1_R0_02           0x804
#define REG_JPEG_DC_T1_R0_03           0x806
#define REG_JPEG_DC_T1_R0_04           0x808
#define REG_JPEG_DC_T1_R0_05           0x80a
#define REG_JPEG_DC_T1_R0_06           0x80c
#define REG_JPEG_DC_T1_R0_07           0x80e
#define REG_JPEG_DC_T1_R0_08           0x810
#define REG_JPEG_DC_T1_R0_09           0x812
#define REG_JPEG_DC_T1_R0_10           0x814
#define REG_JPEG_DC_T1_R0_11           0x816
#define REG_JPEG_DC_T1_R0_12           0x818
#define REG_JPEG_DC_T1_R0_13           0x81a
#define REG_JPEG_DC_T1_R0_14           0x81c
#define REG_JPEG_DC_T1_R0_15           0x81e

#define REG_JPEG_DC_T1_R1_00           0x820
#define REG_JPEG_DC_T1_R1_01           0x822
#define REG_JPEG_DC_T1_R1_02           0x824
#define REG_JPEG_DC_T1_R1_03           0x826
#define REG_JPEG_DC_T1_R1_04           0x828
#define REG_JPEG_DC_T1_R1_05           0x82a
#define REG_JPEG_DC_T1_R1_06           0x82c
#define REG_JPEG_DC_T1_R1_07           0x82e
#define REG_JPEG_DC_T1_R1_08           0x830
#define REG_JPEG_DC_T1_R1_09           0x832
#define REG_JPEG_DC_T1_R1_10           0x834
#define REG_JPEG_DC_T1_R1_11           0x836

#define REG_JPEG_AC_T1_R0_00           0x840
#define REG_JPEG_AC_T1_R0_01           0x842
#define REG_JPEG_AC_T1_R0_02           0x844
#define REG_JPEG_AC_T1_R0_03           0x846
#define REG_JPEG_AC_T1_R0_04           0x848
#define REG_JPEG_AC_T1_R0_05           0x84a
#define REG_JPEG_AC_T1_R0_06           0x84c
#define REG_JPEG_AC_T1_R0_07           0x84e
#define REG_JPEG_AC_T1_R0_08           0x850
#define REG_JPEG_AC_T1_R0_09           0x852
#define REG_JPEG_AC_T1_R0_10           0x854
#define REG_JPEG_AC_T1_R0_11           0x856
#define REG_JPEG_AC_T1_R0_12           0x858
#define REG_JPEG_AC_T1_R0_13           0x85a
#define REG_JPEG_AC_T1_R0_14           0x85c
#define REG_JPEG_AC_T1_R0_15           0x85e

#define REG_JPEG_AC_T1_R1_00           0x860
#define REG_JPEG_AC_T1_R1_01           0x862
```

```
#define REG_JPEG_AC_T1_R1_02            0x864
#define REG_JPEG_AC_T1_R1_03            0x866
#define REG_JPEG_AC_T1_R1_04            0x868
#define REG_JPEG_AC_T1_R1_05            0x86a
#define REG_JPEG_AC_T1_R1_06            0x86c
#define REG_JPEG_AC_T1_R1_07            0x86e
#define REG_JPEG_AC_T1_R1_08            0x870
#define REG_JPEG_AC_T1_R1_09            0x872
#define REG_JPEG_AC_T1_R1_10            0x874
#define REG_JPEG_AC_T1_R1_11            0x876
#define REG_JPEG_AC_T1_R1_12            0x878
#define REG_JPEG_AC_T1_R1_13            0x87a
#define REG_JPEG_AC_T1_R1_14            0x87c
#define REG_JPEG_AC_T1_R1_15            0x87e
#define REG_JPEG_AC_T1_R1_16            0x880
#define REG_JPEG_AC_T1_R1_17            0x882
#define REG_JPEG_AC_T1_R1_18            0x884
#define REG_JPEG_AC_T1_R1_19            0x886
#define REG_JPEG_AC_T1_R1_20            0x888
#define REG_JPEG_AC_T1_R1_21            0x88a
#define REG_JPEG_AC_T1_R1_22            0x88c
#define REG_JPEG_AC_T1_R1_23            0x88e
#define REG_JPEG_AC_T1_R1_24            0x890
#define REG_JPEG_AC_T1_R1_25            0x892
#define REG_JPEG_AC_T1_R1_26            0x894
#define REG_JPEG_AC_T1_R1_27            0x896
#define REG_JPEG_AC_T1_R1_28            0x898
#define REG_JPEG_AC_T1_R1_29            0x89a
#define REG_JPEG_AC_T1_R1_30            0x89c
#define REG_JPEG_AC_T1_R1_31            0x89e
#define REG_JPEG_AC_T1_R1_32            0x8a0
#define REG_JPEG_AC_T1_R1_33            0x8a2
#define REG_JPEG_AC_T1_R1_34            0x8a4
#define REG_JPEG_AC_T1_R1_35            0x8a6
#define REG_JPEG_AC_T1_R1_36            0x8a8
#define REG_JPEG_AC_T1_R1_37            0x8aa
#define REG_JPEG_AC_T1_R1_38            0x8ac
#define REG_JPEG_AC_T1_R1_39            0x8ae
#define REG_JPEG_AC_T1_R1_40            0x8b0
#define REG_JPEG_AC_T1_R1_41            0x8b2
#define REG_JPEG_AC_T1_R1_42            0x8b4
#define REG_JPEG_AC_T1_R1_43            0x8b6
#define REG_JPEG_AC_T1_R1_44            0x8b8
#define REG_JPEG_AC_T1_R1_45            0x8ba
#define REG_JPEG_AC_T1_R1_46            0x8bc
#define REG_JPEG_AC_T1_R1_47            0x8be
#define REG_JPEG_AC_T1_R1_48            0x8c0
#define REG_JPEG_AC_T1_R1_49            0x8c2
#define REG_JPEG_AC_T1_R1_50            0x8c4
#define REG_JPEG_AC_T1_R1_51            0x8c6
#define REG_JPEG_AC_T1_R1_52            0x8c8
#define REG_JPEG_AC_T1_R1_53            0x8ca
#define REG_JPEG_AC_T1_R1_54            0x8cc
#define REG_JPEG_AC_T1_R1_55            0x8ce
#define REG_JPEG_AC_T1_R1_56            0x8d0
#define REG_JPEG_AC_T1_R1_57            0x8d2
#define REG_JPEG_AC_T1_R1_58            0x8d4
#define REG_JPEG_AC_T1_R1_59            0x8d6
#define REG_JPEG_AC_T1_R1_60            0x8d8
#define REG_JPEG_AC_T1_R1_61            0x8da
#define REG_JPEG_AC_T1_R1_62            0x8dc
#define REG_JPEG_AC_T1_R1_63            0x8de
#define REG_JPEG_AC_T1_R1_64            0x8e0
#define REG_JPEG_AC_T1_R1_65            0x8e2
#define REG_JPEG_AC_T1_R1_66            0x8e4
#define REG_JPEG_AC_T1_R1_67            0x8e6
#define REG_JPEG_AC_T1_R1_68            0x8e8
#define REG_JPEG_AC_T1_R1_69            0x8ea
#define REG_JPEG_AC_T1_R1_70            0x8ec
#define REG_JPEG_AC_T1_R1_71            0x8ee
#define REG_JPEG_AC_T1_R1_72            0x8f0
#define REG_JPEG_AC_T1_R1_73            0x8f2
```

```
#define REG_JPEG_AC_T1_R1_74            0x8f4
#define REG_JPEG_AC_T1_R1_75            0x8f6
#define REG_JPEG_AC_T1_R1_76            0x8f8
#define REG_JPEG_AC_T1_R1_77            0x8fa
#define REG_JPEG_AC_T1_R1_78            0x8fc
#define REG_JPEG_AC_T1_R1_79            0x8fe
#define REG_JPEG_AC_T1_R1_80            0x900
#define REG_JPEG_AC_T1_R1_81            0x902
#define REG_JPEG_AC_T1_R1_82            0x904
#define REG_JPEG_AC_T1_R1_83            0x906
#define REG_JPEG_AC_T1_R1_84            0x908
#define REG_JPEG_AC_T1_R1_85            0x90a
#define REG_JPEG_AC_T1_R1_86            0x90c
#define REG_JPEG_AC_T1_R1_87            0x90e
#define REG_JPEG_AC_T1_R1_88            0x910
#define REG_JPEG_AC_T1_R1_89            0x912
#define REG_JPEG_AC_T1_R1_90            0x914
#define REG_JPEG_AC_T1_R1_91            0x916
#define REG_JPEG_AC_T1_R1_92            0x918
#define REG_JPEG_AC_T1_R1_93            0x91a
#define REG_JPEG_AC_T1_R1_94            0x91c
#define REG_JPEG_AC_T1_R1_95            0x91e
#define REG_JPEG_AC_T1_R1_96            0x920
#define REG_JPEG_AC_T1_R1_97            0x922
#define REG_JPEG_AC_T1_R1_98            0x924
#define REG_JPEG_AC_T1_R1_99            0x926
#define REG_JPEG_AC_T1_R1_100           0x928
#define REG_JPEG_AC_T1_R1_101           0x92a
#define REG_JPEG_AC_T1_R1_102           0x92c
#define REG_JPEG_AC_T1_R1_103           0x92e
#define REG_JPEG_AC_T1_R1_104           0x930
#define REG_JPEG_AC_T1_R1_105           0x932
#define REG_JPEG_AC_T1_R1_106           0x934
#define REG_JPEG_AC_T1_R1_107           0x936
#define REG_JPEG_AC_T1_R1_108           0x938
#define REG_JPEG_AC_T1_R1_109           0x93a
#define REG_JPEG_AC_T1_R1_110           0x93c
#define REG_JPEG_AC_T1_R1_111           0x93e
#define REG_JPEG_AC_T1_R1_112           0x940
#define REG_JPEG_AC_T1_R1_113           0x942
#define REG_JPEG_AC_T1_R1_114           0x944
#define REG_JPEG_AC_T1_R1_115           0x946
#define REG_JPEG_AC_T1_R1_116           0x948
#define REG_JPEG_AC_T1_R1_117           0x94a
#define REG_JPEG_AC_T1_R1_118           0x94c
#define REG_JPEG_AC_T1_R1_119           0x94e
#define REG_JPEG_AC_T1_R1_120           0x950
#define REG_JPEG_AC_T1_R1_121           0x952
#define REG_JPEG_AC_T1_R1_122           0x954
#define REG_JPEG_AC_T1_R1_123           0x956
#define REG_JPEG_AC_T1_R1_124           0x958
#define REG_JPEG_AC_T1_R1_125           0x95a
#define REG_JPEG_AC_T1_R1_126           0x95c
#define REG_JPEG_AC_T1_R1_127           0x95e
#define REG_JPEG_AC_T1_R1_128           0x960
#define REG_JPEG_AC_T1_R1_129           0x962
#define REG_JPEG_AC_T1_R1_130           0x964
#define REG_JPEG_AC_T1_R1_131           0x966
#define REG_JPEG_AC_T1_R1_132           0x968
#define REG_JPEG_AC_T1_R1_133           0x96a
#define REG_JPEG_AC_T1_R1_134           0x96c
#define REG_JPEG_AC_T1_R1_135           0x96e
#define REG_JPEG_AC_T1_R1_136           0x970
#define REG_JPEG_AC_T1_R1_137           0x972
#define REG_JPEG_AC_T1_R1_138           0x974
#define REG_JPEG_AC_T1_R1_139           0x976
#define REG_JPEG_AC_T1_R1_140           0x978
#define REG_JPEG_AC_T1_R1_141           0x97a
#define REG_JPEG_AC_T1_R1_142           0x97c
#define REG_JPEG_AC_T1_R1_143           0x97e
#define REG_JPEG_AC_T1_R1_144           0x980
#define REG_JPEG_AC_T1_R1_145           0x982
```

