emWin

Graphic Library with Graphical User Interface

User & Reference Guide

Document: UM03001
Software version: V5.30
Document revision: 0
Date: July 6, 2015

A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

Disclaimer

Copyright notice

Trademarks

Contact address

SEGGER Microcontroller GmbH & Co. KG
In den Weiden 11
D-40721 Hilden
Germany
Tel.+49 2103-2878-0
Fax.+49 2103-2878-28
E-mail: support@segger.com
Internet: http://www.segger.com

Manual versions

This manual describes the current software version. If any error occurs, inform us and we will try to assist you as soon as possible.
Contact us for further information on topics or routines not yet specified.
Print date: July 6, 2015

Software	Rev.	Date	By	Description
5.30	0	150706	JE AS	Chapter 8 ’2-D Graphic Library’ - Function GUI_PreserveTrans() added. - Function GUI_SetAlphaMask8888() added. Chapter 10 ’Bitmap Converter’ - Saving of PNGs now supported. - New bitmap format M8888I supported. Chapter 12 ’Font Converter’ - Loading of Adobe (B)itmap (D)istribution (F)ormat (BDF) added. Chapter 14 ’Animations’ - Function GUI_ANIM_AddItem() added. - Function GUI_ANIM_Create() added. - Function GUI_ANIM_Delete() added. - Function GUI_ANIM_Exec() added. - Function GUI_ANIM_Start() added. Chapter 15 ’Colors’ - New logical color mode ARGB added. Chapter 16 ’Memory Devices’ - Function GUI_MEMDEV_WriteOpaque() added. - Function GUI_MEMDEV_WriteOpaqueAt() added. Chapter 18 ’Window Manager’ - Explanation of tiling algorithm added. Chapter 19 ’Widgets’ - Function LISTWHEEL_IsMoving() added. - Function PROGBAR_GetMinMax() added. - Function PROGBAR_GetValue() added. - Function TEXT_GetBkColor() added. - Function TEXT_GetFont() added. - Function TEXT_GetTextAlign() added. - Function TEXT_GetTextColor() added. - Function TEXT_GetWrapMode() added. - Function TEXT_GetDefaultTextColor() added. - Function TEXT_GetDefaultWrapMode() added. Chapter 33 ’Display Drivers’ - Support for HX8369 added to GUIDRV_FlexColor. - Support for ST7715 added to GUIDRV_FlexColor. - Support for SSD1325 added to GUIDRV_SLin.
5.28	1	150206	JE	Licensing information of TTF and PNG support changed.
5.28	0	150128	JE AS	Chapter 3 ’Simulation’ - Function SIM_GUI_SetTransMode() added. New Chapter 5 ’emWinSPY’ - New tool emWinSPY added: - Function GUI_SPY_Process() added. - Function GUI_SPY_SetMemHandler() added. - Function GUI_SPY_StartServer() added. - Function GUI_SPY_X_StartServer() added. Chapter 8 ’2-D Graphic Library’ - Function GUI_SetRefreshHook() added. Chapter 11 ’Fonts’ - Commas added to standard digit fonts. Chapter 18 ’Widgets’ - Some new default values added to FRAMEWIN and WINDOW. - Function LISTBOX_EnableWrapMode() added.

Table 0.1: Revision history

Software	Rev.	Date	By	Description
5.28	0	150128	JE AS	Chapter 24 ’Multi layer / multi display support’ - Softlayers added: - Function GUI_SOFTLAYER_Enable() added. - Function GUI_SOFTLAYER_MULTIBUF_Enable() added. - Function GUI_SOFTLAYER_Refresh() added. - Function GUI_SOFTLAYER_SetCompositeColor() added. Chapter 28 ’Sprites’ - Sprites now support true color bitmaps with alpha blending. Chapter 32 ’Display drivers’ - LCD_DEVFUNC_DRAWBMP_8BPP added to LCD_SetDevFunc(). - Support for ST7789 added to GUIDRV_FlexColor. - Support for UC1638 added to GUIDRV_SPage - Support for Avant Electronics SBN0064G added to GUIDRV_SPage - New display driver GUIDRV_7528 added Chapter 37 ’Configuration’ - Function GUITASK_GetMaxTask() added. - Function GUICC_M1555I_SetCustColorConv() added. - Function GUICC_M565_SetCustColorConv() added. - Function GUICC_M4444I_SetCustColorConv() added. - Function GUICC_M888_SetCustColorConv() added. - Function GUICC_M8888I_SetCustColorConv() added. - Function GUI_SetFuncAlphaBlending() added. - Function GUI_SetFuncGetpPalConvTable() added. - Function GUI_SetFuncMixColors() added. - Function GUI_SetFuncMixColorsBulk() added. - Function GUI_AA_SetpfDrawCharAA4() added. - Function GUI_MEMDEV_SetDrawMemdev16bppFunc() added.
5.26	2	141128	AS	Various corrections.
5.26	1	140821	JE AS	Chapter 17 ’Widgets’ - Function MULTIPAGE_SetDefaultBorderSizeX() added. - Function MULTIPAGE_SetDefaultBorderSizeY() added. Various corrections.
5.26	0	140805	JE AS	Chapter 7 ’2-D Graphic Library’ - Function GUI_DIRTYDEVICE_Create() added. - Function GUI_DIRTYDEVICE_CreateEx() added. - Function GUI_DIRTYDEVICE_Delete() added. - Function GUI_DIRTYDEVICE_DeleteEx() added. - Function GUI_DIRTYDEVICE_Fetch() added. - Function GUI_DIRTYDEVICE_FetchEx() added. Chapter 9 ’Bitmap Converter’ - Dithering added. Chapter 14 ’Memory Devices’ - Function GUI_MEMDEV_FadeOutDevices() added. - Function GUI_MEMDEV_RotateHQAlpha() added. - Function GUI_MEMDEV_RotateAlpha() added. - Function GUI_MEMDEV_Dither32() added. Chapter 16 ’The Window Manager (WM)’ - Function WM_GetScrollbarH() added. - Function WM_GetScrollbarV() added. - Function WM_SetModalLayer() added. - Function WM_GetModalLayer() added. Chapter 17 ’Widgets’ - Function LISTVIEW_EnableCellSelect() added. - Function LISTVIEW_GetItemRect() added. - Function LISTVIEW_SetItemTextSorted() added. - Function MULTIPAGE_EnableScrollBar() added. - Function MULTIPAGE_SetBitmap() added. - Function MULTIPAGE_SetBitmapEx() added. - Function MULTIPAGE_SetTabHeight() added. - Function MULTIPAGE_SetTabWidth() added. - Function MULTIPAGE_SetTextAlign() added. Chapter 29 ’Antialiasing’ - Function GUI_AA_FillEllipse() added. Chapter 31 ’Display drivers’ - Support for LGDP4525 added to GUIDRV_FlexColor. - Support for Ilitek ILI9488 added to GUIDRV_FlexColor. - Support for Himax HX8357 added to GUIDRV_FlexColor. - Support for Raio RA8875 added to GUIDRV_FlexColor. - Support for OriseTech SPLC502B added to GUIDRV_SPage.

Table 0.2: Revision history

Software	Rev.	Date	By	Description
5.26	0	140805	JE AS	Chapter 16 'The Window Manager (WM)' - Function WM_GetScrollbarH() added. - Function WM_GetScrollbarV() added. - Function WM_SetModalLayer() added. - Function WM_GetModalLayer() added. Chapter 17 'Widgets' - Function LISTVIEW_EnableCellSelect() added. - Function LISTVIEW_GetItemRect() added. - Function LISTVIEW_SetItemTextSorted() added. - Function MULTIPAGE_EnableScrollBar() added. - Function MULTIPAGE_SetBitmap() added. - Function MULTIPAGE_SetBitmapEx() added. - Function MULTIPAGE_SetTabHeight() added. - Function MULTIPAGE_SetTabWidth() added. - Function MULTIPAGE_SetTextAlign() added. Chapter 29 'Antialiasing' - Function GUI_AA_FillEllipse() added. Chapter 31 'Display drivers' - Support for LGDP4525 added to GUIDRV_FlexColor. - Support for Ilitek ILI9488 added to GUIDRV_FlexColor. - Support for Himax HX8357 added to GUIDRV_FlexColor. - Support for Raio RA8875 added to GUIDRV_FlexColor. - Support for OriseTech SPLC502B added to GUIDRV_SPage.
5.24	2	140429	Sc AS	Table titles were added for all tables except "Permitted values". Chapter 13 'Colors' - Structure 'LCD_PHYSPALETTE' added. - Section 'Look-up table API' added. - Function 'LCD_SetLUT()' added. - Function 'LCD_SetLUTEx()' added. - Function 'LCD_SetLUTEntryEx()' added.
5.24	1	140225	Sc AS	Various corrections.
5.24	0	130801	JE	Chapter 7 '2-D Graphic Library' - New function GUI_CreateBitmapFromStreamA555() added. - New function GUI_CreateBitmapFromStreamAM555() added. - New function GUI_CreateBitmapFromStreamA565() added. - New function GUI_CreateBitmapFromStreamAM565() added. - New function GUI_DrawStreamedBitmapA555Ex() added. - New function GUI_DrawStreamedBitmapAM555Ex() added. - New function GUI_DrawStreamedBitmapA565Ex() added. - New function GUI_DrawStreamedBitmapAM565Ex() added. Chapter 9 'Bitmap Converter' - New bitmap formats added: 16bpp + 8 bit alpha channel. Chapter 10 'Fonts' - New function GUI_TTF_CreateFontAA() added. Chapter 13 'Memory Devices' - New function GUI_MEMDEV_ClearAlpha() added. - New function GUI_MEMDEV_CreateFixed32() added. - New function GUI_MEMDEV_BlendColor32() added. Chapter 16 'Window Manager' - Circular motion support added. Chapter 17 'Widgets' - New widget "KNOB" added. - New function LISTWHEEL_SetDeceleration() added. - New function DROPDOWN_SetListHeight() added. - New function LISTVIEW_OwnerDraw() added. - New function LISTVIEW_SetOwnerDraw() added. - New function LISTVIEW_GetWrapMode() added. - New function MULTIPAGE_GetPageText() added. - New function SPINBOX_SetEditMode() added. - New function SPINBOX_SetStep() added. New chapter 25 'MultiTouch' added. - New function GUI_MTOUCH_Enable() added. - New function GUI_MTOUCH_GetEvent() added. - New function GUI_MTOUCH_GetTouchInput() added. - New function GUI_MTOUCH_IsEmpty() added. - New function GUI_MTOUCH_SetOrientation() added. - New function GUI_MTOUCH_SetOrientationEx() added. - New function GUI_MTOUCH_StoreEvent() added. - New function WM_EnableGestures() added.

Table 0.3: Revision history

Software	Rev.	Date	By	Description
5.24	0	130801	JE	Chapter 29 ’Antialiasing’ - New function GUI_AA_PreserveTrans() added. Chapter 30 ’Language Support’ - Remark that Devanagari transitions are not supported. Chapter 31 ’Display Drivers’ - New function LCD_SetChromaEx() added. - New function LCD_SetChromaModeEx() added. - New function LCD_SetAlpha() added. - New function LCD_SetAlphaModeEx() added. - New function LCD_SetVisEx() added. - Support for Ilitek ILI9163 added to GUIDRV_FlexColor. - Support for RAIO8870 added to GUIDRV_FlexColor. - Support for Solomon SSD1351 added to GUIDRV_FlexColor. - Support for RAIO 8835 added to GUIDRV_SLin. - Support for Samsung S6B0108 added to GUIDRV_SPage. - Support for Hitachi HD61202 added to GUIDRV_SPage. Chapter 32 ’Touch drivers’ - New multi touch driver added for PIXCIR TangoC32. Chapter 35 ’Performance and Resource Usage’ - New sub chapter ’Optimizing Footprint’ added.
5.22	2	140108	AS	GUIDRV_CompactColor_16: - Support for Samsung S6D0128 added to 66772. - Support for Sitronix ST7789 added to 66717.
5.22	1	130801	AS	Various corrections.
5.22	0	130625	JE AS	New chapter 11 ’Movies’ added. Chapter 12 ’Colors’ - New color conversion GUICC_8 added. Chapter 13 ’Memory Devices’ - New function GUI_MEMDEV_BlendWinBk() added. - New function GUI_MEMDEV_BlurAndBlendWinBk() added. - New function GUI_MEMDEV_BlurWinBk() added. Chapter 13 ’Memory Devices’ - New function GUI_MEMDEV_CreateBlurredDevice32() added. - New function GUI_MEMDEV_CreateBlurredDevice32HQ() added. - New function GUI_MEMDEV_CreateBlurredDevice32LQ() added. - New function GUI_MEMDEV_PunchOutDevice() added. - New function GUI_MEMDEV_RotateHQHR() added. - New function GUI_MEMDEV_RotateHR() added. - New function GUI_MEMDEV_SetBlurHQ() added. - New function GUI_MEMDEV_SetBlurLQ() added. Chapter 16 ’ Widgets’ - Added notification messages sent by the IMAGE widget. - Added notification messages sent by the TEXT widget. - New function GRAPH_SetAutoScrollbar() added. - New function GRAPH_GetScrollValue() added. - New function GRAPH_SetScrollValue() added. - New function ICONVIEW_SetIconAlign() added. - New function LISTWHEEL_GetItemFromPos() added. - New function MULTIEDIT_SetFocussable() added. - New function MULTIPAGE_GetPageText() added. - New function TREEVIEW_ScrollToSel() added. - New section 16.1.2 ’Custom widgets’ added. Chapter 25 ’Sprites’ - New function GUI_SPRITE_SetLoop() added. - New function GUI_SPRITE_StartAnim() added. - New function GUI_SPRITE_StopAnim() added. Chapter 27 ’Antialiasing’ - New function GUI_AA_FillRoundedRect() added. - New function GUI_AA_FillRoundedRectEx() added. - New function GUI_AA_DrawRoundedRect() added. - New function GUI_AA_DrawRoundedRectEx() added. Chapter 29 ’Display drivers’ - New display driver GUIDRV_UC1698G added. - Support for Solomon SSD1306 added to GUIDRV_SPage.

Table 0.4: Revision history

Software	Rev.	Date	By	Description
5.22	0	130625	JE AS	Chapter 33 'Timing- and execution-related functions' - New function GUI_TIMER_Create() added. - New function GUI_TIMER_Delete() added. - New function GUI_TIMER_Restart() added. - New function GUI_TIMER_SetPeriod() added.
5.20	4	130409	AS	GUIDRV_SPage: - Support for Solomon SSD1305 added to 1510.
5.20	3	130409	AS	Various corrections / improvements.
5.20	2	130308	AS	Chapter 28 'Language Support' - New function GUI_LANG_GetTextBuffered() added. - New function GUI_LANG_GetTextBufferedEx() added.
5.20	1	130305	AS	Chapter 19 'Skinning' - New function CHECKBOX_GetSkinFlexButtonSize() added. - New function CHECKBOX_SetSkinFlexButtonSize() added.
5.20	0	130218	JE	Chapter 10 'Fonts' - Support for iType® fonts of Monotype Imaging added. Chapter 12 'Colors' - New color conversions added: GUICC_M8888I, GUICC_M1555I, GUICC_M4444I, GUICC_1616I, GUICC_88666I Chapter 17 'Dialogs' - CALENDAR dialog and functions added: CALENDAR_Create() CALENDAR_GetDate() CALENDAR_GetSel() CALENDAR_SetDate() CALENDAR_SetSel() CALENDAR_SetDefaultBkColor() CALENDAR_SetDefaultColor() CALENDAR_SetDefaultDays() CALENDAR_SetDefaultFont() CALENDAR_SetDefaultMonths() CALENDAR_SetDefaultSize() Chapter 25 'Sprites' - New function GUI_SPRITE_CreateHidden() added. - New function GUI_SPRITE_CreateHiddenEx() added. Chapter 29 'Display Drivers' GUIDRV_FlexColor: - Support for Himax HX8340 added to 66712. - New module 66772 added wit support for: Hitachi HD66772, Samsung S6D0117, Sitronix ST7712, Himax HX8301, Ilitek ILI9220 and ILI9221 GUIDRV_SLin: - Support for Epson S1D13305 added. Chapter 30 'VNC-Server' - New function GUI_VNC_SetLockFrame() added. Chapter 32 'Timing and execution' - New function GUI_Error() added. Chapter 34 'Configuration' - New function GUI_SetOnErrorFunc() added.
5.18	0	120917	JE AS	Chapter 8 'Displaying bitmap files' - New function GUI_BMP_SerializeExBpp() added. Chapter 9 'Bitmap Converter' - New functions added to create animated sprites and cursors out of animated GIF files. Chapter 13 'Memory Devices' - New function GUI_MEMDEV_SerializeBMP() added. Chapter 15 'The Window Manager (WM)' New function WM_SetCaptureMove() added. New function WM_Screen2hWin() added. New function WM_Screen2hWinEx() added. Chapter 16 'Window Objects (Widgets)' - New functions added: TEXT_GetText() LISTVIEW_SetWrapMode() Chapter 27 'Antialiasing' - New function GUI_AA_SetDrawMode() added.