```
    #define REG_JPEG_AC_T1_R1_146           0x984
    #define REG_JPEG_AC_T1_R1_147           0x986
    #define REG_JPEG_AC_T1_R1_148           0x988
    #define REG_JPEG_AC_T1_R1_149           0x98a
    #define REG_JPEG_AC_T1_R1_150           0x98c
    #define REG_JPEG_AC_T1_R1_151           0x98e
    #define REG_JPEG_AC_T1_R1_152           0x990
    #define REG_JPEG_AC_T1_R1_153           0x992
    #define REG_JPEG_AC_T1_R1_154           0x994
    #define REG_JPEG_AC_T1_R1_155           0x996
    #define REG_JPEG_AC_T1_R1_156           0x998
    #define REG_JPEG_AC_T1_R1_157           0x99a
    #define REG_JPEG_AC_T1_R1_158           0x99c
    #define REG_JPEG_AC_T1_R1_159           0x99e
    #define REG_JPEG_AC_T1_R1_160           0x9a0
    #define REG_JPEG_AC_T1_R1_161           0x9a2

    #define REG_JPEG_QTABLE_CONST_0         0x9b0
    #define REG_JPEG_QTABLE_CONST_1         0x9b2
    #define REG_JPEG_QTABLE_CONST_2         0x9b4
    #define REG_JPEG_QTABLE_CONST_3         0x9b6

    #define REG_JPEG_QTABLE0_SAMPLE         0x9b8
    #define REG_JPEG_QTABLE1_SAMPLE         0x9bc
    #define REG_JPEG_QTABLE2_SAMPLE         0x9c0

    #define REG_JPEG_DRI_CONST_0            0x9c4
    #define REG_JPEG_DRI_CONST_1            0x9c6
    #define REG_JPEG_DRI_CONST_2            0x9c8
    #define REG_JPEG_DRI_CONST_3            0x9ca

    #define REG_JPEG_SOS_CONST_0            0x9cc
    #define REG_JPEG_SOS_CONST_1            0x9ce
    #define REG_JPEG_SOS_CONST_2            0x9d0
    #define REG_JPEG_SOS_CONST_3            0x9d2
    #define REG_JPEG_SOS_CONST_4            0x9d4

    #define REG_JPEG_EOI_CONST_0            0x9e4
    #define REG_JPEG_EOI_CONST_1            0x9e6
    #define REG_JPEG_EOI_CONST_2            0x9e8
    #define REG_JPEG_EOI_CONST_3            0x9ea
    #define REG_JPEG_EOI_CONST_4            0x9ec

    #define REG_JPEG_VERT_PIX_SIZE0         0x9f0
    #define REG_JPEG_VERT_PIX_SIZE1         0x9f2
    #define REG_JPEG_HORI_PIX_SIZE0         0x9f4
    #define REG_JPEG_HORI_PIX_SIZE1         0x9f6
    #define REG_JPEG_DRI_CONFIG0            0x9f8
    #define REG_JPEG_DRI_CONFIG1            0x9fa

    #define GFX_LCD_TFT  0x01           // Type TFT Display
    #define GFX_LCD_CSTN 0x03           // Type Color STN Display
    #define GFX_LCD_MSTN 0x02           // Type Mono STN Display


#endif // _SSD1926_H
```

14.3.13 TCON_HX8238.c

This is file TCON_HX8238.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Himax HX8238 TCON driver
 *****************************************************************************
 * FileName:        TCON_HX8238.c
```

```
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2011 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/17/08     ...
 * 04/13/11     - Added GFX_USE_DISPLAY_CONTROLLER_xxx check.
 *              - Replaced TCON_Delay() with Delay10us() from TimeDelay.h
 *              - Renamed TCON_Init() to GfxTconInit() and the rest
 *                of the internal functions.
 *              - Graphics Library Version 3.00 Support
 ****************************************************************************/
#include "GraphicsConfig.h"
#include "HardwareProfile.h"
#include "TimeDelay.h"

#ifdef USE_TCON_HX8238

#if defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)
    #include "Graphics/SSD1926.h"
#elif defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)
    // do not need any driver header file
#else
    #warning "This Timing Controller Driver is written for SSD1926, Microchip Display
Controller using DMA and Microchip Graphics Module driver. If you are not using those
drivers you may need to re-write this driver."
#endif

/*
    This module can use SPI or bitbang the SPI transfer using
    the SSD1926 GPIO pins.
    This assumes that the timing controller uses SPI to
    receive Timing Programming data.
    Select the usage using the hardware profile.
*/
    // BB means BitBang
    #define BB_CS                   0x01
    #define BB_SCL                  0x02
    #define BB_SDO                  0x04
    #define BB_DC                   0x08

#if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)

    #define TCON_CSLow()            (TCON_CS_LAT = 0)
    #define TCON_CSHigh()           (TCON_CS_LAT = 1)
```

```
    #define TCON_CLKLow()           (TCON_SCL_LAT = 0)
    #define TCON_CLKHigh()          (TCON_SCL_LAT = 1)
    #define TCON_DataLow()          (TCON_SDO_LAT = 0)
    #define TCON_DataHigh()         (TCON_SDO_LAT = 1)

    #define TCON_SetCommand()       (TCON_DC_LAT = 0)
    #define TCON_SetData()          (TCON_DC_LAT = 1)

    // set the IOs used to outputs. and initialize them all to "1"
    // set up the TRIS first for outputs, then set the pins
    // to digital in case it is needed and initialize the signals
    // to all high.
    #define InitBitBangedIO() {                     \
                                TCON_CS_TRIS = 0;   \
                                TCON_SCL_TRIS = 0;  \
                                TCON_SDO_TRIS = 0;  \
                                TCON_DC_TRIS = 0;   \
                                TCON_CS_DIG();      \
                                TCON_SCL_DIG();     \
                                TCON_SDO_DIG();     \
                                TCON_DC_DIG();      \
                                TCON_CSHigh();      \
                                TCON_CLKHigh();     \
                                TCON_DataHigh();    \
                                TCON_SetData();     \
                              }

// end of #if defined (USE_DISPLAY_CONTROLLER_MCHP_DA210)


#elif defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)

    // use the bitbang version using SSD1926 GPIO pins

    #define TCON_CSLow()            (GfxTconSetIO(CS, 0))
    #define TCON_CSHigh()           (GfxTconSetIO(CS, 1))
    #define TCON_CLKLow()           (GfxTconSetIO(SCL, 0))
    #define TCON_CLKHigh()          (GfxTconSetIO(SCL, 1))
    #define TCON_DataLow()          (GfxTconSetIO(SDO, 0))
    #define TCON_DataHigh()         (GfxTconSetIO(SDO, 1))

    #define TCON_SetCommand()       (GfxTconSetIO(DC, 0))
    #define TCON_SetData()          (GfxTconSetIO(DC, 1))

    // this is only needed here since the controller IO's are used
    // instead of the IO's from the PIC device.
    #define SetCtrlBitBangedIO(addr, data)    (SetReg(addr, data))

    // set the GPIO of SSD1926 to as outputs. (used for TCON signals)
    // and initialize them all to "1"
    #define InitBitBangedIO() {                                 \
                                SetCtrlBitBangedIO(0xA8, 0x1F);     \
                                SetCtrlBitBangedIO(0xAC, 0x1F);     \
                              }
// end of #elif defined (USE_TCON_SSD1289)

#endif

/***************************************************************************
* Function: void GfxTconSetIO(BYTE mask, BYTE
level)
*
* Overview: This sets the IO specified by mask to the value set by
*           level.
*
* Input: mask - specifies the IO to be toggles.
*        level - specifies the logic where the IO will be set.
*
* Output: none
*
***************************************************************************/
void GfxTconSetIO(BYTE mask, BYTE level)
```

```
{

 #if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)

    switch(mask)
    {
        case BB_CS:    (level == 1 ? TCON_CSHigh() : TCON_CSLow());
                    break;

        case BB_SCL:   (level == 1 ? TCON_CLKHigh() : TCON_CLKLow());
                    break;

        case BB_SDO:   (level == 1 ? TCON_DataHigh() : TCON_DataLow());
                    break;

        case BB_DC:    (level == 1 ? TCON_SetData() : TCON_SetCommand());
                    break;
        default:
                    break;
    }

    Nop();

 #elif defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)

    static BYTE value = 0xFF;

    if(level == 0)
    {
        value &= ~mask;
    }
    else
    {
        value |= mask;
    }

    SetCtrlBitBangedIO(0xAC, value);

#endif

}

/***************************************************************************
* Function: void GfxTconWriteByte(BYTE value)
*
* Overview: This writes a bytes to SPI.
*
* Input: value - the byte data to be written.
*
* Output: none
*
***************************************************************************/
void GfxTconWriteByte(BYTE value)
{
    BYTE    mask;

    mask = 0x80;
    while(mask)
    {
        GfxTconSetIO(BB_SCL, 0);
        Delay10us(1);
        if(mask & value)
        {
            GfxTconSetIO(BB_SDO, 1);
        }
        else
        {
            GfxTconSetIO(BB_SDO, 0);
        }

        GfxTconSetIO(BB_SCL, 1);
```

```
        mask >>= 1;
    }
}

/***************************************************************************
* Function: void GfxTconWriteCommand(BYTE index, WORD
value)
*
* Overview: This writes a word to SPI by calling the write byte
*           routine.
*
* Input: index - The index (or address) of the register to be written.
*        value - The value that will be written to the register.
*
* Output: none
*
***************************************************************************/
void GfxTconWriteCommand(WORD index, WORD value)
/* */
{
    GfxTconSetIO(BB_CS, 0);

    // Index
    GfxTconWriteByte(0x70);
    GfxTconWriteByte(((WORD_VAL) index).v[1]);
    GfxTconWriteByte(((WORD_VAL) index).v[0]);

    GfxTconSetIO(BB_CS, 1);
    Delay10us(1);
    GfxTconSetIO(BB_CS, 0);

    // Data
    GfxTconWriteByte(0x72);
    GfxTconWriteByte(((WORD_VAL) value).v[1]);
    GfxTconWriteByte(((WORD_VAL) value).v[0]);
    GfxTconSetIO(BB_CS, 1);
    Delay10us(1);
}


/***************************************************************************
* Function: void GfxTconInit(void)
*
* Overview: Initialize the IOs to implement Bitbanged SPI interface
*           to connect to the Timing Controller through SPI.
*
* Input: none
*
* Output: none
*
***************************************************************************/
void GfxTconInit(void)
{
    InitBitBangedIO();

    DelayMs(20);

    GfxTconWriteCommand(0x0001, 0x6300);  //Driver Output Control
    GfxTconWriteCommand(0x0002, 0x0200);  //LCD-Driving-Waveform Control:N-line
inversion,N=0
    GfxTconWriteCommand(0x0003, 0x7184);  //Power control 1:Set the step-up cycle of the
step-up circuit for 262k-color mode
    DelayMs(100);
    GfxTconWriteCommand(0x0004, 0x0447);  //Input Data and Color Filter Control
    GfxTconWriteCommand(0x0005, 0xb854);  //Function Control
    GfxTconWriteCommand(0X000a, 0x4008);  //Contrast/Brightness Control

    //GfxTconWriteCommand(0X000a,0xff18);//Contrast/Brightness Control

    DelayMs(40);
    GfxTconWriteCommand(0x000b, 0xd400);  //Frame Cycle Control
    GfxTconWriteCommand(0x000d, 0x123a);  //Power Control 2
```

```
    DelayMs(200);
    GfxTconWriteCommand(0x000e, 0x3000);  //Power Control 3
    DelayMs(200);
    GfxTconWriteCommand(0x000f, 0x0000);  //Gate Scan Position
    GfxTconWriteCommand(0x0016, 0x9f80);  //Horizontal Porch
    GfxTconWriteCommand(0x0017, 0x2212);  //Vertical Porch
    DelayMs(200);
    GfxTconWriteCommand(0x001e, 0x00ef);  //Set the VCOMH voltage
    DelayMs(200);

    GfxTconWriteCommand(0x0030, 0x0507);  //Gamma Control
    GfxTconWriteCommand(0x0031, 0x0004);  //Gamma Control
    GfxTconWriteCommand(0x0032, 0x0707);  //Gamma Control
    GfxTconWriteCommand(0x0033, 0x0000);  //Gamma Control
    GfxTconWriteCommand(0x0034, 0x0000);  //Gamma Control
    GfxTconWriteCommand(0x0035, 0x0307);  //Gamma Control
    GfxTconWriteCommand(0x0036, 0x0700);  //Gamma Control
    GfxTconWriteCommand(0x0037, 0x0000);  //Gamma Control
    GfxTconWriteCommand(0x003a, 0x140b);  //Gamma Control
    GfxTconWriteCommand(0x003b, 0x140b);  //Gamma Control
}

#endif // #ifdef USE_TCON_HX8238
```

14.3.14 TCON_SSD1289.c

This is file TCON_SSD1289.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Solomon Systech. SSD1289 TCON  driver
 *****************************************************************************
 * FileName:        TCON_SSD1289.c
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2011 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 11/17/08
 * 04/05/10     Modified initialization to reduce flicker.
 * 02/11/11     - Added GFX_USE_DISPLAY_CONTROLLER_xxx check.
```

```
 *                - Replaced TCON_Delay() with Delay10us() from TimeDelay.h
 *                - Renamed TCON_Init() to GfxTconInit() and the rest
 *                  of the internal functions.
 ****************************************************************************/
#include "GraphicsConfig.h"
#include "HardwareProfile.h"
#include "TimeDelay.h"

#if defined(USE_TCON_SSD1289)

#if defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)
    #include "Graphics/SSD1926.h"
#elif defined (GFX_USE_DISPLAY_CONTROLLER_S1D13517)
    #include "Graphics/S1D13517.h"
#elif defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)
    // do not need any driver header file
#else
    #warning "This Timing Controller Driver is written for SSD1926, S1D13517 and Microchip
Graphics Module driver. If you are not using those drivers you may need to re-write this
driver."
#endif

/*
    This module can use SPI or bitbang the SPI transfer using
    the SSD1926 GPIO pins.
    This assumes that the timing controller uses SPI to
    receive Timing Programming data.
    Select the usage using the hardware profile.
*/
    // BB means BitBang
    #define BB_CS                   0x01
    #define BB_SCL                  0x02
    #define BB_SDO                  0x04
    #define BB_DC                   0x08
    #define BB_BL                   0x10

#if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)

    #define TCON_CSLow()            (TCON_CS_LAT = 0)
    #define TCON_CSHigh()           (TCON_CS_LAT = 1)
    #define TCON_CLKLow()           (TCON_SCL_LAT = 0)
    #define TCON_CLKHigh()          (TCON_SCL_LAT = 1)
    #define TCON_DataLow()          (TCON_SDO_LAT = 0)
    #define TCON_DataHigh()         (TCON_SDO_LAT = 1)

    #define TCON_SetCommand()       (TCON_DC_LAT = 0)
    #define TCON_SetData()          (TCON_DC_LAT = 1)

    // set the IOs used to outputs. and initialize them all to "1"
    // set up the TRIS first for outputs, then set the pins
    // to digital in case it is needed and initialize the signals
    // to all high.
    #define InitBitBangedIO() {                     \
                                TCON_CS_TRIS = 0;   \
                                TCON_SCL_TRIS = 0;  \
                                TCON_SDO_TRIS = 0;  \
                                TCON_DC_TRIS = 0;   \
                                TCON_CS_DIG();      \
                                TCON_SCL_DIG();     \
                                TCON_SDO_DIG();     \
                                TCON_DC_DIG();      \
                                TCON_CSHigh();      \
                                TCON_CLKHigh();     \
                                TCON_DataHigh();    \
                                TCON_SetData();     \
                              }

// end of #if defined (USE_DISPLAY_CONTROLLER_MCHP_DA210)
```

```
#elif defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)

    // use the bitbang version using SSD1926 GPIO pins

    #define TCON_CSLow()            (GfxTconSetIO(BB_CS, 0))
    #define TCON_CSHigh()           (GfxTconSetIO(BB_CS, 1))
    #define TCON_CLKLow()           (GfxTconSetIO(BB_SCL, 0))
    #define TCON_CLKHigh()          (GfxTconSetIO(BB_SCL, 1))
    #define TCON_DataLow()          (GfxTconSetIO(BB_SDO, 0))
    #define TCON_DataHigh()         (GfxTconSetIO(BB_SDO, 1))

    #define TCON_SetCommand()       (GfxTconSetIO(BB_DC, 0))
    #define TCON_SetData()          (GfxTconSetIO(BB_DC, 1))

    // this is only needed here since the controller IO's are used
    // instead of the IO's from the PIC device.
    #define SetCtrlBitBangedIO(addr, data)    (SetReg(addr, data))

    // set the GPIO of SSD1926 to as outputs. (used for SSD1289 TCON signals)
    // and initialize them all to "1"
    #define InitBitBangedIO() {                                     \
                                SetCtrlBitBangedIO(0xA8, 0x1F);     \
                                SetCtrlBitBangedIO(0xAC, 0x1F);     \
                              }
// end of #elif defined (USE_TCON_SSD1289)

#elif defined (GFX_USE_DISPLAY_CONTROLLER_S1D13517)

     // use the bitbang version using SSD1926 GPIO pins

    #define TCON_CSLow()            (GfxTconSetIO(CS, 0))
    #define TCON_CSHigh()           (GfxTconSetIO(CS, 1))
    #define TCON_CLKLow()           (GfxTconSetIO(SCL, 0))
    #define TCON_CLKHigh()          (GfxTconSetIO(SCL, 1))
    #define TCON_DataLow()          (GfxTconSetIO(SDO, 0))
    #define TCON_DataHigh()         (GfxTconSetIO(SDO, 1))

    #define TCON_SetCommand()       (GfxTconSetIO(DC, 0))
    #define TCON_SetData()          (GfxTconSetIO(DC, 1))

    // this is only needed here since the controller IO's are used
    // instead of the IO's from the PIC device.
    #define SetCtrlBitBangedIO(addr, data)    (SetReg(addr, data))

    // set the GPIO of SSD1926 to as outputs. (used for SSD1289 TCON signals)
    // and initialize them all to "1"
    #define InitBitBangedIO() {                                     \
                                SCL_TRIS = 0;                       \
                                SDO_TRIS = 0;                       \
                              }

// end of #elif defined (USE_DISPLAY_CONTROLLER_S1D13517)

#endif

/***************************************************************************
* Function: void GfxTconSetIO(BYTE mask, BYTE
level)
*
* Overview: This sets the IO specified by mask to the value set by
*           level.
*
* Input: mask - specifies the IO to be toggles.
*        level - specifies the logic where the IO will be set.
*
* Output: none
*
***************************************************************************/
void GfxTconSetIO(BYTE mask, BYTE level)
{
  #if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)
```

```
    switch(mask)
    {
        case BB_CS:    (level == 1 ? TCON_CSHigh() : TCON_CSLow());
                    break;

        case BB_SCL:   (level == 1 ? TCON_CLKHigh() : TCON_CLKLow());
                    break;

        case BB_SDO:   (level == 1 ? TCON_DataHigh() : TCON_DataLow());
                    break;

        case BB_DC:    (level == 1 ? TCON_SetData() : TCON_SetCommand());
                    break;
        default:
                    break;
    }

    Nop();

  #elif defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)

    static BYTE value = 0xFF;

    if(level == 0)
    {
        value &= ~mask;
    }
    else
    {
        value |= mask;
    }

    SetCtrlBitBangedIO(0xAC, value);

#elif defined (GFX_USE_DISPLAY_CONTROLLER_S1D13517)

    switch(mask)
    {
        case CS:
                    temp = GetReg(REG6E_GPO_1);
                    if(level == 1)  temp  |= 0x02;
                    else            temp  &= 0xFD;
                    SetReg(REG6E_GPO_1,temp);
                    break;

        case SCL:   SCL_PORT = level;
                    break;

        case SDO:   SDO_PORT = level;
                    break;

        case DC:
                    temp = GetReg(REG6E_GPO_1);
                    if(level == 1)  temp  |= 0x04;
                    else            temp  &= 0xFB;
                    SetReg(REG6E_GPO_1,temp);
                    break;
    }

    Nop();

#else

    #error "This TCON_SSD1289 is written for SSD1926, S1D13517 and Microchip Graphics
Module driver. If you are not using those drivers you may need to re-write this driver and
remove this error message."


#endif

}
```

```
/*************************************************************************
* Function: void GfxTconWriteByte(BYTE value)
*
* Overview: This writes a bytes to SPI.
*
* Input: value - the byte data to be written.
*
* Output: none
*
*************************************************************************/
void GfxTconWriteByte(BYTE value)
{
    BYTE    mask;

    mask = 0x80;
    while(mask)
    {
        GfxTconSetIO(BB_SCL, 0);
        Delay10us(1);
        if(mask & value)
        {
            GfxTconSetIO(BB_SDO, 1);
        }
        else
        {
            GfxTconSetIO(BB_SDO, 0);
        }

        GfxTconSetIO(BB_SCL, 1);
        mask >>= 1;
    }
}

/*************************************************************************
* Function: void GfxTconWriteCommand(BYTE index, WORD
value)
*
* Overview: This writes a word to SPI by calling the write byte
*           routine.
*
* Input: index - The index (or address) of the register to be written.
*        value - The value that will be written to the register.
*
* Output: none
*
*************************************************************************/
void GfxTconWriteCommand(WORD index, WORD value)
{
    GfxTconSetIO(BB_CS, 0);

    // Index
    GfxTconSetIO(BB_DC, 0);
    GfxTconWriteByte(((WORD_VAL) index).v[1]);
    GfxTconWriteByte(((WORD_VAL) index).v[0]);

    GfxTconSetIO(BB_CS, 1);
    Delay10us(1);
    GfxTconSetIO(BB_CS, 0);

    // Data
    GfxTconSetIO(BB_DC, 1);
    GfxTconWriteByte(((WORD_VAL) value).v[1]);
    GfxTconWriteByte(((WORD_VAL) value).v[0]);
    GfxTconSetIO(BB_CS, 1);
    Delay10us(1);
}

/*************************************************************************
* Function: void GfxTconInit(void)
*
* Overview: Initialize the IOs to implement Bitbanged SPI interface
```

```
*           to connect to the Timing Controller through SPI.
*
* Input: none
*
* Output: none
*
***************************************************************************/
void GfxTconInit(void)
{
    InitBitBangedIO();

    GfxTconWriteCommand(0x0028, 0x0006);
    GfxTconWriteCommand(0x0000, 0x0001);
    DelayMs(15);

    GfxTconWriteCommand(0x002B, 0x9532);
    GfxTconWriteCommand(0x0003, 0xAAAC);
    GfxTconWriteCommand(0x000C, 0x0002);
    GfxTconWriteCommand(0x000D, 0x000A);
    GfxTconWriteCommand(0x000E, 0x2C00);
    GfxTconWriteCommand(0x001E, 0x00AA);
    GfxTconWriteCommand(0x0025, 0x8000);
    DelayMs(15);

    GfxTconWriteCommand(0x0001, 0x2B3F);
    GfxTconWriteCommand(0x0002, 0x0600);
    GfxTconWriteCommand(0x0010, 0x0000);
    DelayMs(20);

    GfxTconWriteCommand(0x0005, 0x0000);
    GfxTconWriteCommand(0x0006, 0x0000);


    GfxTconWriteCommand(0x0016, 0xEF1C);
    GfxTconWriteCommand(0x0017, 0x0003);
    GfxTconWriteCommand(0x0007, 0x0233);
    GfxTconWriteCommand(0x000B, 0x5312);
    GfxTconWriteCommand(0x000F, 0x0000);
    DelayMs(20);

    GfxTconWriteCommand(0x0041, 0x0000);
    GfxTconWriteCommand(0x0042, 0x0000);
    GfxTconWriteCommand(0x0048, 0x0000);
    GfxTconWriteCommand(0x0049, 0x013F);
    GfxTconWriteCommand(0x0044, 0xEF00);
    GfxTconWriteCommand(0x0045, 0x0000);
    GfxTconWriteCommand(0x0046, 0x013F);
    GfxTconWriteCommand(0x004A, 0x0000);
    GfxTconWriteCommand(0x004B, 0x0000);
    DelayMs(20);