Table 0.5: Revision history

Software	Rev.	Date	By	Description
5.18	0	120917	JE AS	Chapter 28 'Foreign Language Support' - New feature "Text- and language resource files" added. Chapter 29 'Display drivers' GUIDRV_FlexColor: - Function GUIDRV_FlexColor_SetInterface66709_B16() replaced by the function GUIDRV_FlexColor_SetReadFunc66709_B16(). - Function GUIDRV_FlexColor_SetInterface66720_B16() replaced by the function GUIDRV_FlexColor_SetReadFunc66720_B16(). - New module 66702 added: Solomon SSD1284, SSD1289, SSD1298 - New module 66715 added: Himax HX8352B - Recommended calling sequence for configuration functions added. GUIDRV_S1D13781: - Additional information about initialized registers added.
5.16	2	120809	AS	Chapter 16 'Window Objects (Widgets)' - New function SPINBOX_SetRange() added. Various corrections.
5.16	1	120628	AS	Chapter 15 'The Window Manager (WM)' Descriptions of the following functions reworked: - WM_GetScrollPosH() - WM_GetScrollPosV() - WM_SetScrollPosH() - WM_SetScrollPosV() Preface, About and Chapter 1 'Intro' reworked.
5.16	0	120605	JE AS	Chapter 12 'Colors' - New color conversion routine added to support 1bpp at different color depths. Chapter 13 'Memory Devices' - New function GUI_MEMDEV_RotateHQT() added. Chapter 15 'The Window Manager (WM)' - Support for ToolTips added. - New functions added: WM_TOOLTIP_AddTool() WM_TOOLTIP_Create() WM_TOOLTIP_Delete() WM_TOOLTIP_SetDefaultFont() WM_TOOLTIP_SetDefaultColor() WM_TOOLTIP_SetDefaultPeriod() Chapter 16 'Window Objects (Widgets)' - New functions added: BUTTON_SetReactOnTouch() DROPDOWN_SetUpMode() ICONVIEW_EnableStreamAuto() - Changed function SPINBOX_SetButtonSize(): New option SPINBOX_EDGE_CENTER. Chapter 17 'Dialogs' - CHOOSECOLOR dialog and functions added: CHOOSECOLOR_Create() CHOOSECOLOR_GetSel() CHOOSECOLOR_SetSel() CHOOSECOLOR_SetDefaultColor() CHOOSECOLOR_SetDefaultSpace() CHOOSECOLOR_SetDefaultBorder() CHOOSECOLOR_SetDefaultButtonSize() - CHOOSEFILE dialog and functions added: CHOOSEFILE_Create() CHOOSEFILE_EnableToolTips() CHOOSEFILE_SetButtonText() CHOOSEFILE_SetDefaultButtonText() CHOOSEFILE_SetToolTips() CHOOSEFILE_SetTopMode() Chapter 23 'Pointer Input Devices' - New function GUI_PID_IsPressed() added. Chapter 24 'Keyboard Input' - New function GUI_GetKeyState() added. Chapter 28 'Foreign Language Support' - New function GUI_LANG_GetNumItems() added.

Table 0.6: Revision history

Software	Rev.	Date	By	Description
5.16	0	120605	JE AS	Chapter 28 'Foreign Language Support' - New function GUI_LANG_GetText() added. - New function GUI_LANG_GetTextEx() added. - New function GUI_LANG_LoadCSV() added. - New function GUI_LANG_LoadCSVEx() added. - New function GUI_LANG_LoadText() added. - New function GUI_LANG_LoadTextEx() added. - New function GUI_LANG_SetLang() added. - New function GUI_LANG_SetMaxNumLang() added. - New function GUI_LANG_SetSep() added. Chapter 29 'Display drivers' - New display controller supported by GUIDRV_SPage: GUIDRV_SPage_Set1510: Epson S1D15605, S1D15606, S1D15607, S1D15608, S1D15705, S1D15710, S1D15714 Integrated Solutions Technology IST3020 New Japan Radio Company NJU6676 Novatek NT7502, NT7534, NT7538, NT75451 Samsung S6B0719, S6B0713, S6B0724, S6B1713 Sino Wealth SH1101A Sitronix ST7522, ST7565, ST7567 Solomon SSD1303, SSD1805, SSD1815, SSD1821 Sunplus SPLC501C UltraChip UC1608, UC1701, UC1601, UC1606 GUIDRV_SPage_Set1512: Epson S1D15E05, S1D15E06, S1D15719, S1D15721 GUIDRV_SPage_SetST7591: Sitronix ST7591 - New display controllers supported by GUIDRV_FlexColor: 66708: Ilitek ILI9335 66709: Ilitek ILI9338, ILI9340, ILI9341, ILI9342 66719: Samsung S6E63D6 - New function LCD_SetMaxNumColors() added. - Support for 1bpp added to GUIDRV_SPage. - Function GUIDRV_SPage_SetS1D15() obsolete Replaced by GUIDRV_SPage_Set1512 - New variants GUIDRV_Lin added: GUIDRV_LIN_OX_8 GUIDRV_LIN_OXY_8 - New driver GUIDRV_S1D13781 added. New chapter 30 'Touch drivers' added. - New driver GUITDRV_ADS7846 added.
5.14	3	120202	AS	Chapter 29 'Display drivers' - New display controller supported by GUIDRV_FlexColor: 66709: Ilitek ILI9340 - New display controllers supported by GUIDRV_SPage: Epson S1D15605, S1D15606, S1D15607, S1D15608, S1D15705, S1D15710, S1D15714 Integrated Solutions Technology IST3020 New Japan Radio Company NJU6676 Novatek NT7502, NT7534, NT7538, NT75451 Samsung S6B0719, S6B0713, S6B0724, S6B1713 Sino Wealth SH1101A Sitronix ST7522, ST7565, ST7567 Solomon SSD1805, SSD1303, SSD1815 UltraChip UC1608, UC1701, UC1601, UC1606 Sunplus SPLC501C - New function GUIDRV_SPage_Set1510 added. - New function GUIDRV_SPage_Set1512 added.
5.14	2	120201	AS	Chapter 8 'Displaying bitmaps files' - Improved description of the 'GetData'-function. Chapter 29 'Display drivers' - GUIDRV_SPage now supports 1bpp. - New display controllers supported by GUIDRV_SPage: Epson S1D15E05, S1D15E06, S1D15719, S1D15721 Sitronix ST7591 - New function GUIDRV_SPage_SetST7591 added.

Table 0.7: Revision history

Software	Rev.	Date	By	Description
5.14	1	120124	AS	Chapter 16 ’Window Objects (Widgets)’ - New function EDIT_GetTextColor(). - New function SPINBOX_GetEditHandle().
5.14	0	120111	JE AS	Chapter 3 ’Simulation’ - New function SIM_GUI_Enable() added. Chapter 7 ’2D Graphic Library’ - New functions added: GUI_DrawStreamedBitmapAuto() GUI_DrawStreamedBitmapExAuto() GUI_DrawStreamedBitmap24Ex() GUI_DrawStreamedBitmap555Ex() GUI_DrawStreamedBitmap565Ex() GUI_DrawStreamedBitmapM555Ex() GUI_DrawStreamedBitmapM565Ex() Chapter 11 ’Font Converter’ - Functions to size, shift and move characters added. Chapter 12 ’Colors’ - Sub chapter ’Gamma correction’ added. Chapter 15 ’The Window Manager (WM)’ - Change of the WM_MOVE message to transmit position changes. - New functions added: WM_MOTION_Enable() WM_MOTION_SetMovement() WM_MOTION_SetDeceleration() WM_MOTION_SetDefaultPeriod() WM_MOTION_SetMotion() WM_MOTION_SetMoveable() WM_MOTION_SetSpeed() Chapter 16 ’Window Objects (Widgets)’ - New widget "SPINBOX" added. - New widget "IMAGE" added. - Return values added to the following functions: BUTTON_SetText() TEXT_SetText() - New function DROPDOWN_GetItemText() added. - New function EDIT_GetBkColor() added. - New function EDIT_SetBkColor() added. - New function EDIT_SetFocussable() added. - New function EDIT_GetFont() added. - New function GUI_EditFloat() added. - Listing of widget IDs added. Chapter 25 ’Sprites’ - New function GUI_SPRITE_CreateAnim() added. - New function GUI_SPRITE_CreateExAnim() added. Chapter 26 ’Cursors’ - New function GUI_CURSOR_SelectAnim() added. Chapter 29 ’Display drivers’ - New display controllers supported by GUIDRV_FlexColor: 66708: FocalTech FT1509 66709: Renesas R61526 66709: Ilitek ILI9342 66712: Himax HX8347 66712: Himax HX8352 - New display controllers supported by GUIDRV_CompactColor_16: 66708: FocalTech FT1509 66709: Renesas R61526 66709: Ilitek ILI9342
5.12	1	111021	AS	Font Converter documentation added as chapter 11. - New function GUIDRV_FlexColor_SetFunc66712() added. - New function GUIDRV_FlexColor_SetInterface66712B16() added. - New display controller supported by GUIDRV_07X1: 741: Novatek NT7508 - New display controller supported by GUIDRV_Page1bpp: 1510: Solomon SSD1821 - GUIDRV_Lin ’Using the Lin driver in systems with cache memory’ changed.

Table 0.8: Revision history

Software	Rev.	Date	By	Description
5.12	0	110621	AS JE	Chapter 16 ’Window Objects (Widgets)’ - New function LISTVIEW_SetHeaderHeight() added. - New function ICONVIEW_AddStreamedBitmapItem() added. - New function ICONVIEW_GetItemText() added. - New function ICONVIEW_GetItemUserData() added. - New function ICONVIEW_GetNumItems() added. - New function ICONVIEW_InsertBitmapItem() added. - New function ICONVIEW_InsertStreamedBitmapItem() added. - New function ICONVIEW_SetBitmapItem() added. - New function ICONVIEW_SetFrame() added. - New function ICONVIEW_SetItemText() added. - New function ICONVIEW_SetItemUserData() added. - New function ICONVIEW_SetSpace() added. - New function ICONVIEW_SetStreamedBitmapItem() added. - New function ICONVIEW_SetTextAlign() added. - New function TEXT_GetNumLines() added. Chapter 29 ’Display Drivers’ - New display drivers added: GUIDRV_Dist GUIDRV_SPage - New display controller supported by GUIDRV_CompactColor_16: 66709: Solomon SSD1961 - LCD_SetDevFunc(): LCD_DEVFUNC_COPYRECT added. - GUIDRV_Lin: Support for LCD_DEVFUNC_COPYRECT added.
5.10	1	110531	AS JE	Chapter 29 ’Display Drivers - New display driver: GUIDRV_FlexColor
5.10	0	110329	AS JE	Chapter 13 ’Memory Devices’ - Default for GUI_USE_MEMDEV_1BPP_FOR_SCREEN set to 1. - New function GUI_MEMDEV_MarkDirty() added. New chapter 18 ’GUIBuilder’ added. Chapter 29 ’Display Drivers’ - New display controllers supported by GUIDRV_CompactColor_16: 66708: Ilitek ILI9328 66709: Sitronix ST7715 66772: Ilitek ILI9221 - New function GUIDRV_BitPlains_Config() added.
5.08	0	110112	AS JE	Chapter 9 ’2D Graphic Library’ - New function GUI_CreateBitmapFromStreamRLEAlpha() added. - New function GUI_CreateBitmapFromStreamRLE32() added. - Function GUI_CreateBitmapFromStream() supports additional formats. Chapter 12 ’Bitmap Converter’ - New format 'Alpha channel, compressed' added. - New format 'True color with alpha channel, compressed' added. - New function 'Image/Convert Into/Best Palette + transparency' added. Chapter 14 ’Memory Devices’ - New functions GUI_MEMDEV_SetAnimationCallback() added. - New functions GUI_MEMDEV_ShiftInWindow() added. - New functions GUI_MEMDEV_ShiftOutWindow() added. Chapter 15 ’Execution Model’ - New function GUI_SetSignalEventFunc() added. - New function GUI_SetWaitEventFunc() added. - New function GUI_SetWaitEventTimedFunc() added. - Definitions of configuration macros changed. Chapter 16 ’Window Manager’ - New function WM_MULTIBUF_Enable() added. - New messages WM_PRE_PAINT and WM_POST_PAINT added. Chapter 17 ’Widgets’ - LISTVIEW_SetUserData() renamed in LISTVIEW_SetUserDataRow(). - LISTVIEW_GetUserData() renamed in LISTVIEW_GetUserDataRow(). - New function <WIDGET>_SetUserData() added for all widgets. - New function <WIDGET>_GetUserData() added for all widgets. - New function <WIDGET>_CreateUser() added for all widgets.

Table 0.9: Revision history

Software	Rev.	Date	By	Description
5.08	0	110112	AS JE	Chapter 17 ’Widgets’ - New function BUTTON_GetTextAlign() added. - New function BUTTON_SetReactOnLevel() added. - New function ICONVIEW_CreateIndirect() added. - New function ICONVIEW_DeleteItem() added. - New function LISTWHEEL_CreateIndirect() added. - New function SCROLLBAR_SetThumbSizeMin() added. - New function SCROLLBAR_GetThumbSizeMin() added. - New function TREEVIEW_ITEM_CollapseAll() added. - New function TREEVIEW_ITEM_ExpandAll() added. Chapter 19 ’Skinning’ - New configuration macro WIDGET_USE_FLEX_SKIN added. - New message WIDGET_ITEM_GET_RADIUS added to frame window skin. Chapter 20 ’Multiple Buffering’. - New function GUI_MULTIBUF_Begin() added. - New function GUI_MULTIBUF_End() added. - New function GUI_MULTIBUF_Config() added. Chapter 28 ’Foreign Language Support’ - New function GUI_UC_EnableBIDI() added.
5.06	0	100907	JE	Chapter 9 ’Fonts’: - New function GUI_SetDefaultFont() added. Chapter 12 ’Memory Devices’: - New function GUI_MEMDEV_FadeDevices() added. Chapter 15 ’Widgets’: - New functions added: BUTTON_SetReactOnLevel() GRAPH_DATA_XY_SetOwnerDraw() LISTVIEW_SetItemBitmap() LISTWHEEL_SetPos() SCROLLBAR_GetNumItems() SCROLLBAR_GetPageSize() New chapter 17 ’Skinning’: - Skinning for the most common widgets added. Chapter 26 ’Display Driver’: - New function GUI_SetOrientation() added. - New OXY-orientations for 16, 24 and 32 bpp added to GUIDRV_Lin.
5.04	2	100526	AS	Chapter ’Widgets’: - New function GRAPH_DATA_XY_SetOwnerDraw() added. - New function LISTVIEW_SetItemBitmap() added. Chapter ’Fonts’: - New function GUI_SetDefaultFont() added. Chapter ’2-D Graphic Library’: - New function GUI_GetPixelIndex() added. Chapter ’Execution Model’: GUITASK_SetMaxTask() - GUIDRV_CompactColor_16: Support for the following display controllers added: Himax HX8353 LGDP4551 Orisetech SPFD54124C Renesas R61505 Sitronix ST7735, ST7787 Solomon SSD1284, SSD2119 - Added driver macros to each driver which uses them.
5.04	1	100505	AS	New Drivers ’GUIDRV_S1D15G00’ and ’GUIDRV_SLin’ added. Various corrections Chapter ’2-D Graphic Library’: - New function GUI_DrawGradientRoundedV() - New function GUI_DrawGradientRoundedH() - New function GUI_DrawRoundedFrame() Chapter ’Memory Devices’: - New function GUI_MEMDEV_MoveInWindow() - New function GUI_MEMDEV_MoveOutWindow() - New function GUI_MEMDEV_FadeInWindow() - New function GUI_MEMDEV_FadeOutWindow() Chapter ’Simulation’ - New function SIM_GUI_SetCallback() - New function SIM_GUI_ShowDevice()

Table 0.10: Revision history

Software	Rev.	Date	By	Description
5.04	0	100104	JE	Chapter 5 ’Displaying Text’: - New function GUI_DispStringInRectWrap() added. - New function GUI_WrapGetNumLines() added. Chapter 7 ’2-D Graphic Library’: - New function GUI_EnableAlpha() added. - New function GUI_RestoreUserAlpha() added. - New function GUI_SetUserAlpha() added. - New function GUI_CreateBitmapFromStream() added. - New function GUI_DrawStreamedBitmapEx() added. - New function GUI_GetStreamedBitmapInfo() added. - New function GUI_GetStreamedBitmapInfoEx() added. - New function GUI_SetStreamedBitmapHook() added. - New function GUI_CreateBitmapFromStreamIDX() added. - New function GUI_CreateBitmapFromStreamRLE4() added. - New function GUI_CreateBitmapFromStreamRLE8() added. - New function GUI_CreateBitmapFromStream565() added. - New function GUI_CreateBitmapFromStreamM565() added. - New function GUI_CreateBitmapFromStream555() added. - New function GUI_CreateBitmapFromStreamM555() added. - New function GUI_CreateBitmapFromStreamRLE16() added. - New function GUI_CreateBitmapFromStreamRLEM16() added. - New function GUI_CreateBitmapFromStream24() added. - New function GUI_CreateBitmapFromStreamAlpha() added. Chapter 9 ’Fonts’: - New font F20F_ASCII (framed) added. - New fonts F6x8_ASCII and F6x8_1 added. - New fonts F8x8_ASCII and F8x8_1 added. - New fonts F8x16_ASCII and F8x16_1 added. - Support for new font formats extended AA2 and extended AA4 added. Chapter 12 ’Memory Devices’: - Considerations for multiple layers/displays added. Chapter 14 ’Window Manager’: - WM_DeleteWindow() now also deletes any associated timer. Chapter 15 ’Widgets’: - New function WINDOW_SetBkColor() added. Chapter 19 ’Pointer Input Devices’: - PID buffer added. - Explanation of touch calibration revised. Chapter 20 ’Keyboard’: - Keyboard buffer added. Chapter 25 ’Display Driver’: - New driver GUIDRV_BitPlains added. - New driver GUIDRV_SLin added. - New driver GUIDRV_SSD1926 added. - Driver GUIDRV_1611 added. - Driver GUIDRV_6331 added. - Driver GUIDRV_7529 added. - Driver GUIDRV_Page1bpp added. - GUIDRV_CompactColor_16: Support for the following display controllers added: Himax HX8340, HX8352 Solomon SSD1298, SSD1355, SSD1963 Epson S1D19122 Orisetech SPFD5414D Ilitek ILI9320, ILI9326 Chapter 26 ’VNC Server’: - New function GUI_VNC_EnableKeyboardInput() - New function GUI_VNC_GetNumConnections() - New function GUI_VNC_SetPassword() - New function GUI_VNC_SetProgName() - New function GUI_VNC_SetSize() - New function GUI_VNC_RingBell()
5.00	1	090409	JE	Chapter 3 ’Simulator’: - Completely revised. Chapter 8 ’Displaying bitmap files’ - PNG support added.
5.00	0	090326	JE	Software has been completely revised. For the version history of earlier versions, refer to older documents.

Table 0.11: Revision history

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler)
- The C programming language
- The target processor
- DOS command line

If you feel that your knowledge of C is not sufficient, we recommend The C Programming Language by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C-programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command-prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Example code in program examples.
`Sample comment`	Comments in program examples.
`New Example`	Example code that has been added to an existing program example.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Warning	Important cautions or reminders.
Emphasis	Very important sections.

Table 0.1: Typographic conventions

SEGGER Microcontroller GmbH & Co. KG develops and distributes software development tools and ANSI C software components (middleware) for embedded systems in several industries such as telecom, medical technology, consumer electronics, automotive industry and industrial automation.

SEGGER's intention is to cut software development time for embedded applications by offering compact flexible and easy to use middleware, allowing developers to concentrate on their application.

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet efficient real-time kernel. emWin, written entirely in ANSI C, can easily be used on any CPU and most any display. It is complemented by the available PC tools: Bitmap Converter, Font Converter, Simulator and Viewer. embOS supports most 8/16/32-bit CPUs. Its small memory footprint makes it suitable for single-chip applications.

Apart from its main focus on software tools, SEGGER develops and produces programming tools for flash micro controllers, as well as J-Link, a JTAG emulator to assist in development, debugging and production, which has rapidly become the industry standard for debug access to ARM cores.

Corporate Office:
http://www.segger.com

United States Office:
http://www.segger-us.com

EMBEDDED SOFTWARE (Middleware)

emWin
Graphics software and GUI
emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display.

embOS
Real Time Operating System
embOS is an RTOS designed to offer the benefits of a complete multitasking system for hard real time applications with minimal resources.

embOS/IP
TCP/IP stack
embOS/IP a high-performance TCP/IP stack that has been optimized for speed, versatility and a small memory footprint.

emFile
File system
emFile is an embedded file system with FAT12, FAT16 and FAT32 support. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.

USB-Stack
USB device/host stack
A USB stack designed to work on any embedded system with a USB controller. Bulk communication and most standard device classes are supported.

SEGGER TOOLS

Flasher
Flash programmer
Flash Programming tool primarily for micro controllers.

J-Link
JTAG emulator for ARM cores
USB driven JTAG interface for ARM cores.

J-Trace
JTAG emulator with trace
USB driven JTAG interface for ARM cores with Trace memory. supporting the ARM ETM (Embedded Trace Macrocell).

J-Link / J-Trace Related Software
Add-on software to be used with SEGGER's industry standard JTAG emulator, this includes flash programming software and flash breakpoints.

Table of Contents

Chapter 1

Introduction to emWin

This introduction gives some information about this document. It also gives an overview of what features emWin consists of and what it requires.

1.1 Purpose of this document

This guide describes how to install, configure and use the emWin graphical user interface for embedded applications. It also explains the internal structure of the software and all the functions which are offered by emWin and intended for direct use (API, Application Programming Interface). Before actually using emWin, you should read or at least glance through this manual in order to become familiar with the software. The following steps are recommended:

- Copy the emWin files to your computer.
- Go through the chapter "Getting Started" on page 41.
- Use the simulator in order to become more familiar with what the software can do (refer to the chapter "Simulation" on page 51).
- Expand your program using the rest of the manual for reference.

1.2 Requirements

A target system is not required in order to develop software with emWin; most of the software can be developed using the simulator. However, the final purpose is usually to be able to run the software on a target system.

1.2.1 Target system (hardware)

Your target system must:

- Have a CPU (8/16/32/64 bits)
- Have a minimum of RAM and ROM
- Have a full graphic display (any type and any resolution)

The RAM needs to be 8-, 16- and 32-bit accessible. Memory requirements vary depending on which parts of the software are used and how efficient your target compiler is. It is therefore not possible to specify precise values, but the following applies to typical systems.

Small systems (no Window Manager)

- RAM: 100 Bytes
- Stack: 600 Bytes
- ROM: 10-25 KBytes (depending on the functionality used)

Big systems (including Window Manager and widgets)

- RAM: 2-6 kb (depending on number of windows required)
- Stack: 1200-1800 bytes (depending on the functionality used)
- ROM: 30-60 kb (depending on the functionality used)

ROM requirements increase according to the number of fonts used in the application. All values are rough estimates and cannot be guaranteed. Detailed information can be found in the chapter "Performance and Resource Usage" on page 1195.

1.2.2 Development environment (compiler)

The CPU used is of no importance; only an ANSI-compliant C compiler complying with at least one of the following international standard is required:

- ISO/IEC/ANSI 9899:1990 (C90) with support for C++ style comments (//)
- ISO/IEC 9899:1999 (C99)
- ISO/IEC 14882:1998 (C++)

If your compiler has some limitations, let us know and we will inform you if these will be a problem when compiling the software. Any compiler for 16/32/64-bit CPUs or DSPs that we know of can be used; most 8-bit compilers can be used as well. A C++ compiler is not required, but can be used. The application program can therefore also be programmed in C++ if desired.

1.3 Features

emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface for any application that operates with a graphical display. It is compatible with single-task and multitask environments, with a proprietary operating system or with any commercial RTOS. emWin is shipped as C source code. It may be adapted to any size physical and virtual display with any display controller and CPU. Its features include the following:

General

- Any (monochrome, grayscale or color) display with any controller supported (if the right driver is available).
- May work without display controller on smaller displays.
- Any interface supported using configuration macros.
- Display-size configurable.
- Characters and bitmaps may be written at any point on the display, not just on even-numbered byte addresses.
- Routines are optimized for both size and speed.
- Compile time switches allow for different optimizations.
- For slower display controllers, display can be cached in memory, reducing access to a minimum and resulting in very high speed.
- Clear structure.
- Virtual display support; the virtual display can be larger than the actual display.

Graphic library

- Bitmaps of different color depths supported.
- Bitmap Converter available.
- Absolutely no floating-point usage.
- Fast line/point drawing (without floating-point usage).
- Very fast drawing of circles/polygons.
- Different drawing modes.

Fonts

- A variety of different fonts are shipped with the basic software: 4*6, 6*8, 6*9, 8*8, 8*9, 8*16, 8*17, 8*18, 24*32, and proportional fonts with pixel-heights of 8, 10, 13, 16. For more information, see chapter 'Fonts'.
- New fonts can be defined and simply linked in.
- Only the fonts used by the application are actually linked to the resulting executable, resulting in minimum ROM usage.
- Using the Font Converter, any font available on the host system (that is, Microsoft Windows) can be converted for use in emWin.
- Scalable iType and TTF fonts are supported.

String/value output routines

- Routines to show values in decimal, binary, hexadecimal, any font.
- Routines to edit values in decimal, binary, hexadecimal, any font.

Window Manager (WM)

- Complete window management including clipping. Overwriting of areas outside a window's client area is impossible.
- Windows can be moved and resized.
- Callback routines supported (usage optional).
- WM uses minimum RAM (app. 50 bytes per window).

Optional widgets for PC look and feel

- Widgets (window objects, also known as controls) are available. They generally operate automatically and are simple to use.

Touch-screen & mouse support

- For window objects such as the button widget, emWin offers touch-screen and mouse support.

PC tools

- Simulation library for WIN32-Environments. The source code may be purchased additionally.
- emWinView.
- Bitmap Converter.
- Font Converter.
- GUIBuilder.

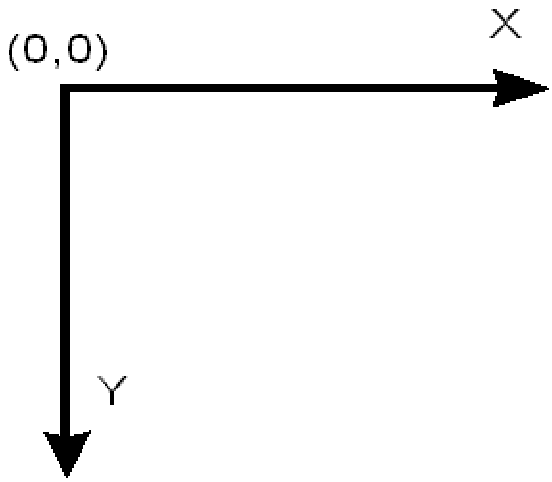
1.4 Examples and demos

To give you a better idea of what emWin can do, we have different demos available as "ready-to-use" simulation executables under
`www.segger.com/emwin-samples.html`.
The source of the sample applications is located in the folder `Sample`. The folder `Sample\Application\GUIDemo`
contains an application program showing many features of emWin. All examples are also available at `www.segger.com`. Example code in this documentation is provided as code snippet, which might require further modifications.

1.5 Starter kits

Complete starter kits including a demo board with a display, a C compiler and an example project are available. For more details, take a look at our website at www.segger.com.

1.6 Screen and coordinates



The screen consists of many dots that can be controlled individually. These dots are called pixels. Most of the text and drawing functions that emWin offers in its API to the user program can write or draw on any specified pixel.
The horizontal scale is called the X-axis, whereas the vertical scale is called the Y-axis. Coordinates are denoted as a pair consisting of an X- and a Y-value (X, Y). The X-coordinate is always first in routines that require X and Y coordinates. The upper left corner of the display (or a window) has per default the coordinates (0,0). Positive X-values are always to the right; positive Y-values are always down. The above graph illustrates the coordinate system and directions of the X- and Y- axes. All coordinates passed to an API function are always specified in pixels.

1.7 How to connect the display to the microcontroller

emWin handles all access to the display. Virtually any display controller can be supported, independently of how it is accessed. For details, refer to the chapter "Configuration" on page 1203. Also, get in contact with us if your display controller is not supported. We are currently writing drivers for all display controllers available on the market and may already have a proven driver for the display controller that you intend to use. It is usually very simple to write the routines (or macros) used to access the display in your application SEGGER Microcontroller GmbH & Co. KG offers the service of making these customizations for you, if necessary with your target hardware.
It does not really matter how the display is connected to the system as long as it is somehow accessible by software, which may be accomplished in a variety of ways. Most of these interfaces are supported by a driver which is supplied in source code form. This driver does not normally require modifications, but is configured for your hardware by making changes in the file `LCDConf.h`. Details about how to customize a driver to your hardware as necessary are provided in the chapter "Display drivers" on page 1055. The most common ways to access the display are described as follows. If you simply want to understand how to use emWin, you may skip this section.

Display with memory-mapped display controller:

The display controller is connected directly to the data bus of the system, which means the controller can be accessed just like a RAM. This is a very efficient way of accessing the display controller and is most recommended. The display addresses are defined to the segment LCDSEG, and in order to be able to access the display the linker/locator simply needs to be told where to locate this segment. The location must be identical to the access address in physical address space. Drivers are available for this type of interface and for different display controllers.

Display with display controller connected to port / buffer

For slower display controllers used on fast processors, the use of port-lines may be the only solution. This method of accessing the display has the disadvantage of being somewhat slower than direct bus-interface but, particularly with a cache that minimizes the accesses to the display, the display update is not slowed down significantly. All that needs to be done is to define routines or macros which set or read the hardware ports/buffers that the display is connected to. This type of interface is also supported by different drivers for the different display controllers.

Proprietary solutions: display without display controller

The display can also be connected without an display controller. In this case, the display data is usually supplied directly by the controller via a 4- or 8-bit shift register. These proprietary hardware solutions have the advantage of being inexpensive, but the disadvantage of using up much of the available computation time. Depending on the CPU, this can be anything between 20 and almost 100 percent; with slower CPUs, it is really not possible at all. This type of interface does not require a specific display driver because emWin simply places all the display data into the display cache. You yourself must write the hardware-dependent portion that periodically transfers the data in the cache memory to your display.
Example code for transferring the video image into the display is available in both C and optimized assembler for M16C and M16C/80.

1.8 Data types

Since C does not provide data types of fixed lengths which are identical on all platforms, emWin uses, in most cases, its own data types as shown in the table below:

Data type	Definition	Description
I8	signed char	8-bit signed value
U8	unsigned char	8-bit unsigned value
I16	signed short	16-bit signed value
U16	unsigned short	16-bit unsigned value
I32	signed long	32-bit signed value
U32	unsigned long	32-bit unsigned value
I16P	signed short	16-bit (or more) signed value
U16P	unsigned short	16-bit (or more) unsigned value

Table 1.1: Data types

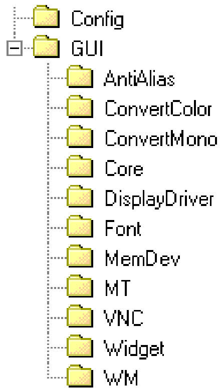
For most 16/32-bit controllers, the settings will work fine. However, if you have similar defines in other sections of your program, you might want to change or relocate them. A recommended place is in the file `Global.h`.

Chapter 2

Getting Started

The following chapter provides an overview of the basic procedures for setting up and configuring emWin on your target system. It also includes a simple program example.
If you find yourself unsure about certain areas, keep in mind that most topics are treated in greater detail in later chapters. You will most likely need to refer to other parts of the manual before you begin more complicated programming.

2.1 Recommended project structure



We recommend keeping emWin separate from your application files. It is good practice to keep all the program files (including the header files) together in the GUI subdirectories of your project's root directory. The directory structure should be similar to the one pictured on the right. This practice has the advantage of being very easy to update to newer versions of emWin by simply replacing the GUI\ directories. Your application files can be stored anywhere.

Warning: When updating to a newer emWin version: Since files may have been added, moved or deleted, the project directories may need to be updated accordingly.

2.2 Subdirectories

The following table shows the contents of all GUI subdirectories:

Directory	Contents
Config	Configuration files
GUI\AntiAlias	Antialiasing support *
GUI\ConvertMono	Color conversion routines used for grayscale displays *
GUI\ConvertColor	Color conversion routines used for color displays *
GUI\Core	emWin core files
GUI\Font	Font files
GUI\DisplayDriver	Display driver
GUI\MemDev	Memory Device support *
GUI\MT	MultiTouch support
GUI\VNC	VNC support *
GUI\Widget	Widget library *
GUI\WM	Window Manager *

Table 2.1: Subdirectories

(* = optional)

2.2.1 Include directories

You should make sure that the include path contains the following directories (the order of inclusion is of no importance):

- Config
- GUI\Core
- GUI\DisplayDriver
- GUI\Widget (if using the widget library)
- GUI\WM (if using Window Manager)

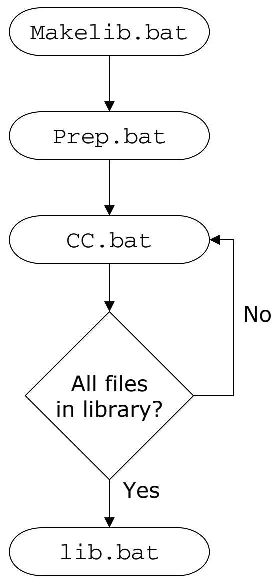
Warning: Always make sure that you have only one version of each file!

It is frequently a major problem when updating to a new version of emWin if you have old files included and therefore mix different versions. If you keep emWin in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the `GUI\` directories prior to updating.

2.3 Adding emWin to the target program

You basically have a choice between including only the source files that you are actually going to use in your project, which will then be compiled and linked, or creating a library and linking the library file. If your tool chain supports "smart" linking (linking in only the modules that are referenced and not those that are not referenced), there is no real need to create a library at all, since only the functions and data structures which are required will be linked. If your tool chain does not support "smart" linking, a library makes sense, because otherwise everything will be linked in and the program size will be excessively large. For some CPUs, we have example projects available to help you get started.