    GfxTconWriteCommand(0x0030, 0x0707);
    GfxTconWriteCommand(0x0031, 0x0704);
    GfxTconWriteCommand(0x0032, 0x0204);
    GfxTconWriteCommand(0x0033, 0x0201);
    GfxTconWriteCommand(0x0034, 0x0203);
    GfxTconWriteCommand(0x0035, 0x0204);
    GfxTconWriteCommand(0x0036, 0x0204);
    GfxTconWriteCommand(0x0037, 0x0502);
    GfxTconWriteCommand(0x003A, 0x0302);
    GfxTconWriteCommand(0x003B, 0x0500);
    DelayMs(20);

    TCON_CLKLow();
}

#endif // #ifdef USE_TCON_SSD1289
```

14.3.15 TCON_HX8257.c

This is file TCON_HX8257.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Himax. HX8257 TCON  driver
 *****************************************************************************
 * FileName:        TCON_HX8257.c
 * Processor:       PIC24, PIC32
 * Compiler:        MPLAB C30, MPLAB C32
 * 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 01/12/09
 * 04/13/11     - Added GFX_USE_DISPLAY_CONTROLLER_xxx check.
 *              - Replaced TCON_Delay() with Delay10us() from TimeDelay.h
 *              - Renamed TCON_Init() to GfxTconInit() and the rest
 *                of the internal functions.
 *              - Graphics Library Version 3.00 Support
 *****************************************************************************/
#include "GraphicsConfig.h"
#include "HardwareProfile.h"
#include "TimeDelay.h"

#ifdef USE_TCON_HX8257

#if defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)
    #include "Graphics/SSD1926.h"
#elif defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)
    // do not need any driver header file
#else
    #warning "This Timing Controller Driver is written for SSD1926, Microchip Display
Controller using DMA and Microchip Graphics Module driver. If you are not using those
drivers you may need to re-write this driver."
#endif

/*
    This module can use SPI or bitbang the SPI transfer using
```

```
    the SSD1926 GPIO pins.
    This assumes that the timing controller uses SPI to
    receive Timing Programming data.
    Select the usage using the hardware profile.
*/
    // BB means BitBang
    #define BB_CS                   0x01
    #define BB_SCL                  0x02
    #define BB_SDO                  0x04
    #define BB_DC                   0x08

#if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)

    #define TCON_CSLow()            (TCON_CS_LAT = 0)
    #define TCON_CSHigh()           (TCON_CS_LAT = 1)
    #define TCON_CLKLow()           (TCON_SCL_LAT = 0)
    #define TCON_CLKHigh()          (TCON_SCL_LAT = 1)
    #define TCON_DataLow()          (TCON_SDO_LAT = 0)
    #define TCON_DataHigh()         (TCON_SDO_LAT = 1)

    #define TCON_SetCommand()       (TCON_DC_LAT = 0)
    #define TCON_SetData()          (TCON_DC_LAT = 1)

    // set the IOs used to outputs. and initialize them all to "1"
    // set up the TRIS first for outputs, then set the pins
    // to digital in case it is needed and initialize the signals
    // to all high.
    #define InitBitBangedIO() {                     \
                                TCON_CS_TRIS = 0;   \
                                TCON_SCL_TRIS = 0;  \
                                TCON_SDO_TRIS = 0;  \
                                TCON_DC_TRIS = 0;   \
                                TCON_CS_DIG();      \
                                TCON_SCL_DIG();     \
                                TCON_SDO_DIG();     \
                                TCON_DC_DIG();      \
                                TCON_CSHigh();      \
                                TCON_CLKHigh();     \
                                TCON_DataHigh();    \
                                TCON_SetData();     \
                              }

// end of #if defined (USE_DISPLAY_CONTROLLER_MCHP_DA210)


#elif defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)

    // use the bitbang version using SSD1926 GPIO pins

    #define TCON_CSLow()            (GfxTconSetIO(CS, 0))
    #define TCON_CSHigh()           (GfxTconSetIO(CS, 1))
    #define TCON_CLKLow()           (GfxTconSetIO(SCL, 0))
    #define TCON_CLKHigh()          (GfxTconSetIO(SCL, 1))
    #define TCON_DataLow()          (GfxTconSetIO(SDO, 0))
    #define TCON_DataHigh()         (GfxTconSetIO(SDO, 1))

    #define TCON_SetCommand()       (GfxTconSetIO(DC, 0))
    #define TCON_SetData()          (GfxTconSetIO(DC, 1))

    // this is only needed here since the controller IO's are used
    // instead of the IO's from the PIC device.
    #define SetCtrlBitBangedIO(addr, data)    (SetReg(addr, data))

    // set the GPIO of SSD1926 to as outputs. (used for TCON signals)
    // and initialize them all to "1"
    #define InitBitBangedIO() {                                 \
                                SetCtrlBitBangedIO(0xA8, 0x1F);     \
                                SetCtrlBitBangedIO(0xAC, 0x1F);     \
                              }
// end of #elif defined (USE_TCON_SSD1289)
```

```
#endif


/***************************************************************************
* Function: void GfxTconSetIO(BYTE mask, BYTE
level)
*
* Overview: This sets the IO specified by mask to the value set by
*           level.
*
* Input: mask - specifies the IO to be toggles.
*        level - specifies the logic where the IO will be set.
*
* Output: none
*
***************************************************************************/
void GfxTconSetIO(BYTE mask, BYTE level)
{

 #if defined (GFX_USE_DISPLAY_CONTROLLER_MCHP_DA210) || defined
(GFX_USE_DISPLAY_CONTROLLER_DMA)

    switch(mask)
    {
        case BB_CS:    (level == 1 ? TCON_CSHigh() : TCON_CSLow());
                    break;

        case BB_SCL:   (level == 1 ? TCON_CLKHigh() : TCON_CLKLow());
                    break;

        case BB_SDO:   (level == 1 ? TCON_DataHigh() : TCON_DataLow());
                    break;

        case BB_DC:    (level == 1 ? TCON_SetData() : TCON_SetCommand());
                    break;
        default:
                    break;
    }

    Nop();

 #elif defined (GFX_USE_DISPLAY_CONTROLLER_SSD1926)

    static BYTE value = 0xFF;

    if(level == 0)
    {
        value &= ~mask;
    }
    else
    {
        value |= mask;
    }

    SetCtrlBitBangedIO(0xAC, value);

#endif

}
/***************************************************************************
* Function: void GfxTconWriteByte(BYTE value)
*
* Overview: This writes a bytes to SPI.
*
* Input: value - the byte data to be written.
*
* Output: none
*
***************************************************************************/
void GfxTconWriteByte(BYTE value)
{
    BYTE    mask;
```

```
    mask = 0x80;
    while(mask)
    {
        GfxTconSetIO(BB_SCL, 0);
        Delay10us(1);
        if(mask & value)
        {
            GfxTconSetIO(BB_SDO, 1);
        }
        else
        {
            GfxTconSetIO(BB_SDO, 0);
        }

        GfxTconSetIO(BB_SCL, 1);
        mask >>= 1;
    }
}

/***************************************************************************
* Function: void GfxTconWriteCommand(BYTE index, WORD
value)
*
* Overview: This writes a word to SPI by calling the write byte
*           routine.
*
* Input: index - The index (or address) of the register to be written.
*        value - The value that will be written to the register.
*
* Output: none
*
***************************************************************************/
void GfxTconWriteCommand(WORD index, WORD value)
{
    TCON_CTRL(BB_CS, 0);

    // Index
    TCONWriteByte(0x70);
    TCONWriteByte(((WORD_VAL) index).v[1]);
    TCONWriteByte(((WORD_VAL) index).v[0]);

    TCON_CTRL(BB_CS, 1);
    TCON_Delay();
    TCON_CTRL(BB_CS, 0);

    // Data
    TCONWriteByte(0x72);
    TCONWriteByte(((WORD_VAL) value).v[1]);
    TCONWriteByte(((WORD_VAL) value).v[0]);
    TCON_CTRL(BB_CS, 1);
    TCON_Delay();
}

/***************************************************************************
* Function: void GfxTconInit(void)
*
* Overview: Initialize the IOs to implement Bitbanged SPI interface
*           to connect to the Timing Controller through SPI.
*
* Input: none
*
* Output: none
*
***************************************************************************/
void GfxTconInit(void)
{
    InitBitBangedIO();

    DelayMs(20);
    GPIO_TCON(0x0006, 0x2020);
    DelayMs(20);
```

```
}

#endif // #ifdef USE_TCON_HX8257
```

14.3.16 CustomDisplayDriver.c

This is file CustomDisplayDriver.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Custom display controller driver template
 *****************************************************************************
 * FileName:        CustomDisplayDriver.c
 * 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *****************************************************************************/
#include "Graphics/Graphics.h"

// Color
WORD_VAL    _color;

// Clipping region control
SHORT       _clipRgn;

// Clipping region borders
SHORT       _clipLeft;
SHORT       _clipTop;
SHORT       _clipRight;
SHORT       _clipBottom;

/*********************************************************************
* Function:  void  DelayMs(WORD time)
*
* PreCondition: none
```

```
*
* Input: time - delay in ms
*
* Output: none
*
* Side Effects: none
*
* Overview: delays execution on time specified in ms
*
* Note: none
*
*********************************************************************/
#ifdef __PIC32MX

/* */
void DelayMs(WORD time)
{
    while(time--)
    {
        unsigned int    int_status;

        int_status = INTDisableInterrupts();
        OpenCoreTimer(GetSystemClock() / 2000); // core timer is at 1/2 system clock
        INTRestoreInterrupts(int_status);

        mCTClearIntFlag();

        while(!mCTGetIntFlag());
    }

    mCTClearIntFlag();
}

#else
    #define DELAY_1MS   16000 / 5               // for 16MIPS

/* */
void DelayMs(WORD time)
{
    unsigned    delay;
    while(time--)
        for(delay = 0; delay < DELAY_1MS; delay++);
}

#endif

/*********************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*********************************************************************/
void ResetDevice(void)
{ }

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: x,y - pixel coordinates
*
```

```
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: puts pixel
 *
 * Note: none
 *
 ********************************************************************/
void PutPixel(SHORT x, SHORT y)
{ }

/********************************************************************
 * Function: WORD GetPixel(SHORT x, SHORT y)
 *
 * PreCondition: none
 *
 * Input: x,y - pixel coordinates
 *
 * Output: pixel color
 *
 * Side Effects: none
 *
 * Overview: returns pixel color at x,y position
 *
 * Note: none
 *
 ********************************************************************/
WORD GetPixel(SHORT x, SHORT y)
{
    return (0);
}
```

14.3.17 UC1610.c

This is file UC1610.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  UltraChip UC1610 controller driver
 *****************************************************************************
 * FileName:        UC1610.c
 * Dependencies:    Graphics.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2009 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 02/25/09     ...
 * 04/12/11     Graphics Library Version 3.00 Support
 ****************************************************************************/
#include "HardwareProfile.h"

#if defined (GFX_USE_DISPLAY_CONTROLLER_UC1610)

#include "Compiler.h"
#include "Graphics/DisplayDriver.h"
#include "Graphics/UC1610.h"
#include "Graphics/Primitive.h"

#if defined (USE_GFX_PMP)
    #include "Graphics/gfxpmp.h"
#elif defined (USE_GFX_EPMP)
    #include "Graphics/gfxepmp.h"
#endif

// Unsupported Graphics Library Features
#ifdef USE_TRANSPARENT_COLOR
    #warning "This driver does not support the transparent feature on PutImage(). Build
will use the PutImage() functions defind in the Primitive.c"
#endif

// Color
GFX_COLOR   _color;
#ifdef USE_TRANSPARENT_COLOR
GFX_COLOR   _colorTransparent;
SHORT       _colorTransparentEnable;
#endif

// Clipping region control
SHORT   _clipRgn;

// Clipping region borders
SHORT   _clipLeft;
SHORT   _clipTop;
SHORT   _clipRight;
SHORT   _clipBottom;

////////////////////// LOCAL FUNCTIONS PROTOTYPES ///////////////////////////
#ifndef USE_TRANSPARENT_COLOR
void    PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void    PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch);
void    PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
void    PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch);
#endif

/*******************************************************************
* Function:  void ResetDevice()
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: resets LCD, initializes PMP
*
* Note: none
*
*******************************************************************/
```

```
void ResetDevice(void)
{
    // Initialize the device
    DriverInterfaceInit();

    DisplayEnable();

    DisplaySetCommand();
    // set Vbias
    DeviceWrite(CMD_BIAS_RATIO_12);
    DeviceWrite(CMD_CONTRAST);
    DeviceWrite(82);

    // set panel loading
    DeviceWrite(CMD_PANEL_LOADING_38NF);

    // set connected COM electrodes
    DeviceWrite(CMD_DISPLAY_START);
    DeviceWrite(4 * (DISP_START_PAGE + 1) - 1);
    DeviceWrite(CMD_DISPLAY_END);
    DeviceWrite(4 * (DISP_END_PAGE + 1) - 1);
    DeviceWrite(CMD_COM_END);
    DeviceWrite(4 * (DISP_END_PAGE + 1) - 1);

    // set line rate
    DeviceWrite(CMD_LINE_RATE_12KLPS);

    DeviceWrite(CMD_MAPPING_MY);

    // inverse color
    DeviceWrite(CMD_INVERSE_ON);

    // set programm window
    DeviceWrite(CMD_WND_PRG_DISABLE);
    DeviceWrite(CMD_START_COLUMN);
    DeviceWrite(DISP_START_COLUMN);
    DeviceWrite(CMD_END_COLUMN);
    DeviceWrite(DISP_END_COLUMN);
    DeviceWrite(CMD_START_PAGE);
    DeviceWrite(DISP_START_PAGE);
    DeviceWrite(CMD_END_PAGE);
    DeviceWrite(DISP_END_PAGE);
    DeviceWrite(CMD_WND_PRG_ENABLE);

    // set autoincrement
    DeviceWrite(CMD_RAM_ADDR_INCR_ON);

    DeviceWrite(CMD_DISPLAY_ON);

    DisplaySetData();

    DisplayDisable();
}

/*********************************************************************
* Function: void ContrastSet(BYTE contrast)
*
* PreCondition: none
*
* Input: contrast value
*
* Output: none
*
* Side Effects: none
*
* Overview: sets contrast
*
* Note: none
*
*********************************************************************/
void ContrastSet(BYTE contrast)
{
```

```
    DisplayEnable();
    DisplaySetCommand();
    DeviceWrite(CMD_CONTRAST);
    DeviceWrite(contrast);
    DisplaySetData();
    DisplayDisable();
}

#ifdef USE_TRANSPARENT_COLOR
/*********************************************************************
* Function:  void TransparentColorEnable(GFX_COLOR color)
*
* Overview: Sets current transparent color.
*
* PreCondition: none
*
* Input: color - Color value chosen.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void TransparentColorEnable(GFX_COLOR color)
{
    _colorTransparent = color;
    _colorTransparentEnable = TRANSPARENT_COLOR_ENABLE;

}
#endif

/*********************************************************************
* Function: void PutPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: pixel position
*
* Output: none
*
* Side Effects: none
*
* Overview: puts pixel with current color at given position
*
* Note: none
*
*********************************************************************/
void PutPixel(SHORT x, SHORT y)
{
    BYTE    value;

    if(_clipRgn)
    {
        if(x < _clipLeft)
            return;
        if(x > _clipRight)
            return;
        if(y < _clipTop)
            return;
        if(y > _clipBottom)
            return;
    }

    x += DISP_START_COLUMN;

    DisplayEnable();

    // set address
    DisplaySetCommand();
    DeviceWrite(CMD_COLUMN_ADDR_LSB | (x & 0x0f));
    DeviceWrite(CMD_COLUMN_ADDR_MSB | ((x >> 4) & 0x0f));
    DeviceWrite(CMD_PAGE_ADDR | (DISP_START_PAGE + (y >> 2)));
```

```
    DisplaySetData();

    // read 4 pixels
    value = DeviceRead();
    value = DeviceRead();

    // set pixel
    switch(y & 0x03)
    {
        case 0: value &= 0b11111100; value |= _color; break;
        case 1: value &= 0b11110011; value |= _color << 2; break;
        case 2: value &= 0b11001111; value |= _color << 4; break;
        case 3: value &= 0b00111111; value |= _color << 6; break;
    }

    // set address
    DisplaySetCommand();
    DeviceWrite(CMD_COLUMN_ADDR_LSB | (x & 0x0f));
    DeviceWrite(CMD_COLUMN_ADDR_MSB | ((x >> 4) & 0x0f));
    DeviceWrite(CMD_PAGE_ADDR | (DISP_START_PAGE + (y >> 2)));
    DisplaySetData();

    // write 4 pixels back
    DeviceWrite(value);

    DisplayDisable();
}

/*********************************************************************
* Function: WORD GetPixel(SHORT x, SHORT y)
*
* PreCondition: none
*
* Input: pixel position
*
* Output: pixel color
*
* Side Effects: none
*
* Overview: returns pixel at given position
*
* Note: none
*
*********************************************************************/
GFX_COLOR GetPixel(SHORT x, SHORT y)
{
    BYTE    value;

    x += DISP_START_COLUMN;

    DisplayEnable();

    // set address
    DisplaySetCommand();
    DeviceWrite(CMD_COLUMN_ADDR_LSB | (x & 0x0f));
    DeviceWrite(CMD_COLUMN_ADDR_MSB | ((x >> 4) & 0x0f));
    DeviceWrite(CMD_PAGE_ADDR | (DISP_START_PAGE + (y >> 2)));
    DisplaySetData();

    // read 4 pixels
    value = DeviceRead();
    value = DeviceRead();

    DisplayDisable();

    // get pixel
    switch(y & 0x03)
    {
        case 0: break;
        case 1: value >>= 2; break;
        case 2: value >>= 4; break;
        case 3: value >>= 6; break;
```

```
    }

    value &= 0x03;
    return ((GFX_COLOR)value);
}

/*********************************************************************
* Function: SetClipRgn(left, top, right, bottom)
*
* Overview: Sets clipping region.
*
* PreCondition: none
*
* Input: left - Defines the left clipping region border.
*        top - Defines the top clipping region border.
*        right - Defines the right clipping region border.
*        bottom - Defines the bottom clipping region border.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClipRgn(SHORT left, SHORT top, SHORT right, SHORT bottom)
{
    _clipLeft=left;
    _clipTop=top;
    _clipRight=right;
    _clipBottom=bottom;

}

/*********************************************************************
* Function: SetClip(control)
*
* Overview: Enables/disables clipping.
*
* PreCondition: none
*
* Input: control - Enables or disables the clipping.
*           - 0: Disable clipping
*           - 1: Enable clipping
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
void SetClip(BYTE control)
{
    _clipRgn=control;
}

/*********************************************************************
* Function: IsDeviceBusy()
*
* Overview: Returns non-zero if LCD controller is busy
*           (previous drawing operation is not completed).
*
* PreCondition: none
*
* Input: none
*
* Output: Busy status.
*
* Side Effects: none
*
*********************************************************************/
WORD IsDeviceBusy(void)
{
    return (0);
}
```

```
/*****************************************************************
* Function: void ClearDevice(void)
*
* PreCondition: none
*
* Input: none
*
* Output: none
*
* Side Effects: none
*
* Overview: clears screen with current color
*
* Note: none
*
*****************************************************************/
void ClearDevice(void)
{
    WORD    counter;
    BYTE    pattern;

    pattern = _color;
    pattern |= pattern << 2;
    pattern |= pattern << 4;

    DisplayEnable();
    DisplaySetCommand();
    DeviceWrite(CMD_COLUMN_ADDR_LSB | (DISP_START_COLUMN & 0x0f));
    DeviceWrite(CMD_COLUMN_ADDR_MSB | ((DISP_START_COLUMN >> 4) & 0x0f));
    DeviceWrite(CMD_PAGE_ADDR | DISP_START_PAGE);
    DisplaySetData();

    for(counter = 0; counter < (DWORD) (GetMaxX() + 1) * (GetMaxY() + 1) / 4; counter++)
    {
        DeviceWrite(pattern);
    }

    DisplayDisable();
}

#ifndef USE_TRANSPARENT_COLOR

#ifdef USE_BITMAP_FLASH

/*****************************************************************
* Function: void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner,
*        image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in flash
*
*****************************************************************/
void PutImage1BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    BYTE                temp;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                cx, cy;
    BYTE                stretchX, stretchY;
```

```
    BYTE                palette[2];
    BYTE                mask;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    palette[0] = (BYTE) * (flashAddress + 1);
    palette[0] >>= 3;
    palette[0] &= 0x03;
    flashAddress += 2;
    palette[1] = (BYTE) * (flashAddress + 1);
    palette[1] >>= 3;
    palette[1] &= 0x03;
    flashAddress += 2;

    cy = top;
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            cx = left;
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *flashAddress;
                    flashAddress++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(palette[1]);
                }
                else
                {
                    SetColor(palette[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    PutPixel(cx++, cy);
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            cy++;
        }
    }
}

/*****************************************************************************
* Function: void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
```

```
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Overview: outputs 16 color image starting from left,top coordinates
 *
 * Note: image must be located in flash
 *
 ***********************************************************************/
void PutImage4BPP(SHORT left, SHORT top, FLASH_BYTE *image, BYTE stretch)
{
    register FLASH_BYTE *flashAddress;
    register FLASH_BYTE *tempFlashAddress;
    WORD                sizeX, sizeY;
    WORD                x, y;
    WORD                cx, cy;
    BYTE                temp;
    register BYTE       stretchX, stretchY;
    BYTE                palette[16];
    WORD                counter;

    // Move pointer to size information
    flashAddress = image + 2;

    // Read image size
    sizeY = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;
    sizeX = *((FLASH_WORD *)flashAddress);
    flashAddress += 2;

    // Read pallete
    for(counter = 0; counter < 16; counter++)
    {
        palette[counter] = (BYTE) * (flashAddress + 1);
        palette[counter] >>= 3;
        palette[counter] &= 0x03;
        flashAddress += 2;
    }

    cy = top;
    for(y = 0; y < sizeY; y++)
    {
        tempFlashAddress = flashAddress;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            flashAddress = tempFlashAddress;
            cx = left;
            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(palette[temp >> 4]);
                }
                else
                {
                    temp = *flashAddress;
                    flashAddress++;

                    // first pixel in byte
                    SetColor(palette[temp & 0x0f]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    PutPixel(cx++, cy);
                }
```

```
            }

            cy++;
        }
    }
}

#endif
#ifdef USE_BITMAP_EXTERNAL

/*****************************************************************
* Function: void PutImage1BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*****************************************************************/
void PutImage1BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            pallete[2];
    BYTE            lineBuffer[((GetMaxX() + 1) / 8) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp;
    BYTE            mask;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            cx, cy;
    BYTE            stretchX, stretchY;

    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (2 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 2 * sizeof(WORD), palette);

    palette[0] >>= 3;
    palette[0] &= 0x03;
    palette[1] >>= 3;
    palette[1] &= 0x03;

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 2 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 3;
    if(bmp.width & 0x0007)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    cy = top;
    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
```

```
        memOffset += byteWidth;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            cx = left;
            mask = 0;
            for(x = 0; x < sizeX; x++)
            {

                // Read 8 pixels from flash
                if(mask == 0)
                {
                    temp = *pData++;
                    mask = 0x80;
                }

                // Set color
                if(mask & temp)
                {
                    SetColor(palette[1]);
                }
                else
                {
                    SetColor(palette[0]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    PutPixel(cx++, cy);
                }

                // Shift to the next pixel
                mask >>= 1;
            }

            cy++;
        }
    }
}

/*************************************************************************
* Function: void PutImage4BPPExt(SHORT left, SHORT top, void* image, BYTE stretch)
*
* PreCondition: none
*
* Input: left,top - left top image corner, image - image pointer,
*        stretch - image stretch factor
*
* Output: none
*
* Side Effects: none
*
* Overview: outputs monochrome image starting from left,top coordinates
*
* Note: image must be located in external memory
*
*************************************************************************/
void PutImage4BPPExt(SHORT left, SHORT top, void *image, BYTE stretch)
{
    register DWORD  memOffset;
    BITMAP_HEADER   bmp;
    WORD            palette[16];
    BYTE            lineBuffer[((GetMaxX() + 1) / 2) + 1];
    BYTE            *pData;
    SHORT           byteWidth;

    BYTE            temp;
    WORD            sizeX, sizeY;
    WORD            x, y;
    WORD            cx, cy;
    BYTE            stretchX, stretchY;
```

```
    // Get image header
    ExternalMemoryCallback(image, 0, sizeof(BITMAP_HEADER), &bmp);

    // Get pallete (16 entries)
    ExternalMemoryCallback(image, sizeof(BITMAP_HEADER), 16 * sizeof(WORD), palette);

    for(temp = 0; temp < 16; temp++)
    {
        palette[temp] >>= 3;
        palette[temp] &= 0x03;
    }

    // Set offset to the image data
    memOffset = sizeof(BITMAP_HEADER) + 16 * sizeof(WORD);

    // Line width in bytes
    byteWidth = bmp.width >> 1;
    if(bmp.width & 0x0001)
        byteWidth++;

    // Get size
    sizeX = bmp.width;
    sizeY = bmp.height;

    cy = top;
    for(y = 0; y < sizeY; y++)
    {

        // Get line
        ExternalMemoryCallback(image, memOffset, byteWidth, lineBuffer);
        memOffset += byteWidth;
        for(stretchY = 0; stretchY < stretch; stretchY++)
        {
            pData = lineBuffer;
            cx = left;

            for(x = 0; x < sizeX; x++)
            {

                // Read 2 pixels from flash
                if(x & 0x0001)
                {

                    // second pixel in byte
                    SetColor(palette[temp >> 4]);
                }
                else
                {
                    temp = *pData++;

                    // first pixel in byte
                    SetColor(palette[temp & 0x0f]);
                }

                // Write pixel to screen
                for(stretchX = 0; stretchX < stretch; stretchX++)
                {
                    PutPixel(cx++, cy);
                }
            }

            cy++;
        }
    }
}

#endif

#endif //#ifndef USE_TRANSPARENT_COLOR

#endif //#if defined (GFX_USE_DISPLAY_CONTROLLER_UC1610)
```