2.4 Creating a library



Building a library from the sources is a simple procedure. The first step is to copy the batch files (located under `Sample\Makelib`) into your project's root directory. That means the parent directory containing the 'Config' and the 'GUI' folder. Then, make any necessary changes. There are a total of four batch files which need to be copied, described in the table below. The main file, Makelib.bat, will be the same for all systems and requires no changes. To build a library for your target system, you will normally need to make slight modifications to the other three smaller files. Finally, start the file Makelib.bat to create the library. The batch files assume that your GUI and Config subdirectories are set up as recommended.
The procedure for creating a library is illustrated in the flow chart to the right. The Makelib.bat file first calls Prep.bat to prepare the environment for the tool chain. Then it calls CC.bat for every file to be included in the library. It does this as many times as necessary. CC.bat adds each object file to a list that will be used by lib.bat. When all files to be added to the library have been listed, Makelib.bat then calls lib.bat, which uses a librarian to put the listed object files into the actual library. Of course you are free to create libraries in another way.
It is not recommended to create an emWin library including a compile-time configurable display driver. Detailed information about the configurability of emWin display drivers can be found in the section "Available display drivers" on page 988.

File	Description
Makelib.bat	Main batch file. No modification required.
Prep.bat	Called by Makelib.bat to prepare environment for the tool chain to be used.
CC.bat	Called by Makelib.bat for every file to be added to the library; creates a list of these object files which will then be used in the next step by the librarian in the lib.bat file.
Lib.bat	Called by Makelib.bat to put the object files listed by CC.bat into a library.

Table 2.2: Batch files required for library creation

The files as shipped assume that a Microsoft compiler is installed in its default location. If all batch files are copied to the root directory (directly above GUI) and no changes are made at all, a simulation library will be generated for the emWin simulation. In order to create a target library, however, it will be necessary to modify `Prep.bat`, `CC.bat`, and `lib.bat`.

2.4.1 Adapting the library batch files to a different system

The following will show how to adapt the files by an example adaptation for a Mitsubishi M32C CPU.

Adapting Prep.bat

Prep.bat is called at the beginning of Makelib.bat. As described above its job is to set the environment variables for the used tools and the environment variable PATH, so that the batch files can call the tools without specifying an absolute path. Assuming the compiler is installed in the folder C:\MTOOL the file Prep.bat could look as follows:

```
@ECHO OFF
SET TOOLPATH=C:\MTOOL
REM *****************************************************************
REM   Set the variable PATH to be able to call the tools
SET PATH=%TOOLPATH%\BIN;%TOOLPATH%\LIB308;%PATH%
REM *****************************************************************
REM   Set the tool internal used variables
SET BIN308=%TOOLPATH%\BIN
SET INC308=%TOOLPATH%\INC308
SET LIB308=%TOOLPATH%\LIB308
SET TMP308=%TOOLPATH%\TMP
```

Adapting CC.bat

The job of `CC.bat` is to compile the passed source file and adding the file name of the object file to a link list. When starting `MakeLib.bat` it creates the following subdirectories relative to its position:

Directory	Contents
`Lib`	This folder should contain the library file after the build process.
`Temp\Output`	Should contain all the compiler output and the link list file. Will be deleted after the build process.
`Temp\Source`	`MakeLib.bat` uses this folder to copy all source and header files used for the build process. Will be deleted after the build process.

Table 2.3: Folders created by MakeLib.bat

The object file should be created (or moved) to `Temp\Output`. This makes sure all the output will be deleted after the build process. Also the link list should be located in the output folder. The following shows an example for the Mitsubishi compiler:

```
@ECHO OFF
GOTO START
REM *****************************************************************
REM   Explanation of the used compiler options:
-silent : Suppresses the copyright message display at startup
-M82    : Generates object code for M32C/80 Series (Remove this switch
          for M16C80 targets)
-c      : Creates a relocatable file (extension .r30) and ends processing
-I      : Specifies the directory containing the file(s) specified in #include
-dir    : Specifies the destination directory
-OS     : Maximum optimization of speed followed by ROM size
-fFRAM  : Changes the default attribute of RAM data to far
-fETI   : Performs operation after extending char-type data to the int type
          (Extended according to ANSI standards)
:START
REM *****************************************************************
REM   Compile the passed source file with the Mitsubishi NC308 compiler
NC308 -silent -M82 -c -IInc -dir Temp\Output -OS -fFRAM -fETI Temp\Source\%1.c
REM *****************************************************************
```

```
REM   Pause if any problem occurs
IF ERRORLEVEL 1 PAUSE
REM ******************************************************************
REM   Add the file name of the object file to the link list
ECHO Temp\Output\%1.R30>>Temp\Output\Lib.dat
```

Adapting Lib.bat

After all source files have been compiled `Lib.bat` will be called from `MakeLib.bat`. The job is to create a library file using the link list created by `CC.bat`. The destination folder of the library file should be the Lib folder created by `MakeLib.bat`. The following shows an example for the Mitsubishi librarian:

```
@ECHO OFF
GOTO START
REM ******************************************************************
REM   Explanation of the used options:
-C : Creates new library file
@  : Specifies command file
:START
REM ******************************************************************
REM   Create the first part of the linker command file
ECHO -C Lib\GUI>Temp\Output\PARA.DAT
REM ******************************************************************
REM   Merge the first part with the link list to the linker command file
COPY Temp\Output\PARA.DAT+Temp\Output\Lib.dat Temp\Output\LINK.DAT
REM ******************************************************************
REM   Call the Mitsubishi librarian
LB308 @Temp\Output\LINK.DAT
REM ******************************************************************
REM   Pause if any problem occurs
IF ERRORLEVEL 1 PAUSE
```


2.5 C files to include in the project

Generally speaking, you need to include the core C files of emWin, the display driver, all font files you plan to use and any optional modules you have ordered with emWin:

- All C files of the folder Config
- All C files of the folder GUI\Core
- The fonts you plan to use (located in GUI\Font)
- Display driver: All C files of the folder GUI\DisplayDriver.

Additional software packages

If you plan to use additional, optional modules you must also include their C files:

- Gray scale converting functions: all C files located in GUI\ConvertMono
- Color conversion functions: all C files located in GUI\ConvertColor
- Antialiasing: all C files located in GUI\AntiAlias
- Memory Devices: all C files located in GUI\MemDev
- VNC support: all C files located in GUI\VNC
- Widget library: all C files located in GUI\Widget
- Window Manager: all C files located in GUI\WM

Target specifics

For displays with indirect interface hardware routines must be included. Examples for several kinds of indirect interface routines are available under Sample\LCD_X_Port.

RTOS specifics

- If emWin is intended to be used with an RTOS, some RTOS dependent functions need to be implemented. emWin comes with several sample files including implementations for common RTOS packages (called GUI_X_<RTOS>.c), as well as the file GUI_X_Ex.c which just contains place holders of the required functions and might be used to make emWin work with any RTOS.
- If multitasking is not required (access of the display by one task only) the file GUI_X.c may be used as a starting point for a custom implementation.

The sample files can be found in the folder Sample\GUI_X which is contained in the emWin package.

Additional information

Be sure to include GUI.h in all emWin accessing source files.

2.6 Configuring emWin

The `Config` folder should contain all configuration files. The chapter 'Configuration' explains in detail how emWin should be configured.
The following types of configuration macros are available:

Binary switch "B"

Switches can have a value of either 0 or 1, where 0 means deactivated and 1 means activated (actually anything other than 0 would work, but using 1 makes it easier to read a config file). These switches can enable or disable a certain functionality or behavior. Switches are the simplest form of configuration macro.

Numerical value "N"

Numerical values are used somewhere in the code in place of a numerical constant. Typical examples are in the configuration of the resolution of a display.

Selection switch "S"

Selection switches are used to select one out of multiple options where only one of those options can be selected. A typical example might be the selection of the type of display controller used, where the number selected denotes which source code (in which display driver) is used to generate object code.

Alias "A"

A macro which operates like a simple text substitute. An example is `U8`, which is replaced by the preprocessor with `unsigned char`.

Function replacement "F"

Macros can basically be treated like regular functions although certain limitations apply, as a macro is still put into the code as simple text replacement. Function replacements are mainly used to add specific functionality to a module (such as the access to a display) which is highly hardware-dependent. This type of macro is always declared using brackets (and optional parameters).

Type replacement "T"

Type replacement macros allow changing the types of certain values.

2.7 Initializing emWin

The following functions should be used to initialize and 'deinitialize' emWin in order to start the configuration process (see chapter "Configuration" on page 1203) or clear internal data from memory again.

Routine	Description
GUI_Init()	Initializes emWin internal data structures and variables.
GUI_Exit()	Clears emWin internal data from memory.

Table 2.4: Initialize and deinitialize functions

GUI_Init()

Description

Initializes emWin internal data structures and variables.

Prototype

```
int GUI_Init(void);
```

Return value

0, if successful; another value if the initialization of the display driver fails.

Additional information

Executing this function is mandatory before using any emWin functions. The only exception is setting create flags for windows (see "WM_SetCreateFlags()" on page 426). If the Window Manager is used, the background window is created from within GUI_Init(). So if create flags are set up before GUI_Init() is called, the background window is created according to them.

GUI_Exit()

Description

Clears emWin internal data from memory to make further calls of GUI_Init() possible.

Prototype

```
void GUI_Exit(void);
```

Additional information

This function should be used if emWin represents a part of the application which is not used continuously and therefore has to be able to be turned on and off again. Please note that after `GUI_Exit()` was called emWin will not work properly until `GUI_Init()` is called again.

2.8 Using emWin with target hardware

The following is just a basic outline of the general steps that should be taken when starting to program with emWin. All steps are explained further in subsequent chapters.

Step 1: Configuring emWin

The first step is usually to customize emWin. For details about the configuration, refer to the chapter "Configuration" on page 1203".

Step 2: Defining access addresses or access routines

For memory-mapped display controllers, the access addresses of the display simply need to be defined in the configuration file of the display controller. For port/buffer-accessed display controllers, interface routines must be defined. Examples of the required routines are available under `Sample\LCD_X_Port`.

Step 3: Compiling, linking and testing the example code

emWin comes with example code for both single- and multitask environments. Compile, link and test these little example programs until you feel comfortable doing so.

Step 4: Modifying the example program

Make simple modifications to the example programs. Add additional commands such as displaying text in different sizes on the display, showing lines and so on.

Step 5: In multitask applications: adapt to your OS (if necessary)

If multiple tasks should be able to access the display simultaneously, the macros `GUI_MAXTASK` and `GUI_OS` come into play, as well as the file `GUITask.c`. For details and example adaptations, refer to the chapter "Configuration" on page 1203".

Step 6: Write your own application using emWin

By now you should have a clearer understanding of how to use emWin. Think about how to structure the program your application requires and use emWin by calling the appropriate routines. Consult the reference chapters later in this manual, as they discuss the specific emWin functions and configuration macros that are available.

2.9 The "Hello world" example program

In the following we will show the "Hello world" example program. If you like to see a wide range of emWin based sample applications as well as further simple tutorial applications, please have a look in the `Sample` folder of your emWin shipment or visit the "emWin Samples" section on `www.segger.com`.

A "Hello world" program has been used as a starting point for C programming since the early days, because it is essentially the smallest program that can be written. An emWin "Hello world" program is shown below and is available as `BASIC_HelloWorld.c` in the `Sample\Tutorial` folder shipped with emWin.
The whole purpose of the program is to write "Hello world" in the upper left corner of the display. In order to be able to do this, the hardware of the application, the display controller and the GUI must be initialized first. emWin is initialized by a simple call of `GUI_Init()` in the beginning of the program. In this example, we assume that the hardware of your application is already initialized.
The “Hello world” program looks as follows:

```
#include "GUI.h"

void MainTask(void) {
  GUI_Init();
  GUI_DispString("Hello world!");
  while(1);
}
```

Adding functionality to the "Hello world" program

Our little program has not been doing too much so far. We can now extend the functionality a bit: after displaying "Hello world", we would like the program to start counting on the display in order to be able to estimate how fast outputs to the display can be made. We can simply add a bit of code to the loop at the end of the main program, which is essentially a call to the function that displays a value in decimal form.
The example is available as `BASIC_Hello1.c` in the `Sample` folder.

```
#include "GUI.h"

void MainTask(void) {
  int i = 0;

  GUI_Init();
  GUI_DispString("Hello world!");
  while(1) {
    GUI_DispDecAt( i++, 20,20,4);
    if (i > 9999) {
      i = 0;
    }
  }
}
```

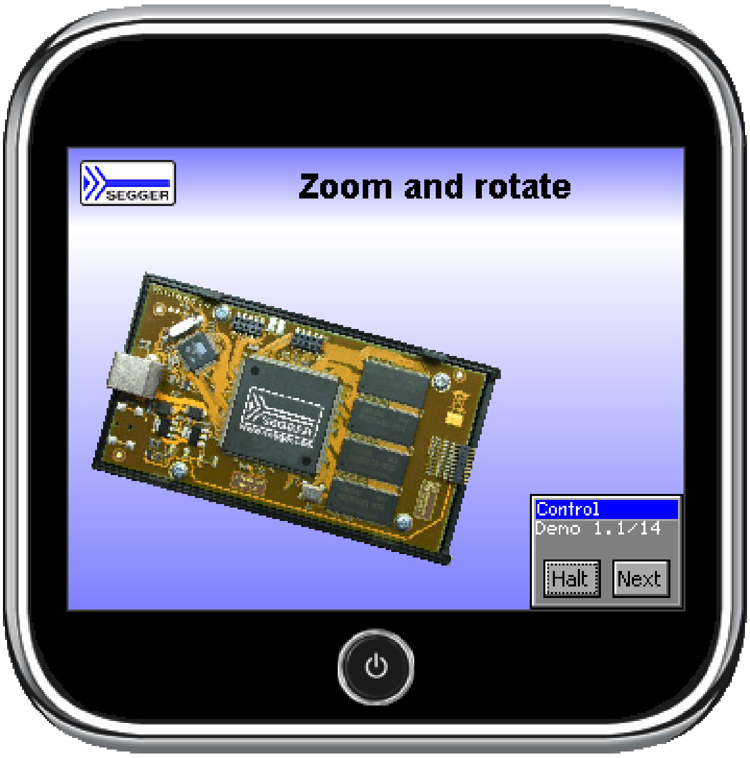

Chapter 3

Simulation

The PC simulation of emWin allows you to compile the same C source on your Windows PC using a native (typically Microsoft) compiler and create an executable for your own application. Doing so allows the following:

- Design of the user interface on your PC (no hardware required!).
- Debugging of the user interface program.
- Creation of demos of your application, which can be used to discuss the user interface.

The resulting executable can be sent easily via e-mail.



3.1 Using the simulation

The emWin simulation requires Microsoft Visual C++ (version 6.00 or higher) and the integrated development environment (IDE) which comes with it. You will see a simulation of your LCD on your PC screen, which has the same resolution in X and Y and can display the exact same colors as your LCD once it has been properly configured. The entire graphic library API and Window Manager API of the simulation are identical to those on your target system; all functions will behave in the very same way as on the target hardware since the simulation uses the same C source code as the target system. The difference lies only in the lower level of the software: the display driver. Instead of using the actual display driver, the PC simulation uses a simulation driver which writes into a bitmap. The bitmap is then displayed on your screen using a second thread of the simulation. This second thread is invisible to the application; it behaves just as if the LCD routines were writing directly to the display.

3.1.1 Rotating and mirroring the screen

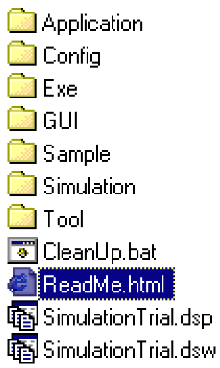
emWin supports rotating and/or mirroring of the screen. Please note that these features do not affect the simulation screen.

3.1.2 Using the simulation with the trial version of emWin

The trial version of emWin contains a full library which allows you to evaluate all available features of emWin. It also includes the emWin viewer (used for debugging applications), as well as demo versions of the Font Converter and the Bitmap Converter. Keep in mind that, being a trial version, you will not be able to view the source code of emWin or the simulation, but you will still be able to become familiar with what emWin can do.

3.1.2.1 Directory structure

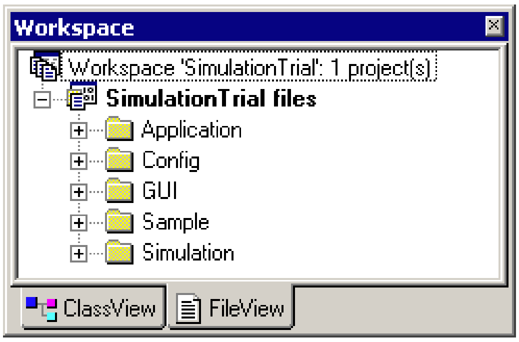
The directory structure of the simulation in the trial version is shown at the right side. The table below explains the contents of the folders:



Directory	Content
Application	Source of the demo program.
Config	Configuration files used to build the library. Note that changes at the header files do not have any effect on the precompiled library!
Exe	Ready-to-use demo program.
GUI	Library files and include files need to use the library.
Sample	Simulation examples.
Simulation	Files needed for the simulation.
Tool	The emWin viewer, a demo version of the Bitmap Converter and a demo version of the Font Converter.

Table 3.1: Directory structure emWin trial version

3.1.2.2 Visual C++ workspace



The root directory shown above includes the Microsoft Visual C++ workspace (SimulationTrial.dsw) and project file (SimulationTrial.dsp). The workspace allows you to modify an application program and debug it before compiling it on your target system. Double-click the workspace file to open the Microsoft IDE. The directory structure of the Visual C++ workspace will look like the one shown to the right.

3.1.2.3 Compiling the demo program

The source files for the demo program are located in the `Application` directory as a ready-to-go simulation, meaning that you only need to rebuild and start it. Note that to rebuild the executable, you will need to have Microsoft Visual C++ (version 6.00 or later) installed.

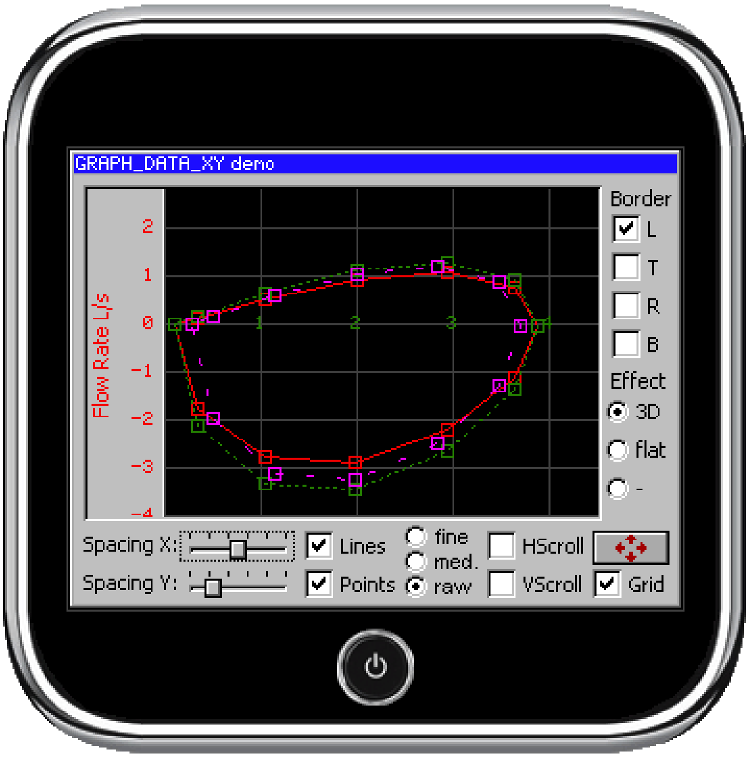
- Step 1: Open the Visual C++ workspace by double-clicking on `SimulationTrial.dsw`.
- Step 2: Rebuild the project by choosing `Build/Rebuild All` from the menu (or by pressing F7).
- Step 3: Start the simulation by choosing `Build/Start Debug/Go` from the menu (or by pressing F5).

The demo project will begin to run and may be closed at any time by right-clicking on it and selecting `Exit`.

3.1.2.4 Compiling the samples

The `Sample` directory contains ready-to-go examples that demonstrate different features of emWin and provide examples of some of their typical uses. In order to build any of these executables, their C source must be 'activated' in the project. This is easily done with the following procedure:

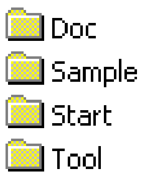
- Step 1: Exclude the `Application` folder from the build process by right-clicking the `Application` folder of the workspace and selecting '`Settings\General\Exclude from build`'.
- Step 2: Open the `Sample` folder of the workspace by double-clicking on it. Include the example which should be used by right-clicking on it and deselecting '`Settings\General\Exclude`' from build.
- Step 3: If the example contains its own configuration files (`LCDConf.c` and/or `SIMConf.c`) the default configuration files located in the config folder need to be excluded from the build process.
- Step 4: Rebuild the example by choosing '`Build\Rebuild All`' from the menu (or by pressing F7).
- Step 5: Start the simulation by choosing '`Build/Start Debug/Go`' from the menu (or by pressing F5). The result of the example selected above is pictured below:



3.1.3 Using the simulation with the emWin source

3.1.3.1 Directory structure

The root directory of the simulation can be anywhere on your PC, for example `C:\Work\emWinSim`. The directory structure will appear as shown to the right. This structure is very similar to that which we recommend for your target application (see Chapter 3: "Getting Started" for more information).

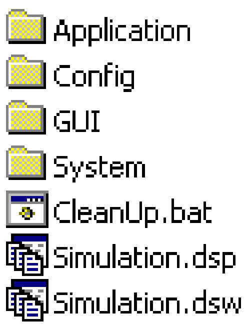


The following table shows the contents of the folders:

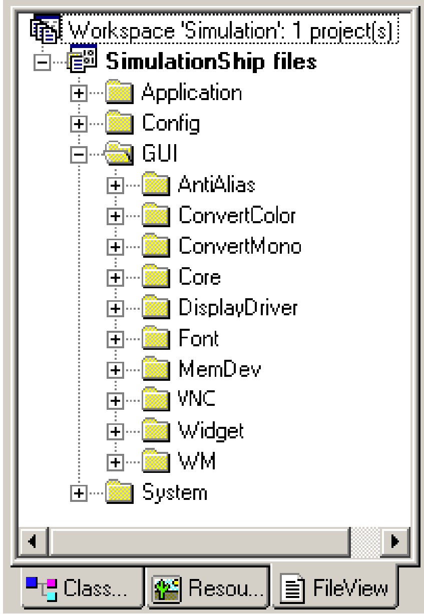
Directory	Content
Doc	Contains the emWin documentation.
Sample	Code examples, described later in this documentation.
Start	All you need to create a new project with emWin.
Tool	Tools shipped with emWin.

Table 3.2: Directory structure emWin source

A new project can be started by making a copy of the Start-folder. It contains all required files for a new project. Subdirectories containing the emWin sources are in the `Start\GUI` folder and should contain the exact same files as the directories of the same names which are used for your target (cross) compiler. The files of the `GUI` subdirectories should not be changed, as this would make updating to a newer version of emWin more difficult.
The Start\`Config` directory contains configuration files which need to be modified in order to reflect your target hardware settings (mainly LCD-size and colors which can be displayed).



3.1.3.2 Visual C++ workspace



The root directory shown above includes the Microsoft Visual C++ workspace (`Simulation.dsw`) and project files (`Simulation.dsp`). The workspace allows you to modify an application program and debug it before compiling it on your target system.
The directory structure of the Visual C++ workspace will appear similar to that shown to the right. Here, the `GUI` folder is open to display the emWin subdirectories. Note that your `GUI` directory may not look exactly like the one pictured, depending on which additional features of emWin you have. The folders `Core`, `Font` and `DisplayDriver` are part of the basic emWin package and will always appear in the workspace directory.

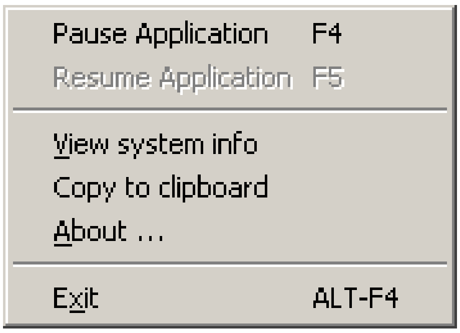
3.1.3.3 Compiling the application

The simulation contains one or more application C files (located in the `Application` directory), which can be modified or removed and additional files can be added to the project. You should then rebuild the program within the Visual C++ workspace in order to test/debug it. Once you have reached a point where you are satisfied with the result and want to use the program in your application, you should be able to compile these same files on your target system and get the same result on the target display. The general procedure for using the simulation would be as follows:

- Step 1: Open the Visual C++ workspace by double-clicking on `Simulation.dsw`.
- Step 2: Compile the project by choosing `Build/Rebuild All` from the menu (or by pressing F7).
- Step 3: Run the simulation by choosing `Build/Start Debug/Go` from the menu (or by pressing F5).
- Step 4: Replace the bitmap with your own logo or image.
- Step 5: Make further modifications to the application program as you wish, by editing the source code or adding/deleting files.
- Step 6: Compile and run the application program within Visual C++ to test the results. Continue to modify and debug as needed.
- Step 7: Compile and run the application program on your target system.

3.1.4 Advanced features of the simulation

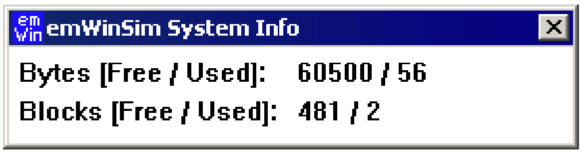
Clicking the right mouse button shows a context menu with several advanced functions:



3.1.4.1 Pause and Resume

These menu items allows to pause and to resume the application currently running in the simulation. The same can be done by pressing <F4> or <F5>. Trying to pause an already paused application or trying to resume an already running application causes an error message.

3.1.4.2 View system info



This menu item opens a further window with information of the memory currently used by the application. The window continuously shows the current status of memory consumption by showing the free and used bytes and the free and used number of memory blocks.

3.1.4.3 Copy to clipboard

This menu item copies the current contents of the display into the clipboard. This makes it easy to use it for documentation purpose with other applications.

3.2 Device simulation

The device simulation supports 3 views:

- Generated frame view
- Custom bitmap view
- Window view

The table below shows the different views:

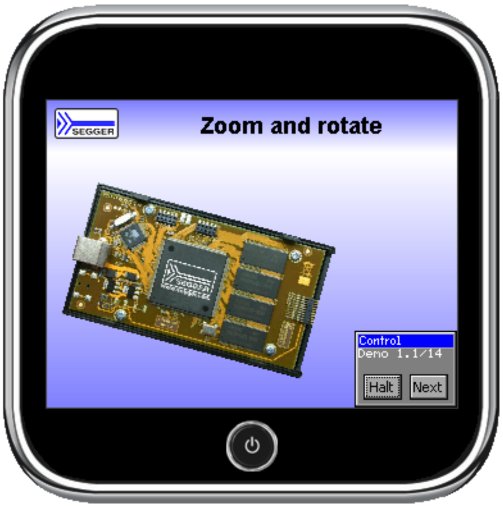
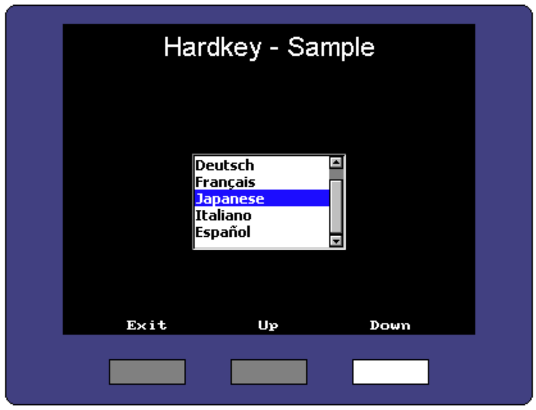
Generated frame view	Custom bitmap view
	

Table 3.3: Generated frame and custom bitmap view

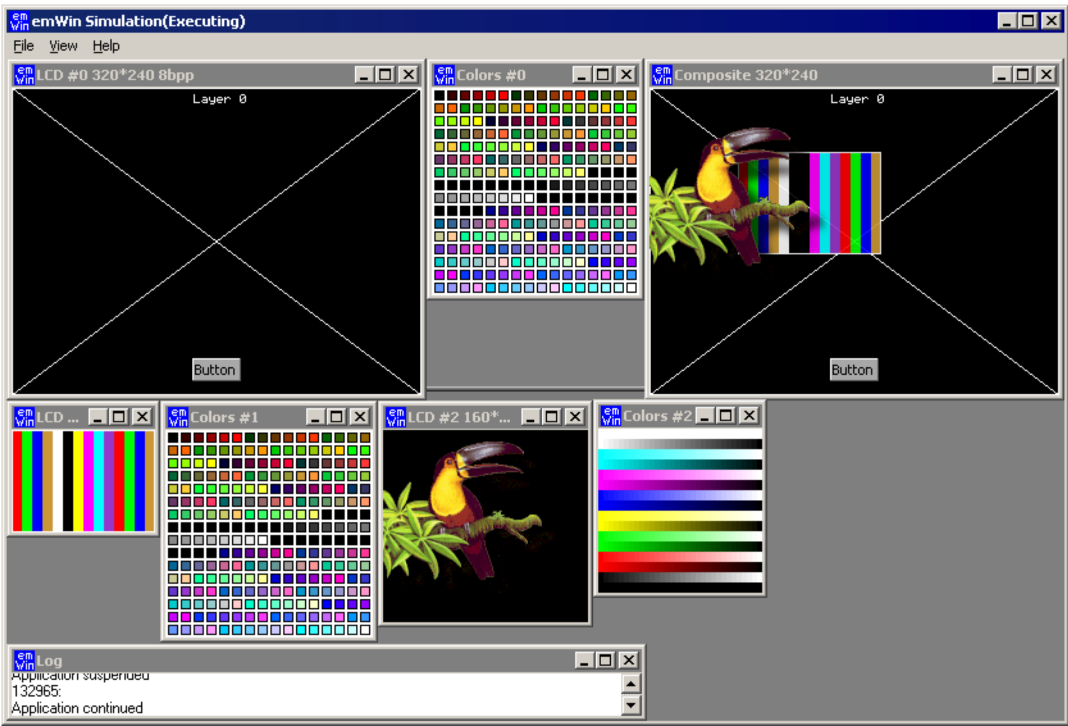
Window view


Table 3.4: Window view

The following will explain in detail how each option can be used.

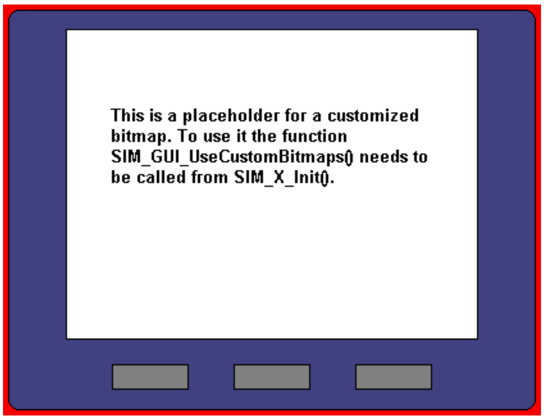
3.2.1 Generated frame view

The simulation shows the display inside an automatically generated frame surrounding the display. The frame contains a small button which per default closes the application. This is the default behavior of the simulation for single layer systems. 'Single layer system' means that only the first layer is initialized.

3.2.2 Custom bitmap view

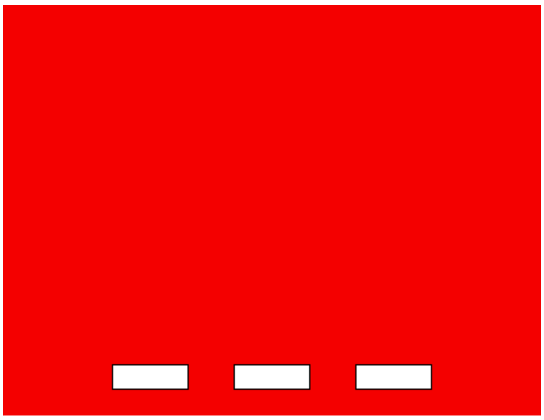
The simulation can show the simulated display in a bitmap of your choice, typically your target device. The bitmap can be used to simulate the behavior of the entire target device. In order to simulate the appearance of the device, bitmaps are required.

Device bitmap



The first bitmap is usually a photo (top view) of the device, and needs to be named Device.bmp. It may be a separate file (in the same directory as the executable), or it may be included as a resource in the application. How to do this is explained later in this chapter.
The file should provide an area for the simulated display of the same size in pixels as the physical display resolution.
If there are any hardkeys to be simulated the bitmap should also show all of them in unpressed state.
Transparent areas need to be colored with exact the same color as defined with the function `SIM_GUI_SetTransColor()`, typically bright red (0xFF0000). These areas do not have to be rectangular; they can have an arbitrary shape (up to a certain complexity which is limited by your operating system, but is normally sufficient). Bright red is the default color for transparent areas, mainly because it is not usually contained in most bitmaps. To use a bitmap with bright red, the default transparency color may be changed with the function `SIM_GUI_SetTransColor()`.

Hardkey bitmap



The second bitmap file is required for defining the hardkeys and must be named `Device1.bmp`. It contains the buttons in pressed state. The non hardkey area has to be filled with the transparent color. This is only a short description. For more details about how to simulate hardkeys, see "Hardkey simulation" on page 65.

Using separate files

When starting the simulation, it checks if the directory of the executable contains the bitmap files `Device.bmp` and `Device1.bmp`. If these files are available, they are used automatically and the resource bitmaps are ignored. Note that this is only valid with single layer systems.

Adding the bitmap to the application resources

The resource file of the simulation can be found under `System\Simulation\Res\Simulation.rc`. It contains the following section:

```
//////////////////////////////////////////////////////////////////////////
```



```
//
// Customizable bitmaps
//
IDB_DEVICE              BITMAP  DISCARDABLE     "Device.bmp"
IDB_DEVICE1             BITMAP  DISCARDABLE     "Device1.bmp"
```

This section can be used to set custom device files. More information can be found in the Win32 documentation.

3.2.3 Window view

Default for simulating a multiple layer system is showing each layer in a separate window without using bitmaps or a generated frames.