14.3.18 UC1610.h

This is file UC1610.h.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  UltraChip UC1610 LCD controllers driver
 *****************************************************************************
 * FileName:        UC1610.h
 * Dependencies:    p24Fxxxx.h
 * Processor:       PIC24
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * Company:         Microchip Technology Incorporated
 *
 * Software License Agreement
 *
 * Copyright © 2009 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
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * 02/25/09     ...
 * 04/12/11     Graphics Library Version 3.00 Support
 *****************************************************************************/
#ifndef _UC1610_H
    #define _UC1610_H

    #include "HardwareProfile.h"
    #include "GraphicsConfig.h"
    #include "GenericTypeDefs.h"

    #ifdef USE_16BIT_PMP
        #error This driver doesn't support 16-bit PMP (remove USE_16BIT_PMP option from
HardwareProfile.h)
    #endif
    #ifndef DISP_HOR_RESOLUTION
        #error DISP_HOR_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef DISP_VER_RESOLUTION
        #error DISP_VER_RESOLUTION must be defined in HardwareProfile.h
    #endif
    #ifndef COLOR_DEPTH
        #error COLOR_DEPTH must be defined in GraphicsConfig.h
    #endif
```

```
    #ifndef DISP_ORIENTATION
        #error DISP_ORIENTATION must be defined in HardwareProfile.h
    #endif

/*********************************************************************
* Overview: Horizontal and vertical screen size.
*********************************************************************/
    #if (DISP_HOR_RESOLUTION != 160)
        #error This driver doesn't supports this resolution. Horizontal resolution must be
160 pixels.
    #endif
    #if (DISP_VER_RESOLUTION != 100)
        #error This driver doesn't supports this resolution. Vertical resolution must be
100 pixels.
    #endif

/*********************************************************************
* Overview: Display orientation.
*********************************************************************/
    #if (DISP_ORIENTATION != 0)
        #error This driver doesn't support this orientation. It must be 0.
    #endif

/*********************************************************************
* Overview: Color depth.
*********************************************************************/
    #if (COLOR_DEPTH != 8)
        #error This driver doesn't support the selected color depth. It should be 8 but the
device uses only 2 bits. The 6 MSbits will be ignored.
    #endif
    #define CMD_DISPLAY_ON          0b10101111
    #define CMD_DISPLAY_OFF         0b10101110

    #define CMD_INVERSE_ON          0b10100111
    #define CMD_INVERSE_OFF         0b10100110

    #define CMD_PANEL_LOADING_16NF  0b00101000
    #define CMD_PANEL_LOADING_21NF  0b00101001
    #define CMD_PANEL_LOADING_28NF  0b00101010
    #define CMD_PANEL_LOADING_38NF  0b00101011

    #define CMD_LINE_RATE_12KLPS    0b10100000
    #define CMD_LINE_RATE_13KLPS    0b10100001
    #define CMD_LINE_RATE_14KLPS    0b10100010
    #define CMD_LINE_RATE_16KLPS    0b10100011

    #define CMD_DISPLAY_START       0b11110010
    #define CMD_DISPLAY_END         0b11110011
    #define CMD_COM_END             0b11110001

    #define CMD_BIAS_RATIO_5        0b11101000
    #define CMD_BIAS_RATIO_10       0b11101001
    #define CMD_BIAS_RATIO_11       0b11101010
    #define CMD_BIAS_RATIO_12       0b11101011

    #define CMD_CONTRAST            0b10000001

    #define CMD_RAM_ADDR_INCR_OFF   0b10001000
    #define CMD_RAM_ADDR_INCR_ON    0b10001101

    #define CMD_MAPPING_CTRL        0b11000000
    #define CMD_MAPPING_MY          0b11000100
    #define CMD_MAPPING_MX          0b11000010

    #define CMD_WND_PRG_DISABLE     0b11111000
    #define CMD_WND_PRG_ENABLE      0b11111001
    #define CMD_START_COLUMN        0b11110100
    #define CMD_START_PAGE          0b11110101
    #define CMD_END_COLUMN          0b11110110
    #define CMD_END_PAGE            0b11110111

    #define CMD_COLUMN_ADDR_LSB     0b00000000
```

```
    #define CMD_COLUMN_ADDR_MSB     0b00010000
    #define CMD_PAGE_ADDR           0b01100000

    #define DISP_START_COLUMN       0
    #define DISP_START_PAGE         7
    #define DISP_END_COLUMN         159
    #define DISP_END_PAGE           31

/*********************************************************************
* Macros: SetPalette(colorNum, color)
*
* Overview:  Sets palette register.
*
* PreCondition: none
*
* Input: colorNum - Register number.
*        color - Color.
*
* Output: none
*
* Side Effects: none
*
*********************************************************************/
    #define SetPalette(colorNum, color)

/*********************************************************************
* Function: void ContrastSet(BYTE contrast)
*
* PreCondition: none
*
* Input: contrast value
*
* Output: none
*
* Side Effects: none
*
* Overview: sets contrast
*
* Note: none
*
*********************************************************************/
void    ContrastSet(BYTE contrast);
#endif // _UC1610_H
```

14.3.19 TCON_Custom.c

This is file TCON_Custom.c.

Body Source

```
/*****************************************************************************
 *  Module for Microchip Graphics Library
 *  Custom TCON controller driver
 *****************************************************************************
 * FileName:        TCON_Custom.c
 * 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.
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Anton Alkhimenok     11/17/08
 ****************************************************************************/
#include "Graphics/Graphics.h"