3.3 Device simulation API

All of the device simulation API functions should be called in the setup phase. The calls should be done from within the routine `SIM_X_Config()`, which is located in the file `SIMConf.c` in the configuration folder. The example below calls `SIM_SetLCDPos()` in the setup:

```
#include "LCD_SIM.h"

void SIM_X_Config() {
  SIM_GUI_SetLCDPos(50, 20); // Define the position of the LCD in the bitmap}
}
```

The table below lists the available device-simulation-related routines in alphabetical order. Detailed descriptions of the routines follow:

Routine	Description
SIM_GUI_ShowDevice()	Manages the visibility of the device bitmap.
SIM_GUI_SetCallback()	Sets a callback function for receiving the handles of the simulation windows.
SIM_GUI_SetCompositeColor()	Sets the background color of the composite window. (Only used with MultiLayer support)
SIM_GUI_SetCompositeSize()	Sets the size of the composite window. (Only used with MultiLayer support)
SIM_GUI_SetLCDColorBlack()	Set the color to be used as black (color monochrome displays).
SIM_GUI_SetLCDColorWhite()	Set the color to be used as white (color monochrome displays).
SIM_GUI_SetLCDPos()	Set the position for the simulated LCD within the target device bitmap.
SIM_GUI_SetMag()	Set magnification factors for X and/or Y axis.
SIM_GUI_SetTransColor()	Set the color to be used for transparent areas (default: 0xFF0000).
SIM_GUI_SetTransMode()	Sets the transparency mode for the given layer.
SIM_GUI_UseCustomBitmaps()	Tells the simulation to use the custom bitmaps from the application resource file.

Table 3.5: Device simulation API list

SIM_GUI_ShowDevice()

Description

When using multiple layers, this function can be used to show the device bitmap. By default each layer and the composite view are displayed in separate windows.

Prototype

```
void SIM_GUI_ShowDevice(int OnOff);
```

Parameter	Description
OnOff	1 for showing the bitmap, 0 for hiding it.

Table 3.6: SIM_GUI_ShowDevice() parameter list

Additional information

This function has no impact in case only 1 layer is configured.

SIM_GUI_SetCallback()

Description

If it is required to simulate more than the display window or hardkeys, you can set a callback function to receive the window handles of the simulation. This opens up the possibility e.g. to add additional controls outside of the display window like leds or sliders. Please note that the emWin functions can not be used there.

Prototype

```
void SIM_GUI_SetCallback(int (* _pfInfoCallback)(SIM_GUI_INFO * pInfo));
```

Parameter	Description
`_pfInfoCallback`	Pointer to the callback function. The function has to expect a pointer to a SIM_GUI_INFO structure as a parameter

Table 3.7: SIM_GUI_SetCallback() parameter list

Elements of structure SIM_GUI_INFO

Data type	Element	Description
`HWND`	`hWndMain`	Handle to the main window.
`HWND`	`ahWndLCD[16]`	Array of handles to the display layers.
`HWND`	`ahWndColor[16]`	Array of handles to the palette layers.

Table 3.8: SIM_GUI_INFO element list

SIM_GUI_SetCompositeColor()

Description

When simulating a multiple layer system each layer can be shown in its own window. However, the physical display has only one area. It shows the result of the blended layers. The simulation shows the result in the composite window which can have its own size independent of the layers. Each layer can have its own position and its own size within the composite window. This means that not necessarily the complete area is covered by the layers. For this case (and also for transparency effects) this function sets the default background color of the composite window.

Prototype

```
void SIM_GUI_SetCompositeColor(U32 Color);
```

Parameter	Description
`Color`	Background color to be used.

Table 3.9: SIM_GUI_SetCompositeColor() parameter list

Additional information

This function does not have an effect when using SoftLayers.

SIM_GUI_SetCompositeSize()

Description

As described above under `SIM_GUI_SetCompositeColor()` the size of the composite window is independent of the size of the layers. This function is used to set the size of the composite window.

Prototype

```
void SIM_GUI_SetCompositeSize(int xSize, int ySize);
```

Parameter	Description
`xSize`	Horizontal size in pixels.
`ySize`	Vertical size in pixels.

Table 3.10: SIM_GUI_SetCompositeSize() parameter list

Example

The following shows a typical use (for MultiLayer support):

```
void SIM_X_Config() {
  SIM_GUI_SetCompositeSize(240, 320);   // Set size of composite window
  SIM_GUI_SetCompositeColor(0x800000); // Define background color of composite window
}
```


SIM_GUI_SetLCDColorBlack()

SIM_GUI_SetLCDColorWhite()

Description

Set the colors to be used as black or white, respectively, on color monochrome displays.

Prototypes

```
int SIM_GUI_SetLCDColorBlack(int DisplayIndex, int Color);
int SIM_GUI_SetLCDColorWhite(int DisplayIndex, int Color);
```

Parameter	Description
DisplayIndex	Reserved for future use; must be 0.
Color	RGB value of the color.

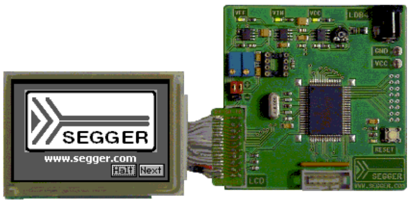
Table 3.11: SIM_GUI_SetLCDColorBlack / SIM_GUI_SetLCDColorWhite() parameter list

Additional information

These functions can be used to simulate the true background color of your display. The default color values are black and white, or 0x000000 and 0xFFFFFF.

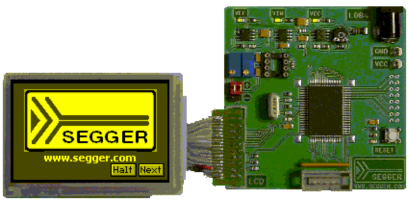
Example using default settings

```
void SIM_X_Config() {
  SIM_GUI_SetLCDPos(14,84);                 // Define the position of the LCD
                                            // in the bitmap
  SIM_GUI_SetLCDColorBlack (0, 0x000000);   // Define the color used as black
  SIM_GUI_SetLCDColorWhite (0, 0xFFFFFF);   // Define the color used as white
  (used for colored monochrome displays)
}
```



Example using yellow instead of white

```
void SIM_X_Config() {
  SIM_GUI_SetLCDPos(14,84);                 // Define the position of the LCD
                                            // in the bitmap
  SIM_GUI_SetLCDColorBlack (0, 0x000000);   // Define the color used as black
  SIM_GUI_SetLCDColorWhite (0, 0x00FFFF);   // Define the color used as white
  (used for colored monochrome displays)
}
```



SIM_GUI_SetLCDPos()

Description

Sets the position for the simulated LCD within the target device bitmap.

Prototype

```
void SIM_GUI_SetLCDPos(int x, int y);
```

Parameter	Description
x	X-position of the upper left corner for the simulated LCD (in pixels).
y	Y-position of the upper left corner for the simulated LCD (in pixels).

Table 3.12: SIM_GUI_SetLCDPos() parameter list

Additional information

The X- and Y-positions are relative to the target device bitmap, therefore position (0,0) refers to the upper left corner (origin) of the bitmap and not your actual LCD. Only the origin of the simulated screen needs to be specified; the resolution of your display should already be reflected in the configuration files in the `Config` directory. The use of this function enables the use of the bitmaps `Device.bmp` and `Device1.bmp`. Note that the values need to be >= 0 for enabling the use of the bitmaps. If the use of the device bitmaps should be disabled, omit the call of this function in `SIM_X_Init()`.

SIM_GUI_SetMag()

Description

Sets magnification factors for X and/or Y axis.

Prototype

```
void SIM_GUI_SetMag(int MagX, int MagY);
```

Parameter	Description
MagX	Magnification factor for X axis.
MagY	Magnification factor for Y axis.

Table 3.13: SIM_GUI_SetMag() parameter list

Additional information

Per default the simulation uses one pixel on the PC for each pixel of the simulated display. The use of this function makes sense for small displays. If using a device bitmap together with a magnification > 1 the device bitmap needs to be adapted to the magnification. The device bitmap is not magnified automatically.

SIM_GUI_SetTransColor()

Description

Sets the color to be used for transparent areas of device or hardkey bitmaps.

Prototype

```
I32 SIM_GUI_SetTransColor(I32 Color);
```

Parameter	Description
Color	RGB value of the color in the format 00000000RRRRRRRRGGGGGGGGBBBBBBBB.

Table 3.14: SIM_GUI_SetTransColor() parameter list

Additional information

The default setting for transparency is bright red (0xFF0000).
You would typically only need to change this setting if your bitmap contains the same shade of red.

SIM_GUI_SetTransMode()

Description

Sets the transparency mode for the given layer.

Prototype

```
void SIM_GUI_SetTransMode(int LayerIndex, int TransMode);
```

Parameter	Description
LayerIndex	Index of the layer for which the transparency mode should be set.
TransMode	Permitted values are listed below.

Table 3.15: SIM_GUI_SetTransMode() parameter list

Permitted values for parameter TransMode	
GUI_TRANSMODE_PIXELALPHA	The alpha value is taken from the pixel data in order to mix colors with the according background.
GUI_TRANSMODE_ZERO	In this mode pixels are fully transparent if the pixel data equals 0.

SIM_GUI_UseCustomBitmaps()

Description

As described earlier in this chapter it is possible to use device bitmaps from the application resources. This function tells the simulation to use the device- and hardkey bitmaps from the application resources and not to generate the default frame bitmap.

Prototype

```
void SIM_GUI_UseCustomBitmaps(void);
```

Additional information

The emWin shipment contains per default 2 bitmaps, `Device.bmp` and `Device1.bmp`, located in `Start\System\Simulation\Res` which can be used as a starting point for your own bitmaps.

3.4 Hardkey simulation

The hardkey simulation can only be used in the custom bitmap view. Hardkeys may also be simulated as part of the device, and may be selected with the mouse pointer. The idea is to be able to distinguish whether a key or button on the simulated device is pressed or unpressed. A hardkey is considered "pressed" as long as the mouse button is held down; releasing the mouse button or moving the pointer off of the hardkey releases the key. A toggle behavior between pressed and unpressed may also be specified with the routine `SIM_HARDKEY_SetMode()`.
In order to simulate hardkeys, you need a second bitmap of the device which is transparent except for the keys themselves (in their pressed state). As described earlier in this chapter, this bitmap can be in a separate file in the directory, or included as a resource in the executable. Hardkeys may be any shape, as long as they are exactly the same size in pixels in both `Device.bmp` and `Device1.bmp`. The following example illustrates this:

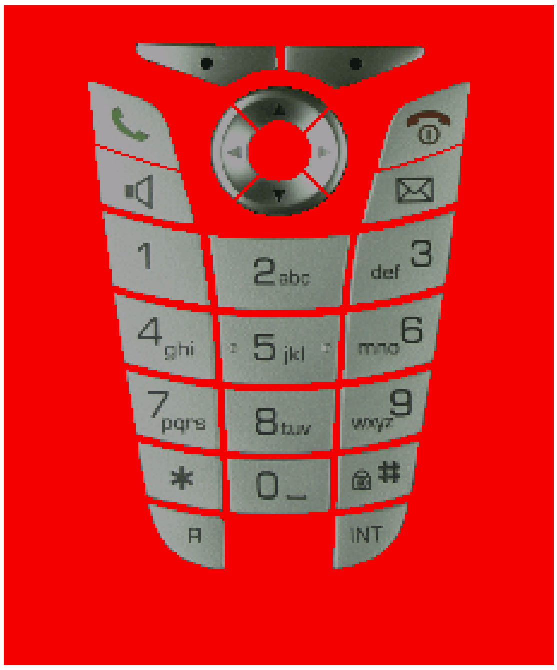
Device bitmap: unpressed hardkey state (Device.bmp)	**Device hardkey bitmap: pressed hardkey state (Device1.bmp)**
	

Table 3.16: Hardkey device bitmaps

When a key is "pressed" with the mouse, the corresponding section of the hardkey bitmap (`Device1.bmp`) will overlay the device bitmap in order to display the key in its pressed state.
The keys may be polled periodically to determine if their states (pressed/unpressed) have changed and whether they need to be updated. Alternatively, a callback routine may be set to trigger a particular action to be carried out when the state of a hardkey changes.

3.4.1 Hardkey simulation API

The hardkey simulation functions are part of the standard simulation program shipped with emWin. If using a user defined emWin simulation these functions may not be available. The table below lists the available hardkey-simulation-related routines in alphabetical order within their respective categories. Detailed descriptions of the routines follow:

Routine	Description
SIM_HARDKEY_GetNum()	Return the number of available hardkeys.
SIM_HARDKEY_GetState()	Return the state of a specified hardkey (0: unpressed, 1: pressed).
SIM_HARDKEY_SetCallback()	Set a callback routine to be executed when the state of a specified hardkey changes.
SIM_HARDKEY_SetMode()	Set the behavior for a specified hardkey (default = 0: no toggle).
SIM_HARDKEY_SetState()	Set the state for a specified hardkey (0: unpressed, 1: pressed).

Table 3.17: Hardkey simulation API list

SIM_HARDKEY_GetNum()

Description

Returns the number of available hardkeys.

Prototype

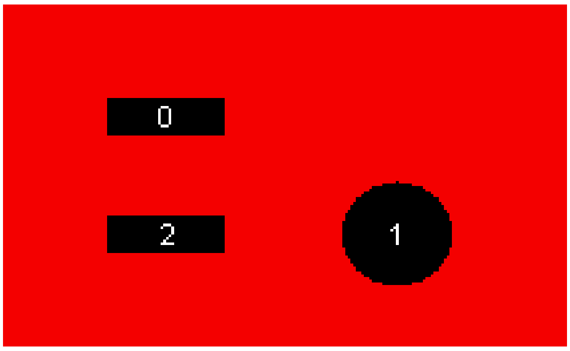
```
int SIM_HARDKEY_GetNum(void);
```

Return value

The number of available hardkeys found in the bitmap.

Additional information

The numbering order for hardkeys is standard reading order (left to right, then top to bottom). The topmost pixel of a hardkey is therefore found first, regardless of its horizontal position. In the bitmap below, for example, the hardkeys are labeled as they would be referenced by the KeyIndex parameter in other functions:



It is recommended to call this function in order to verify that a bitmap is properly loaded.

SIM_HARDKEY_GetState()

Description

Returns the state of a specified hardkey.

Prototype

```
int SIM_HARDKEY_GetState(unsigned int KeyIndex);
```

Parameter	Description
KeyIndex	Index of hardkey (0 = index of first key).

Table 3.18: SIM_HARDKEY_GetState() parameter list

Return value

State of the specified hardkey:
0: unpressed
1: pressed

SIM_HARDKEY_SetCallback()

Description

Sets a callback routine to be executed when the state of a specified hardkey changes.

Prototype

```
SIM_HARDKEY_CB * SIM_HARDKEY_SetCallback(unsigned int     KeyIndex
                                         SIM_HARDKEY_CB * pfCallback);
```

Parameter	Description
KeyIndex	Index of hardkey (0 = index of first key).
pfCallback	Pointer to callback routine.

Table 3.19: SIM_HARDKEY_SetCallback() parameter list

Return value

Pointer to the previous callback routine.

Additional information

Note that multi tasking support has to be enabled if GUI functions need to be called within the callback functions. Without multi tasking support only the GUI functions which are allowed to be called within an interrupt routine should be used.
The callback routine must have the following prototype:

Prototype

```
void SIM_HARDKEY_CB(int KeyIndex, int State);
```

Parameter	Description
KeyIndex	Index of hardkey (0 = index of first key).
State	State of the specified hardkey. See table below.

Table 3.20: SIM_HARDKEY_CB() parameter list

Permitted values for parameter State	
0	Unpressed.
1	Pressed.

SIM_HARDKEY_SetMode()

Description

Sets the behavior for a specified hardkey.

Prototype

```
int SIM_HARDKEY_SetMode(unsigned int KeyIndex, int Mode);
```

Parameter	Description
KeyIndex	Index of hardkey (0 = index of first key).
Mode	Behavior mode. See table below.

Table 3.21: SIM_HARDKEY_SetMode() parameter list

Permitted values for parameter Mode	
0	Normal behavior (default).
1	Toggle behavior.

Additional information

Normal (default) hardkey behavior means that a key is considered pressed only as long as the mouse button is held down on it. When the mouse is released or moved off of the hardkey, the key is considered unpressed.
With toggle behavior, each click of the mouse toggles the state of a hardkey to pressed or unpressed. That means if you click the mouse on a hardkey and it becomes pressed, it will remain pressed until you click the mouse on it again.

SIM_HARDKEY_SetState()

Description

Sets the state for a specified hardkey.

Prototype

```
int SIM_HARDKEY_SetState(unsigned int KeyIndex, int State);
```

Parameter	Description
KeyIndex	Index of hardkey (0 = index of first key).
State	State of the specified hardkey. See table below.

Table 3.22: SIM_HARDKEY_SetState() parameter list

Permitted values for parameter State	
0	Unpressed.
1	Pressed.

Additional information

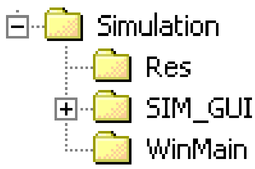
This function is only usable when `SIM_HARDKEY_SetMode()` is set to 1 (toggle mode).

3.5 Integrating the emWin simulation into an existing simulation

In order to integrate the emWin simulation into an existing simulation, the source code of the simulation is not required. The source code of the simulation is not normally shipped with emWin. It is a separate (optional) software item and is not included in the emWin basic package.
Normally the source code of the emWin simulation is not needed but available as an optional software item. As described earlier in this chapter the basic package and the trial version contains a simulation library. The API functions of this library can be used if for example the emWin simulation should be added to an existing hardware or real time kernel (RTOS) simulation.
To add the emWin simulation to an existing simulation (written in C or C++, using the Win32 API), only a few lines of code need to be added.