/* */

void TCON_Init(void)
{ }
```

15 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).
8. Microchip Technical Brief TB3012 - "Conversion of Graphics PICtail Plus Board 2 for Compatibility with USB PICtail Plus" (DS93012), Microchip Technology Incorporated.

Index

_

_BMPDECODER structure 512
_bottom variable 443
_delay_ms variable 444
_destpageaddr variable 444
_GIFDECODER structure 512
_JPEGDECODER structure 513
_left variable 444
_page variable 445
_param1 variable 445
_param2 variable 445
_pDefaultGolScheme variable 334
_pGolObjects variable 334
_pObjectFocused variable 335
_right variable 445
_srcpageaddr variable 446
_top variable 446
_transitionpending variable 446
_type variable 447

A

AC_DISABLED macro 55
AC_DRAW macro 55
AC_HIDE macro 55
AC_PRESSED macro 55
AC_TICK macro 56
AcCreate function 56
AcDraw function 58
AcHandsDraw function 59
AcMsgDefault function 61
AcSetHour function 60
AcSetMinute function 60
AcSetSecond function 61
AcTranslateMsg function 62
Adding New Device Driver 397
Advanced Display Driver Features 399
Alpha Blending 399
AlphaBlendWindow function 400
Analog Clock 53
Analog Clock States 54
ANALOGCLOCK structure 64
AnalogClock.c 625
AnalogClock.h 637
Application Configuration 41
Arc function 356

B

Bar function 352
Bevel function 357
Bitmap Functions 362
Bitmap Settings 364
Bitmap Source 365
BITMAP_HEADER structure 364
BLACK macro 389
BLUE macro 389
BmpDecoder.c 457
BmpDecoder.h 464
BRIGHTBLUE macro 389
BRIGHTCYAN macro 390
BRIGHTGREEN macro 390
BRIGHTMAGENTA macro 390
BRIGHTRED macro 390
BRIGHTYELLOW macro 390
BROWN macro 391
BTN_DISABLED macro 66
BTN_DRAW macro 67
BTN_DRAW_FOCUS macro 67
BTN_FOCUSED macro 67
BTN_HIDE macro 67
BTN_NOPANEL macro 69
BTN_PRESSED macro 67
BTN_TEXTBOTTOM macro 68
BTN_TEXTLEFT macro 68
BTN_TEXTRIGHT macro 68
BTN_TEXTTOP macro 68
BTN_TOGGLE macro 68
BTN_TWOTONE macro 69
BtnCreate function 69
BtnDraw function 70
BtnGetBitmap macro 73
BtnGetText macro 72

BtnMsgDefault function 74
BtnSetBitmap macro 74
BtnSetText function 72
BtnTranslateMsg function 75
Button 65
Button States 66
BUTTON structure 77
Button.c 644
Button.h 655

C

CB_CHECKED macro 117
CB_DISABLED macro 118
CB_DRAW macro 118
CB_DRAW_CHECK macro 118
CB_DRAW_FOCUS macro 118
CB_FOCUSED macro 118
CB_HIDE macro 119
CbCreate function 119
CbDraw function 120
CbGetText macro 121
CbMsgDefault function 122
CbSetText function 121
CbTranslateMsg function 123
CH_3D_ENABLE macro 81
CH_BAR macro 82
CH_BAR_HOR macro 82
CH_CLR0 macro 112
CH_CLR1 macro 112
CH_CLR10 macro 114
CH_CLR11 macro 114
CH_CLR12 macro 114
CH_CLR13 macro 115
CH_CLR14 macro 115
CH_CLR15 macro 115
CH_CLR2 macro 112
CH_CLR3 macro 113
CH_CLR4 macro 113
CH_CLR5 macro 113
CH_CLR6 macro 113
CH_CLR7 macro 113
CH_CLR8 macro 114
CH_CLR9 macro 114
CH_DISABLED macro 81
CH_DONUT macro 82
CH_DRAW macro 81
CH_DRAW_DATA macro 81
CH_HIDE macro 83
CH_LEGEND macro 82
CH_NUMERIC macro 82
CH_PERCENT macro 83
CH_PIE macro 83
CH_VALUE macro 83
ChAddDataSeries function 88
Changing the default Font 522
Chart 77
Chart Examples 84
Chart States 80
CHART structure 110
Chart.c 663
Chart.h 708
CHARTPARAM structure 111
ChCreate function 86
ChDraw function 87
Check Box States 117
Checkbox 115
CHECKBOX structure 124
CheckBox.c 725
CheckBox.h 732
ChFreeDataSeries function 108
ChGetAxisLabelFont macro 106
ChGetColorTable macro 103
ChGetGridLabelFont macro 107
ChGetPercentMax macro 101
ChGetPercentMin macro 102
ChGetPercentRange macro 99
ChGetSampleEnd macro 98
ChGetSampleLabel macro 97
ChGetSampleRange macro 100
ChGetSampleStart macro 97
ChGetShowSeriesCount macro 91
ChGetShowSeriesStatus macro 92
ChGetTitle macro 104
ChGetTitleFont macro 105

ChGetValueLabel macro 93
ChGetValueMax macro 94
ChGetValueMin macro 94
ChGetValueRange macro 95
ChHideSeries macro 91
ChRemoveDataSeries function 89
ChSetAxisLabelFont macro 106
ChSetColorTable macro 102
ChSetGridLabelFont macro 107
ChSetPercentRange function 98
ChSetSampleLabel macro 96
ChSetSampleRange function 99
ChSetTitle macro 103
ChSetTitleFont macro 104
ChSetValueLabel macro 92
ChSetValueRange function 95
ChShowSeries macro 90
ChTranslateMsg function 109
Circle Functions 354
Circle macro 355
ClearDevice function 367
ClearPaletteChangeError function 428
CLIP_DISABLE macro 381
CLIP_ENABLE macro 381
ClrState macro 289
Color Definition 388
Color Table 111
COLOR_DEPTH macro 41
Common Object States 286
Configuration Setting 41
CopyBlock function 384
CopyBlock macro 419
CopyPageWindow function 385, 441
CopyWindow function 386
CustomDisplayDriver.c 1202
CYAN macro 391

D

DARKGRAY macro 391
DASHED_LINE macro 351
Data Series Status Settings 84
DATASERIES structure 110
Decompress function 426
Decompressing DEFLATEd data 426
Default Style Scheme Settings 331
Demo Compatibility Matrix 29
Demo Projects 28
Demo Summary 28
Device Driver Layer 1030
Device Driver Options 44
Dial 124
Dial States 126
Digital Meter 135
Digital Meter States 136
DIGITALMETER structure 143
DigitalMeter.c 736
DigitalMeter.h 742
DISABLED macro 287
DisablePalette function 428
Display Device Driver Control 395
Display Device Driver Layer 373
Display Device Driver Layer Configuration 37
Display Device Driver Level Primitives 373
DisplayBrightness function 383
DM_CENTER_ALIGN macro 137
DM_DISABLED macro 136
DM_DRAW macro 136
DM_FRAME macro 137
DM_HIDE macro 137
DM_RIGHT_ALIGN macro 137
DM_UPDATE macro 137
DmCreate function 138
DmDecVal macro 141
DmDraw function 139
DmGetValue macro 140
DmIncVal macro 141
DmSetValue function 140
DmTranslateMsg function 142
DOTTED_LINE macro 351
Double Buffering 422
DRAW macro 287
DRAW_FOCUS macro 288
DRAW_FUNC type 52
DRAW_UPDATE macro 288

DrawPoly function 354
drvTFT001.c 1031
drvTFT001.h 1057

E

EB_CARET macro 146
EB_CENTER_ALIGN macro 145
EB_DISABLED macro 145
EB_DRAW macro 145
EB_DRAW_CARET macro 146
EB_FOCUSED macro 146
EB_HIDE macro 146
EB_RIGHT_ALIGN macro 146
EbAddChar function 150
EbCreate function 147
EbDeleteChar function 150
EbDraw function 148
EbGetText macro 149
EbMsgDefault function 151
EbSetText function 149
EbTranslateMsg function 152
Edit Box 143
Edit Box States 145
EDITBOX structure 153
EditBox.c 746
EditBox.h 753
EnablePalette function 429
External Memory 365
EXTERNAL_FONT_BUFFER_SIZE macro 366

F

Files 530
FillBevel function 358
FillCircle macro 355
Focus Support Selection 31
FOCUSED macro 287
Font Source Selection 42
Font Type Selection 36
FONT_EXTERNAL type 340
FONT_FLASH structure 340
FONT_HEADER structure 339
FONTDEFAULT variable 332
FREE_FUNC type 53

G

GB_CENTER_ALIGN macro 171
GB_DISABLED macro 172
GB_DRAW macro 172
GB_HIDE macro 172
GB_RIGHT_ALIGN macro 172
GbCreate function 172
GbDraw function 174
GbGetText macro 174
GbSetText function 175
GbTranslateMsg function 176
GetClipBottom macro 379
GetClipLeft macro 379
GetClipRight macro 380
GetClipTop macro 380
GetColor macro 376
GetFontOrientation macro 341
GetImageHeight function 363
GetImageWidth function 363
GetMaxX macro 377
GetMaxY macro 377
GetObjID macro 299
GetObjNext macro 300
GetObjType macro 299
GetPageAddress macro 384
GetPaletteChangeError function 429
GetPixel function 374
GetPositionAddress function 401
GetState macro 288
GetTextHeight function 344
GetTextWidth function 345
Getting Started 21
GetTransparentColor macro 382
GetX macro 360
GetY macro 360
GFX_COLOR type 337
GFX_EXTDATA structure 370
GFX_free macro 43
GFX_GRADIENT_STYLE structure 371
GFX_GRADIENT_TYPE enumeration 371

GFX_IMAGE_HEADER structure 368
GFX_LCD_CSTN macro 39
GFX_LCD_MSTN macro 39
GFX_LCD_OFF macro 39
GFX_LCD_TFT macro 39
GFX_LCD_TYPE macro 38
GFX_malloc macro 43
GFX_PAGE enumeration 442
GFX_PAGE_INFO structure 443
GFX_RESOURCE enumeration 368
GFX_WINDOW_INFO structure 443
GFXExecutePendingTransition function 405
GFXGetPageOriginAddress function 402
GFXGetPageXYAddress function 402
GFXGetWindowOriginAddress function 403
GFXGetWindowXYAddress function 403
GFXIsTransitionPending function 405
GFXSetupTransition function 405
GFXTransition function 406
GFXTranslateDirection function 406
GifDecoder.c 464
GifDecoder.h 475
GOL Global Variables 333
GOL Layer 530
GOL Messages 311
GOL Objects 48
GOL.c 531
GOL.h 552
GOL_EMBOSS_SIZE macro 332
GOL_MSG structure 316
GOL_OBJ_TYPE enumeration 51
GOL_SCHEME structure 331
GOLAddObject function 292
GOLCanBeFocused function 307
GOLCreateScheme function 328
GOLDeleteObject function 301
GOLDeleteObjectByID function 301
GOLDraw function 296
GOLDrawCallback function 297
GOLDrawComplete macro 296
GOLFindObject function 293
GOLFontDefault variable 332
GOLFontDefault.c 579
GOLFree function 298
GOLGetFocus macro 306
GOLGetFocusNext function 307
GOLGetFocusPrev function 308
GOLGetList macro 303
GOLGetScheme macro 330
GOLGetSchemeDefault macro 330
GOLInit function 306
GOLMsg function 314
GOLMsgCallback function 314
GOLNewList macro 302
GOLPanelDraw macro 308
GOLPanelDrawTsk function 310
GOLRedraw macro 294
GOLRedrawRec function 295
GOLSchemeDefault variable 332
GOLSetFocus function 304
GOLSetList macro 303
GOLSetScheme macro 329
GOLTwoTonePanelDrawTsk function 310
Gradients 346
Graphic Cursor 359
Graphics Hardware Platform 41
Graphics Library Configuration 30
Graphics Object Layer 48
Graphics Object Layer Configuration 30
Graphics Object Selection 31
Graphics Primitive Layer 338
Graphics Primitive Layer Configuration 36
Graphics.h 620
GraphicsConfig.h 928
GraphicsConfig.h Example 44
GRAY0 macro 391
GRAY1 macro 392
GRAY2 macro 392
GRAY4 macro 392
GRAY6 macro 392
GREEN macro 392
Grid 153
Grid Item States 157
Grid States 154

GRID structure 169
Grid.c 758
Grid.h 766
GRID_DISABLED macro 155
GRID_DRAW_ALL macro 156
GRID_DRAW_ITEMS macro 156
GRID_FOCUSED macro 155
GRID_HIDE macro 156
GRID_OUT_OF_BOUNDS macro 162
GRID_SHOW_BORDER_ONLY macro 156
GRID_SHOW_FOCUS macro 155
GRID_SHOW_LINES macro 155
GRID_SHOW_SEPARATORS_ONLY macro 156
GRID_SUCCESS macro 163
GridClearCellState function 161
GridCreate function 159
GridDraw function 160
GridFreeItems function 163
GridGetCell function 163
GridGetFocusX macro 161
GridGetFocusY macro 162
GRIDITEM structure 169
GRIDITEM_DRAW macro 159
GRIDITEM_IS_BITMAP macro 158
GRIDITEM_IS_TEXT macro 157
GRIDITEM_SELECTED macro 157
GRIDITEM_TEXTBOTTOM macro 158
GRIDITEM_TEXTLEFT macro 158
GRIDITEM_TEXTRIGHT macro 158
GRIDITEM_TEXTTOP macro 158
GridMsgDefault function 166
GridSetCell function 164
GridSetCellState function 165
GridSetFocus function 166
GridTranslateMsg function 167
Group Box 170
Group Box States 171
GROUPBOX structure 176
GroupBox.c 773
GroupBox.h 778

H

Hardware Profile 44
HardwareProfile.h Example 45
Height variable 447
HIDE macro 287
HIDE_DATA macro 84
HIT1270.c 1060
HIT1270.h 1075
HX8347.c 1077
HX8347.h 1095

I

Image Decoder Demo 457
Image Decoders 449
Image Decoders Files 457
Image Source Selection 42
IMAGE_EXTERNAL type 372
IMAGE_FLASH structure 369
IMAGE_NORMAL macro 365
IMAGE_RAM structure 370
IMAGE_X2 macro 365
ImageAbort macro 456
ImageDecode function 452
ImageDecoder.c 476
ImageDecoder.h 480
ImageDecoderInit function 453
ImageDecodeTask function 454
ImageFullScreenDecode macro 455
ImageLoopCallbackRegister function 454
InitGraph function 367
Input Device Selection 30
INPUT_DEVICE_EVENT enumeration 319
INPUT_DEVICE_TYPE enumeration 319
Introduction 1
InvalidateRectangle function 425
IsDeviceBusy function 395
IsObjUpdated macro 305
IsPaletteEnabled function 430

J

jidctint.c 507
JpegDecoder.c 487
JpegDecoder.h 505

K

Key Command Types 264
KEYMEMBER structure 277

L

LB_CENTER_ALIGN macro 179
LB_DISABLED macro 180
LB_DRAW macro 180
LB_DRAW_FOCUS macro 180
LB_DRAW_ITEMS macro 180
LB_FOCUSED macro 180
LB_HIDE macro 181
LB_RIGHT_ALIGN macro 179
LB_SINGLE_SEL macro 179
LB_STS_REDRAW macro 181
LB_STS_SELECTED macro 181
LbAddItem function 184
LbChangeSel function 186
LbClrtSel macro 191
LbCreate function 181
LbDelItem function 186
LbDelItemsList function 193
LbDraw function 183
LbGetBitmap macro 192
LbGetCount macro 189
LbGetFocusedItem function 188
LbGetItemList macro 184
LbGetSel function 188
LbGetVisibleCount macro 190
LbMsgDefault function 193
LbSetBitmap macro 191
LbSetFocusedItem function 189
LbSetSel macro 187
LbTranslateMsg function 194
Library Structure 47
LIGHTBLUE macro 393
LIGHTCYAN macro 393
LIGHTGRAY macro 393
LIGHTGREEN macro 393
LIGHTMAGENTA macro 394
LIGHTRED macro 394
Line function 347
Line Functions 347
Line Size 351
Line Types 350
LineRel macro 348
LineTo macro 349
List Box 177
List Box States 179
List Item Status 181
LISTBOX structure 195
ListBox.c 782
ListBox.h 794
LISTITEM structure 196

M

MAGENTA macro 394
Memory Type 366
Meter 196
Meter States 198
METER structure 209
Meter.c 804
Meter.h 817
METER_DISPLAY_VALUES_ENABLE macro 207
METER_TYPE macro 206
Microchip Application Library Abbreviations 28
Microchip Graphics Controller 407
Microchip Software Bundle License.txt 2
Miscellaneous 43
Miscellaneous Topics 516
MoveRel macro 361
MoveTo macro 361
MSG_DEFAULT_FUNC type 53
MSG_FUNC type 53
MTR_ACCURACY macro 207
MTR_DISABLED macro 198
MTR_DRAW macro 199

MTR_DRAW_UPDATE macro 199
MTR_HIDE macro 199
MTR_RING macro 199
MtrCreate function 200
MtrDecVal macro 203
MtrDraw function 201
MtrGetVal macro 203
MtrIncVal macro 204
MtrMsgDefault function 208
MtrSetScaleColors macro 204
MtrSetTitleFont macro 205
MtrSetVal function 202
MtrSetValueFont macro 206
MtrTranslateMsg function 208

N

NORMAL_LINE macro 351

O

OBJ_HEADER structure 52
Object Management 290
Object Rendering 335
Object States 285
OutChar function 342
OutText function 343
OutTextXY function 343

P

Palette Mode 427
Palette.h 435
PALETTE_ENTRY union 433
PALETTE_EXTERNAL type 434
PALETTE_FLASH structure 433
PALETTE_HEADER structure 434
PaletteInit function 430
PB_DISABLED macro 219
PB_DRAW macro 219
PB_DRAW_BAR macro 220
PB_HIDE macro 220
PB_VERTICAL macro 220
PbCreate function 220
PbDraw function 221
PbGetPos macro 224
PbGetRange macro 223
PbSetPos function 223
PbSetRange function 222
PbTranslateMsg function 225
PICT_DISABLED macro 211
PICT_DRAW macro 212
PICT_FRAME macro 212
PICT_HIDE macro 212
PictCreate function 212
PictDraw function 213
PictGetBitmap macro 215
PictGetScale macro 215
PictSetBitmap macro 214
PictSetScale macro 216
PictTranslateMsg function 216
Picture Control 210
Picture States 211
PICTURE structure 217
Picture.c 826
Picture.h 830
pixelWindow variable 447
Primitive Layer 941
Primitive.c 941
Primitive.h 1008
Progress Bar 218
Progress Bar States 219
PROGRESSBAR structure 226
ProgressBar.c 834
ProgressBar.h 840
PutImage function 362
PutPixel function 375

R

Radio Button 226
Radio Button States 228
RADIOBUTTON structure 237
RadioButton.c 858
RadioButton.h 866
RB_CHECKED macro 228
RB_DISABLED macro 228
RB_DRAW macro 228

RB_DRAW_CHECK macro 229
RB_DRAW_FOCUS macro 229
RB_FOCUSED macro 229
RB_GROUP macro 229
RB_HIDE macro 229
RbCreate function 230
RbDraw function 231
RbGetCheck function 232
RbGetText macro 233
RbMsgDefault function 235
RbSetCheck function 233
RbSetText function 234
RbTranslateMsg function 236
RCC_COPY macro 421
RCC_DEST_ADDR_CONTINUOUS macro 420
RCC_DEST_ADDR_DISCONTINUOUS macro 420
RCC_ROP_0 macro 421
RCC_ROP_1 macro 421
RCC_ROP_2 macro 421
RCC_ROP_3 macro 421
RCC_ROP_4 macro 421
RCC_ROP_5 macro 421
RCC_ROP_6 macro 421
RCC_ROP_7 macro 421
RCC_ROP_8 macro 421
RCC_ROP_9 macro 421
RCC_ROP_A macro 421
RCC_ROP_B macro 421
RCC_ROP_C macro 421
RCC_ROP_D macro 421
RCC_ROP_E macro 421
RCC_ROP_F macro 421
RCC_SOLID_FILL macro 421
RCC_SRC_ADDR_CONTINUOUS macro 420
RCC_SRC_ADDR_DISCONTINUOUS macro 420
RCC_TRANSPARENT_COPY macro 421
RDIA_DISABLED macro 126
RDIA_DRAW macro 126
RDIA_HIDE macro 126
RDIA_ROT_CCW macro 127
RDIA_ROT_CW macro 127
RdiaCreate function 127
RdiaDecVal macro 130
RdiaDraw function 128
RdiaGetVal macro 130
RdiaIncVal macro 129
RdiaMsgDefault function 132
RdiaSetVal macro 131
RdiaTranslateMsg function 132
Rectangle Copy Operations 407
Rectangle Functions 352
Rectangle macro 353
RED macro 394
References 1220
Release Notes 5
RequestDisplayUpdate function 425
RequestEntirePaletteChange macro 434
RequestPaletteChange function 430
ResetDevice function 396
RESOLUTION macro 207
RGBConvert macro 333
ROPBlock function 417
ROUNDDIAL structure 134
RoundDial.c 844
RoundDial.h 852

S

Scan Key Codes 320
SCAN_BS_PRESSED macro 321
SCAN_BS_RELEASED macro 321
SCAN_CR_PRESSED macro 321
SCAN_CR_RELEASED macro 321
SCAN_DEL_PRESSED macro 322
SCAN_DEL_RELEASED macro 322
SCAN_DOWN_PRESSED macro 322
SCAN_DOWN_RELEASED macro 322
SCAN_END_PRESSED macro 323
SCAN_END_RELEASED macro 323
SCAN_HOME_PRESSED macro 323
SCAN_HOME_RELEASED macro 323
SCAN_LEFT_PRESSED macro 323
SCAN_LEFT_RELEASED macro 324
SCAN_PGDOWN_PRESSED macro 324
SCAN_PGDOWN_RELEASED macro 324

SCAN_PGUP_PRESSED macro 324
SCAN_PGUP_RELEASED macro 325
SCAN_RIGHT_PRESSED macro 325
SCAN_RIGHT_RELEASED macro 325
SCAN_SPACE_PRESSED macro 325
SCAN_SPACE_RELEASED macro 325
SCAN_TAB_PRESSED macro 326
SCAN_TAB_RELEASED macro 326
SCAN_UP_PRESSED macro 326
SCAN_UP_RELEASED macro 326
ScanCodes.h 622
Scroll function 419
Set Up Functions 366
SetActivePage function 387
SetBevelDrawType macro 359
SetClip function 378
SetClipRgn function 378
SetColor macro 376
SetEntirePalette macro 435
SetFont function 340
SetFontOrientation macro 341
SetLineThickness macro 349
SetLineType macro 350
SetPalette function 431
SetPalette macro 396
SetPaletteBpp function 432
SetPaletteFlash function 432
SetState macro 289
SetVisualPage function 388
SH1101A_SSD1303.c 1096
SH1101A_SSD1303.h 1104
SHOW_DATA macro 84
SLD_DISABLED macro 239
SLD_DRAW macro 240
SLD_DRAW_FOCUS macro 240
SLD_DRAW_THUMB macro 240
SLD_FOCUSED macro 240
SLD_HIDE macro 240
SLD_SCROLLBAR macro 241
SLD_VERTICAL macro 241
SldCreate function 241
SldDecPos macro 249
SldDraw function 243
SldGetPage macro 244
SldGetPos macro 246
SldGetRange macro 247
SldIncPos macro 248
SldMsgDefault function 250
SldSetPage function 244
SldSetPos function 245
SldSetRange function 247
SldTranslateMsg function 251
Slider States 239
SLIDER structure 252
Slider.c 872
Slider.h 886
Slider/Scroll Bar 237
SOLID_LINE macro 351
SSD1339.c 1106
SSD1339.h 1122
SSD1926.c 1124
SSD1926.h 1168
ST_CENTER_ALIGN macro 254
ST_DISABLED macro 254
ST_DRAW macro 254
ST_FRAME macro 255
ST_HIDE macro 255
ST_RIGHT_ALIGN macro 255
ST_UPDATE macro 255
Starting a New Project 516
Startx variable 448
Starty variable 448
Static Text 252
Static Text States 254
STATICTEXT structure 259
StaticText.c 895
StaticText.h 900
StCreate function 255
StDraw function 256
StGetText macro 257
STN_DISPLAY_WIDTH macro 40
STN_DISPLAY_WIDTH_16 macro 40
STN_DISPLAY_WIDTH_4 macro 40
STN_DISPLAY_WIDTH_8 macro 41

StSetText function 258
StTranslateMsg function 258
Style Scheme 327
SwitchOffDoubleBuffering function 424
SwitchOnDoubleBuffering function 424

T

TCON_Custom.c 1218
TCON_HX8238.c 1187
TCON_HX8257.c 1198
TCON_SSD1289.c 1192
TE_DELETE_COM macro 264
TE_DISABLED macro 263
TE_DRAW macro 263
TE_ECHO_HIDE macro 263
TE_ENTER_COM macro 264
TE_HIDE macro 263
TE_KEY_PRESSED macro 262
TE_SPACE_COM macro 265
TE_UPDATE_KEY macro 263
TE_UPDATE_TEXT macro 264
TeAddChar function 270
TeClearBuffer function 268
TeCreate function 265
TeCreateKeyMembers function 270
TeDelKeyMembers function 272
TeDraw function 266
TeGetBuffer macro 267
TeGetKeyCommand function 268
TeIsKeyPressed function 271
TeMsgDefault function 273
TeSetBuffer function 267
TeSetKeyCommand function 269
TeSetKeyText function 273
TeSpaceChar function 272
TeTranslateMsg function 275
Text Entry 260
Text Functions 338
TextEntry States 262
TEXTENTRY structure 276
TextEntry.c 904
TextEntry.h 919
THICK_LINE macro 352
TRANS_MSG enumeration 317
Transitions 403
TRANSPARENT_COLOR_DISABLE macro 383
TRANSPARENT_COLOR_ENABLE macro 383
TransparentColorDisable macro 382
TransparentColorEnable function 381

U

UC1610.c 1204
UC1610.h 1216
UPDATE_HOUR macro 56
UPDATE_MINUTE macro 56
UPDATE_SECOND macro 56
UpdateDisplayNow function 426
USE_ANALOGCLOCK macro 36
USE_BITMAP_FLASH macro 43
USE_BUTTON macro 32
USE_BUTTON_MULTI_LINE macro 32
USE_CHECKBOX macro 32
USE_CUSTOM macro 35
USE_DIGITALMETER macro 33
USE_EDITBOX macro 33
USE_FOCUS macro 31
USE_FONT_FLASH macro 42
USE_GOL macro 35
USE_GROUPBOX macro 33
USE_KEYBOARD macro 30
USE_LISTBOX macro 33
USE_METER macro 33
USE_MULTIBYTECHAR macro 37
USE_NONBLOCKING_CONFIG macro 42
USE_PICTURE macro 34
USE_PROGRESSBAR macro 34
USE_RADIOBUTTON macro 34
USE_ROUNDDIAL macro 34
USE_SLIDER macro 35
USE_STATICTEXT macro 35
USE_TEXTENTRY macro 36
USE_TOUCHSCREEN macro 30
USE_WINDOW macro 35
Using Microchip Graphics Module Color Look Up Table in Applications 524

Using Primitive Rendering Functions in Blocking and Non-Blocking Modes 523

W

WHITE macro 394

Width variable 448

Window 278

Window States 279

WINDOW structure 285

Window.c 932

Window.h 937

WND_DISABLED macro 279

WND_DRAW macro 280

WND_DRAW_CLIENT macro 280

WND_DRAW_TITLE macro 280

WND_FOCUSED macro 280

WND_HIDE macro 280

WND_TITLECENTER macro 281

WndCreate function 281

WndDraw function 282

WndGetText macro 283

WndSetText function 283

WndTranslateMsg function 284

X

XCHAR macro 345

Y

YELLOW macro 395