3.5.1 Directory structure

The subfolder Simulation of the System folder contains the emWin simulation. The directory structure is shown on the right. The table below explains the contents of the subfolders:



Directory	Content
`Simulation`	Simulation source and header files to be used with and without the simulation source code. The folder also contains a ready to use simulation library.
`Res`	Resource files.
`SIM_GUI`	GUI simulation source code (optional).
`WinMain`	Contains the WinMain routine.

Table 3.23: Content of the subfolder Simulation

3.5.2 Using the simulation library

The following steps will show how to use the simulation library to integrate the emWin simulation into an existing simulation:

- Step 1: Add the simulation library `GUISim.lib` to the project.
- Step 2: Add all GUI files to the project as described in the chapter 2.1.1, "Subdirectories".
- Step 3: Add the include directories to the project as described in the chapter 2.1.2, "Include Directories".
- Step 4: Modify WinMain.

3.5.2.1 Modifying WinMain

Every windows Win32 program starts with `WinMain()` (contrary to a normal C program from the command line, which starts with `main()`. All that needs to be done is to add a few lines of code to this routine.
The following function calls need to be added (normally in the order as it’s shown in the following application code example):

- `SIM_GUI_Enable`
- `SIM_GUI_Init`
- `SIM_GUI_CreateLCDWindow`
- `CreateThread`
- `SIM_GUI_Exit`

3.5.2.2 Example application

The following application is available under `Sample\WinMain\SampleApp.c` and shows how to integrate the emWin simulation into an existing application:

```
#include <windows.h>
#include "GUI_SIM_Win32.h"

void MainTask(void);

/*********************************************************************
*
*       _Thread
*/
static DWORD __stdcall _Thread(void * Parameter) {
  MainTask();
  return 0;
}

/*********************************************************************
*
*       _WndProcMain
*/
static LRESULT CALLBACK _WndProcMain(HWND   hWnd,   UINT   message,
                                     WPARAM wParam, LPARAM lParam) {
  SIM_GUI_HandleKeyEvents(message, wParam);
  switch (message) {
  case WM_DESTROY:
    PostQuitMessage(0);
    break;
  }
  return DefWindowProc(hWnd, message, wParam, lParam);
}

/*********************************************************************
*
*       _RegisterClass
*/
static void _RegisterClass(HINSTANCE hInstance) {
  WNDCLASSEX wcex;

  memset(&wcex, 0, sizeof(wcex));
  wcex.cbSize        = sizeof(WNDCLASSEX);
  wcex.hInstance     = hInstance;
  wcex.style         = CS_HREDRAW | CS_VREDRAW;
  wcex.lpfnWndProc   = (WNDPROC)_WndProcMain;
  wcex.hIcon         = 0;
  wcex.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wcex.hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE + 1);
  wcex.lpszMenuName  = 0;
  wcex.lpszClassName = "GUIApplication";
  RegisterClassEx(&wcex);
}

/*********************************************************************
*
*       WinMain
*/
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine, int       nCmdShow) {
  DWORD ThreadID;
  MSG   Msg;
  HWND  hWndMain;
  //
  // Register window class
  //
  _RegisterClass(hInstance);
  //
  // Make sure the driver configuration is done
  //
  SIM_GUI_Enable();
  //
  // Create main window
  //
  hWndMain = CreateWindow("GUIApplication", "Application window",
                          WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_VISIBLE,
                          0, 0, 328, 267, NULL, NULL, hInstance, NULL);
  //
  // Initialize the emWin simulation and create an LCD window
  //
  SIM_GUI_Init(hInstance, hWndMain, lpCmdLine, "embOS - emWin Simulation");
```

```
  SIM_GUI_CreateLCDWindow(hWndMain, 0, 0, 320, 240, 0);
  //
  // Create a thread which executes the code to be simulated
  //
  CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)_Thread, NULL, 0, &ThreadID);
  //
  // Main message loop
  //
  while (GetMessage(&Msg, NULL, 0, 0)) {
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
  }
  SIM_GUI_Exit();
}
```

3.5.3 Integration into the embOS Simulation

3.5.3.1 WinMain

The following code example shows how to modify the existing WinMain of the embOS simulation in order to integrate the emWin simulation. The red colored lines should be added to WinMain to initialize the emWin simulation, to create a simulation window and to exit the emWin simulation:

```
...
#include "GUI_SIM_Win32.h"
...
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine, int       nCmdShow) {
  MSG    Msg;
  HACCEL hAccelTable;
  HWND   hWndMain;
  BITMAP BmpDevice;
  DWORD  ThreadID;

  //
  // Init global data
  //
  _StopHyperThreading();
  _hInst = hInstance;
  //
  // Register main window class
  //
  _RegisterClass();
  //
  // Load bitmap
  //
  _hBmpDevice = (HBITMAP)LoadImage(_hInst,       (LPCTSTR) IDB_DEVICE,
                                   IMAGE_BITMAP, 0, 0, 0);
  _hMenuPopup = LoadMenu(_hInst, (LPCSTR)IDC_CONTEXTMENU);
  _hMenuPopup = GetSubMenu(_hMenuPopup, 0);
  //
  // Make sure the driver configuration is done
  //
  SIM_GUI_Enable();
  //
  // Create main window
  //
  GetObject(_hBmpDevice, sizeof(BmpDevice), &BmpDevice);
  hWndMain = CreateWindowEx(WS_EX_TOPMOST, _sWindowClass, "embOS Simulation",
                            WS_SYSMENU | WS_CLIPCHILDREN | WS_POPUP | WS_VISIBLE,
                            10,   20, BmpDevice.bmWidth, BmpDevice.bmHeight,
                            NULL, NULL, _hInst, NULL);
  if (!hWndMain) {
    return 1;   // Error
  }
  //
  // Init emWin simulation and create window
  //
  SIM_GUI_Init(hInstance, hWndMain, lpCmdLine, "embOS - emWin Simulation");
  SIM_GUI_CreateLCDWindow(hWndMain, 80, 50, 128, 64, 0);
  //
  // Show main window
  //
  ShowWindow(hWndMain, nCmdShow);
  //
  // Load accelerator table
  //
```



```
  hAccelTable = LoadAccelerators(_hInst, (LPCTSTR)IDC_WINMAIN);
  //
  // Application initialization
  //
  CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Thread, NULL, 0, &ThreadID);
  //
  // Main message loop
  //
  if (SIM_Init(hWndMain) == 0) {
    while (GetMessage(&Msg, NULL, 0, 0)) {
      if (!TranslateAccelerator(Msg.hwnd, hAccelTable, &Msg)) {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
      }
    }
  }
  //
  // Exit emWin simulation
  //
  SIM_GUI_Exit();
  return 0;
}
```

3.5.3.2 Target program (main)

The emWin API can be called from one or more target threads. Without RTOS, the WIN32 API function `CreateThread` is normally used to create a target thread which calls the emWin API; within an RTOS simulation, a target task/thread (Created by the simulated RTOS) is used to call the emWin API. In other words: Use `OS_CreateTask` to create a task for the user interface. Below a modified embOS start application:

```
#include <windows.h>
#include "RTOS.H"
#include "HW_LED.h"
#include "GUI.h"

OS_STACKPTR int Stack0[128], Stack1[128], Stack2[2000]; // Task stacks
OS_TASK         TCB0,        TCB1,        TCB2;         // Task-control-blocks

void Task0(void) {
  while (1) {
    HW_LED_Toggle0();
    OS_Delay(100);
  }
}

void Task1(void) {
  while (1) {
    HW_LED_Toggle1();
    OS_Delay(500);
  }
}

void MainTask(void) {
  int i;

  GUI_COLOR aColor[] = {GUI_RED, GUI_YELLOW};
  GUI_Init();
  while (1) {
    for (i = 0; i < 2; i++) {
      GUI_Clear();
      GUI_SetColor(aColor[i]);
      GUI_SetFont(&GUI_FontComic24B_ASCII);
      GUI_DispStringAt("Hello world!", 1, 1);
      OS_Delay(200);
    }
  }
}

/*********************************************************
*
*       main
*/
void main(void) {
  OS_IncDI();             // Initially disable interrupts
  OS_InitKern();          // Initialize OS
  OS_InitHW();            // Initialize Hardware for OS
  //
  // At least one task here needs to be created here
```

```
    //
    OS_CREATETASK(&TCB0, "HP Task",  Task0,    100, Stack0);
    OS_CREATETASK(&TCB1, "LP Task",  Task1,     50, Stack1);
    OS_CREATETASK(&TCB2, "GUI Task", MainTask,  80, Stack2);
    OS_Start();            // Start multitasking
}
```

The following table shows the simulation before and after integrating the emWin simulation:

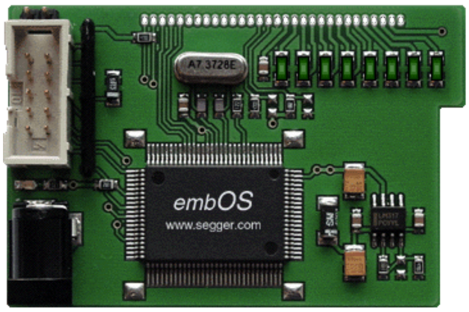
Before	After
	

Table 3.24: Integrated emWin simulation

3.5.4 GUI simulation API

The table below lists the available routines for user defined simulation programs in alphabetical order within their respective categories. The functions are only available with the source code of the emWin simulation. Detailed descriptions of the routines follow:

Routine	Description
`SIM_GUI_CreateLCDInfoWindow()`	Creates a window which shows the available colors of the given layer with the given size and position.
`SIM_GUI_CreateLCDWindow()`	Creates a LCD window with the given size and position.
`SIM_GUI_Enable()`	Executes memory and driver configuration.
`SIM_GUI_Exit()`	Stops the GUI simulation.
`SIM_GUI_Init()`	Initializes the GUI simulation.
`SIM_GUI_SetLCDWindowHook()`	Sets a hook function to be called if the LCD window receives a message.

Table 3.25: GUI simulation API list

SIM_GUI_CreateLCDInfoWindow()

Description

Creates a window which shows the available colors for the given layer.

Prototype

```
HWND SIM_GUI_CreateLCDInfoWindow(HWND hParent, int x, int y, int xSize,
                                 int ySize, int LayerIndex);
```

Parameter	Description
`hParent`	Handle of the parent window.
`x`	X position in parent coordinates.
`y`	Y position in parent coordinates.
`xSize`	X size in pixel of the new window. Should be 160 if using a color depth between 1 and 8 or 128 if working in high color mode.
`ySize`	Y size in pixel of the new window. Should be 160 if using a color depth between 1 and 8 or 128 if working in high color mode.
`LayerIndex`	Index of the layer to be shown.

Table 3.26: SIM_GUI_CreateLCDInfoWindow() parameter list

Additional information

The created color window has no frame, no title bar and no buttons.

Example

```
SIM_GUI_CreateLCDInfoWindow(hWnd, 0, 0, 160, 160, 0);
```

Screenshot

SIM_GUI_CreateLCDWindow()

Description

Creates a window which simulates a LCD display with the given size at the given position.

Prototype

```
HWND SIM_GUI_CreateLCDWindow(HWND hParent, int x, int y,
                                            int xSize, int ySize int LayerIndex);
```

Parameter	Description
hParent	Handle of the parent window.
x	X position in parent coordinates.
y	Y position in parent coordinates.
xSize	X size in pixel of the new window.
ySize	Y size in pixel of the new window.
LayerIndex	Index of layer to be shown.

Table 3.27: SIM_GUI_CreateLCDWindow() parameter list

Additional information

All display output to the given layer will be shown in this window. The size of the window should be the same as configured in `LCDConf.c`.
The created simulation window has no frame, no title bar and no buttons.

SIM_GUI_Enable()

Description

The function needs to be called at the beginning of the application to make sure that memory and driver will be configured at first.

Prototype

```
void SIM_GUI_Enable(void);
```

SIM_GUI_Exit()

Description

The function should be called before the simulation returns to the calling process.

Prototype

```
void SIM_GUI_Exit(void);
```


SIM_GUI_Init()

Description

This function initializes the emWin simulation and should be called before any other SIM_GUI... function call.

Prototype

```
int SIM_GUI_Init(HINSTANCE hInst, HWND hWndMain, char * pCmdLine,
                 const char * sAppName);
```

Parameter	Description
hInst	Handle to current instance passed to WinMain.
hWndMain	Handle of the simulations main window.
pCmdLine	Pointer to command line passed to WinMain
sAppName	Pointer to a string that contains the application name.

Table 3.28: SIM_GUI_Init() parameter list

Additional information

The parameters hWndMain and sAppName are used in case a message box should be displayed.

SIM_GUI_SetLCDWindowHook()

Description

Sets a hook function to be called from the simulation if the LCD window receives a message.

Prototype

```
void SIM_GUI_SetLCDWindowHook(SIM_GUI_tfHook * pfHook);
```

Parameter	Description
pfHook	Pointer to hook function.

Table 3.29: SIM_GUI_SetLCDWindowHook() parameter list

Prototype of hook function

```
int Hook(HWND   hWnd,     UINT Message, WPARAM wParam, LPARAM lParam,
         int  * pResult);
```

Parameter	Description
hWnd	Handle of LCD window.
Message	Message received from the operating system.
wParam	wParam message parameter passed by the system.
lParam	lParam message parameter passed by the system.
pResult	Pointer to an integer which should be used as return code if the message has been processed by the hook function.

Table 3.30: Hook function parameter list

Return value

The hook function should return 0 if the message has been processed. In this case the GUI simulation ignores the message.

Chapter 4

Viewer

If you use the simulation when debugging your application, you cannot see the display output when stepping through the source code. The primary purpose of the viewer is to solve this problem. It shows the contents of the simulated display(s) while debugging in the simulation.

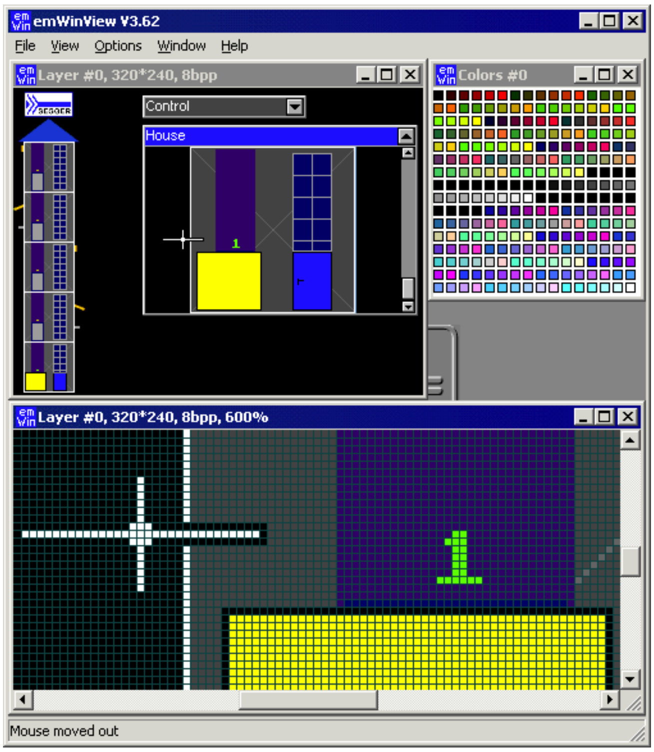
The viewer gives you the following additional capabilities:

- Multiple windows for each layer
- Watching the whole virtual layer in one window
- Magnification of each layer window
- Composite view if using multiple layers

4.1 Using the viewer

The viewer allows you to:

- Open multiple windows for any layer/display
- Zoom in on any area of a layer/display
- See the contents of the individual layers/displays as well as the composite view in multi-layer configurations
- See the contents of the virtual screen and the visible display when using the virtual screen support.



The screenshot shows the viewer displaying the output of a single layer configuration. The upper left corner shows the simulated display. In the upper right corner is a window, which shows the available colors of the display configuration. At the bottom of the viewer a second display window shows a magnified area of the simulated display. If you start to debug your application, the viewer shows one display window per layer and one color window per layer. In a MultiLayer configuration, a composite view window will also be visible.

4.1.1 Using the simulation and the viewer

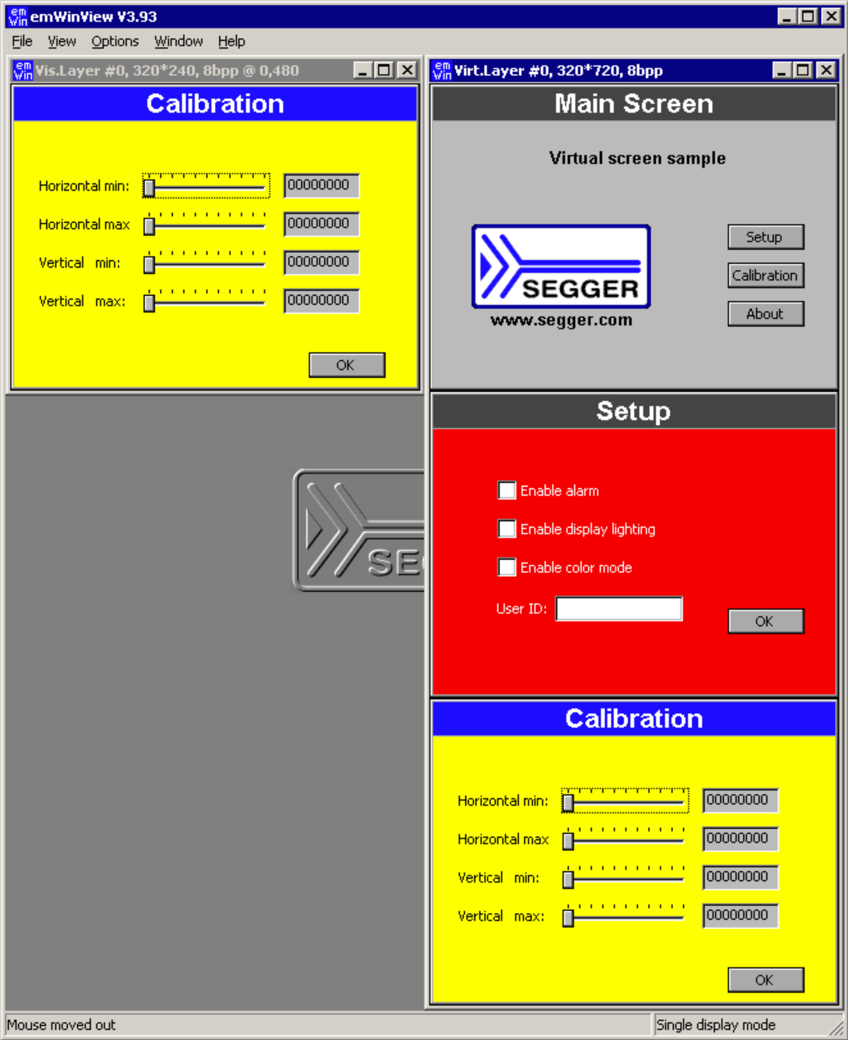
If you use the simulation when debugging your application, you cannot see the display output when stepping through the source code. This is due to a limitation of Win32: If one thread (the one being debugged) is halted, all other threads of the process are also halted. This includes the thread which outputs the simulated display on the screen.
The emWin viewer solves this problem by showing the display window and the color window of your simulation in a separate process. It is your choice if you want to start the viewer before debugging your application or while you are debugging. Our suggestion:

- Step 1: Start the viewer. No display- or color window is shown until the simulation has been started.
- Step 2: Open the Visual C++ workspace.
- Step 3: Compile and run the application program.
- Step 4: Debug the application as described previously.

The advantage is that you can now follow all drawing operations step by step in the LCD window.

4.1.2 Using the viewer with virtual pages

By default the viewer opens one window per layer which shows the visible part of the video RAM, normally the display. If the configured virtual video RAM is larger than the display, the command `View/Virtual Layer/Layer (0...4)` can be used to show the whole video RAM in one window. When using the function `GUI_SetOrg()`, the contents of the visible screen will change, but the virtual layer window remains unchanged:



For more information about virtual screens, refer to chapter "Virtual screens / Virtual pages" on page 945.

4.1.3 Always on top

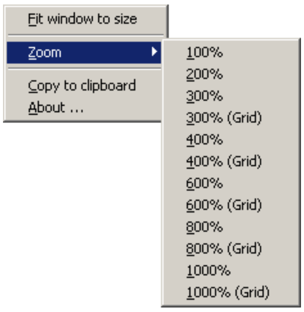
Per default the viewer window is always on top. You can change this behavior by selecting `Options\Always on top` from the menu.

4.1.4 Open further windows of the display output

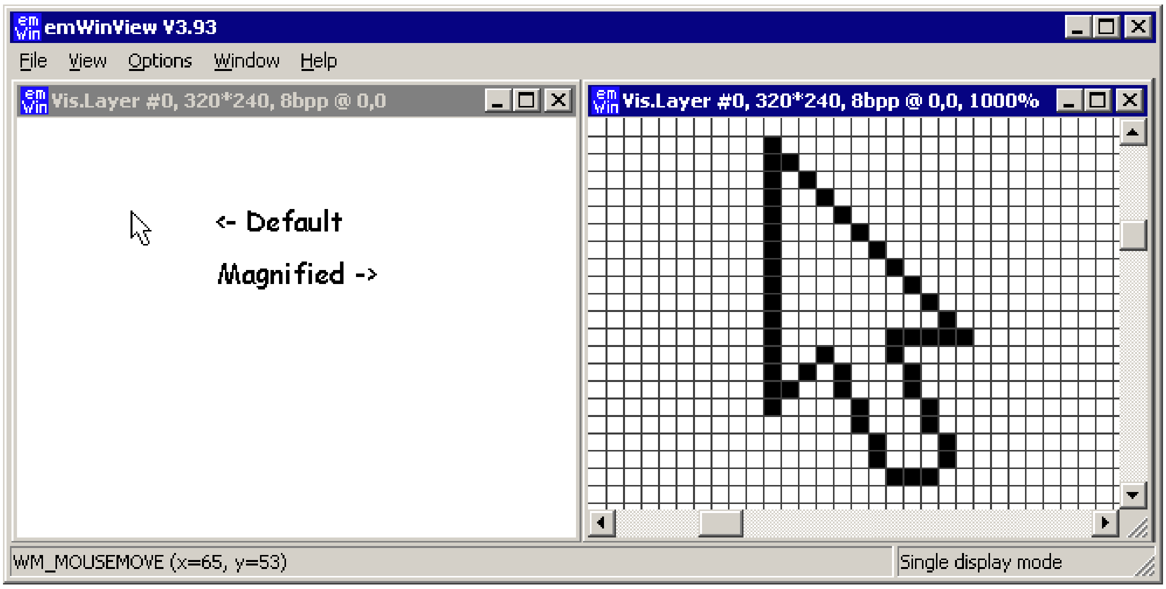
If you want to show a magnified area of the LCD output or the composite view of a MultiLayer configuration it could be useful to open more than one output window. You can do this by `View/Visible Layer/Layer (1...4)`, `View/Virtual Layer/Layer (1...4)` or `View/Composite`.

4.1.5 Zooming

Zooming in or out is easy:
Right-click on a layer or composite window opens the `Zoom` popup menu. Choose one of the zoom options:



Using the grid



If you magnify the LCD output >= 300%, you have the choice between showing the output with or without a grid. It is possible to change the color of the grid. This can be done choosing the Menu point `Options/Grid color`.

Adapting the size of the window

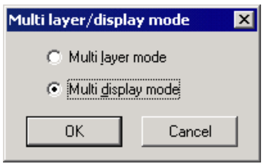
If you want to adapt the size of the window to the magnification choose `Fit window to size` from the first popup menu.

4.1.6 Copy the output to the clipboard

Click onto a LCD window or a composite view with the right mouse key and choose `Copy to clipboard`. Now you can paste the contents of the clipboard for example into the `MS Paint` application.

4.1.7 Using the viewer with multiple displays

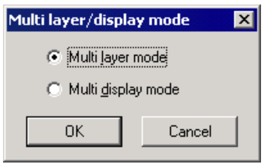
If you are working with multiple displays you should set the viewer into 'Multi display mode' by using the command `Options/Multi layer/display`.



When starting the debugger the viewer will open one display window and one color window for each display:

4.1.8 Using the viewer with multiple layers

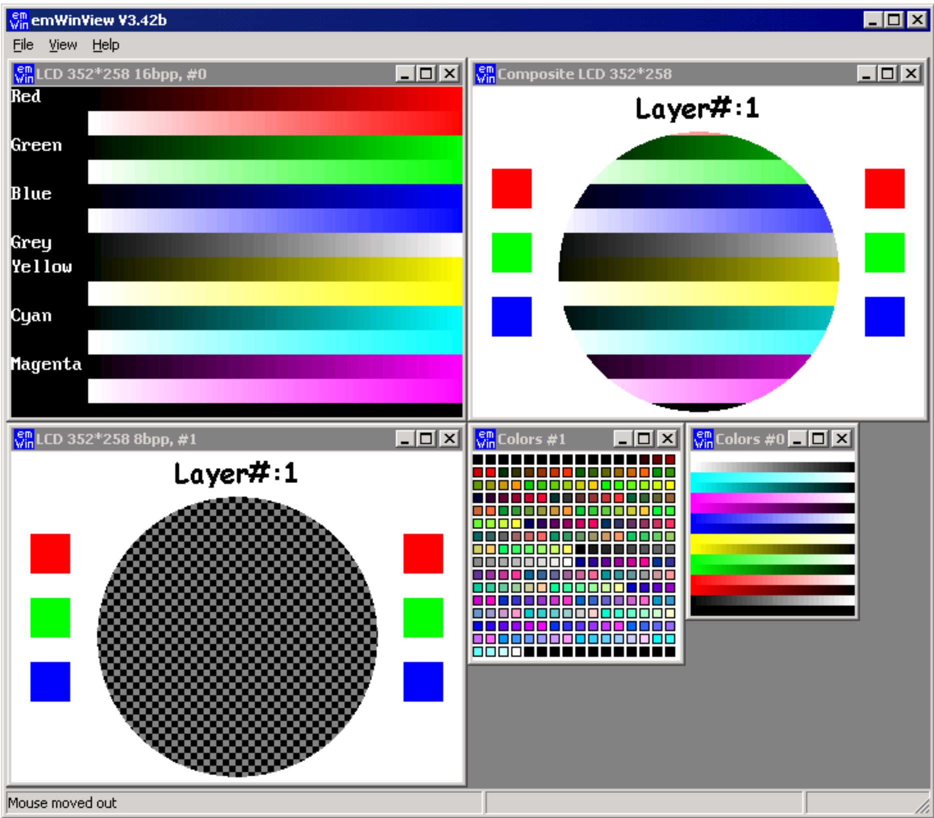
If you are working with multiple layers you should set the viewer into 'Multi layer mode' by using the command `Options/Multi layer/display`.



When starting the debugger the viewer will open one LCD window and one color window for each layer and one composite window for the result.

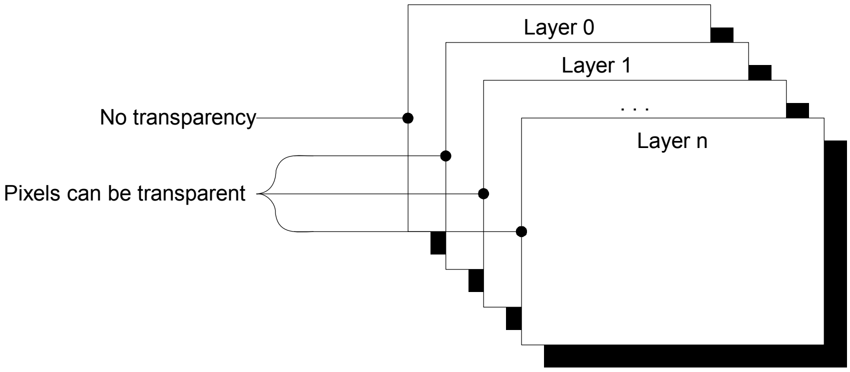
Example

The example below shows a screenshot of the viewer with 2 layers. Layer 0 shows color bars with a high color configuration. Layer 1 shows a transparent circle on a white background with colored rectangles. The composite window shows the result which is actually visible on the display



Transparency

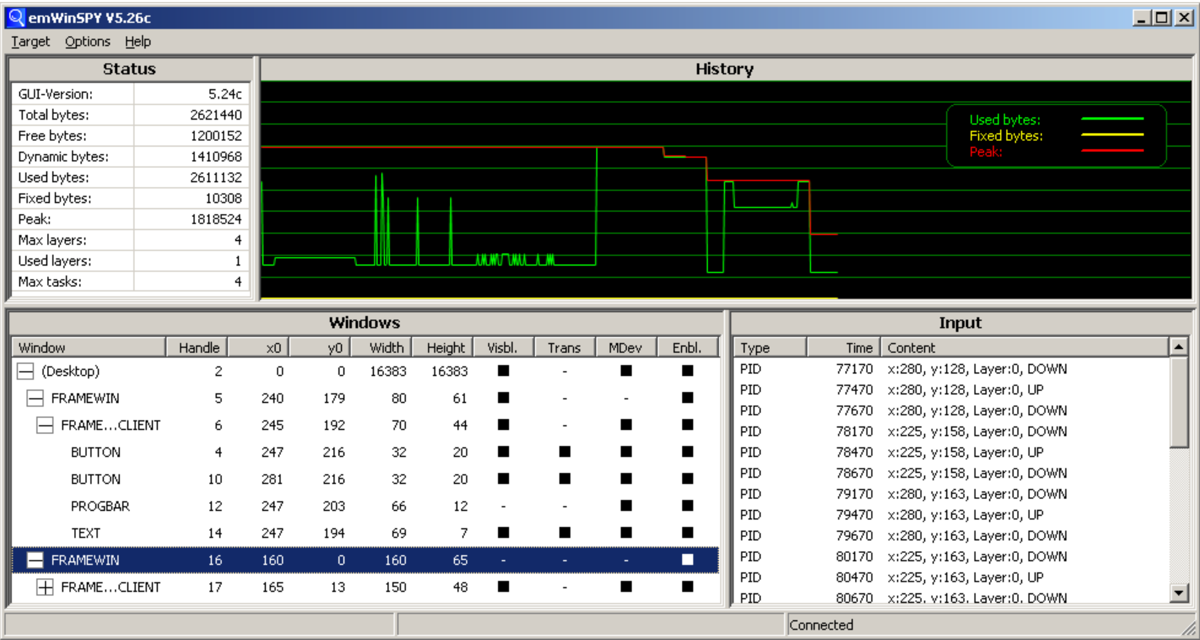
The composite window of the viewer shows all layers; layers with higher index are on top of layers with lower index and can have transparent pixels:



Chapter 5

emWinSPY

emWinSPY is designed for showing runtime information of the embedded target on a PC. It shows information about the currently connected emWin application like memory status, a tree with detailed information about all currently existing windows and a list of user input, which optionally can be written into a log file. Further it is able to take screenshots of the current screen. Communication works via TCP/IP.



5.1 Introduction

The emWinSPY consists of two types of components: A server on application side, which is responsible for supplying data, and the emWinSPY viewer, which requests and shows that data. Server and viewer may be on different machines and on different architectures.

5.1.1 Requirements

TCP/IP stack

Since the communication between server and viewer is based on a TCP/IP connection, emWinSPY requires a TCP/IP stack. In the Win32 simulation environment, TCP/IP (Winsock) is normally present. In the target, a TCP/IP stack needs to be present. The TCP/IP stack is NOT part of emWin. The flexible interface ensures that any TCP/IP stack can be used.

Multi tasking

The emWinSPY server needs to run as a separate thread. Therefore a multi tasking system is required.

Compile-time configuration

Support for emWinSPY needs to be enabled using the following definition in GUIConf.h:

```
#define GUI_SUPPORT_SPY 1
```

5.1.2 Availability

Currently emWinSPY (server and viewer) is part of the basic package. It is also available in the emWin simulation and trial version.

5.2 Starting the emWinSPY server...

5.2.1 ...in the simulation environment

The only thing to be done here is calling `GUI_SPY_StartServer()`. In the simulation environment that function automatically starts an emWinSPY server thread which waits on port 2468 for an incoming connection.

5.2.2 ...on the target hardware

Starting the server on the target hardware works exactly as in the simulation by calling `GUI_SPY_StartServer()`. But whereas the simulation already contains a routine for starting the server that routine needs to be supplied by the application on hardware side. The routine to be added is `GUI_SPY_X_StartServer()`.

5.2.3 GUI_SPY_X_StartServer

It should create a thread which listens on port 2468 until an incoming connection is detected and then calls GUI_SPY_Process(), which is the implementation of the actual server.

Ports

The emWinSPY server should listen on port 2468.

Prototype

```
int GUI_SPY_X_StartServer(void)
```

Example

A sample implementation of that routine can be found in the sample folder:

Sample\GUI_X\GUI_SPY_X_StartServer.c

That sample contains a ready to use implementation for embOS/IP. It is easily adaptable to any IP-stack and any RTOS.

5.3 The emWinSPY viewer

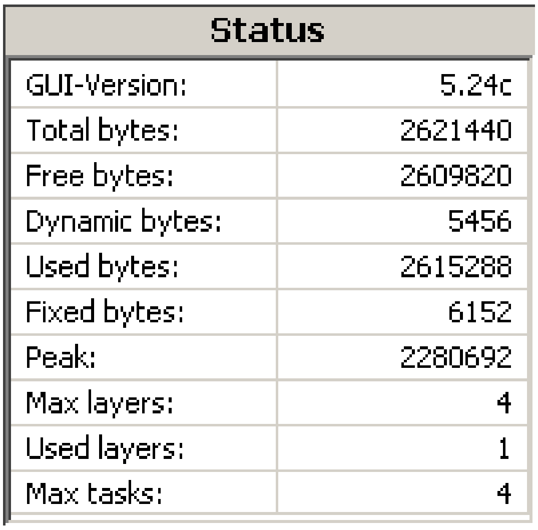
5.3.1 The screen

The screen of the viewer is divided into 4 areas:

Area	Description
Status	Major purpose is showing the current state of memory allocation.
History	History of memory allocation.
Windows	Shows a tree of all existing windows.
Input	List of user interface input: Keyboard, Touch and MTouch

Table 5.1: Screen areas of emWinSPY

5.3.1.1 Status area



The table below shows the information of the status area:

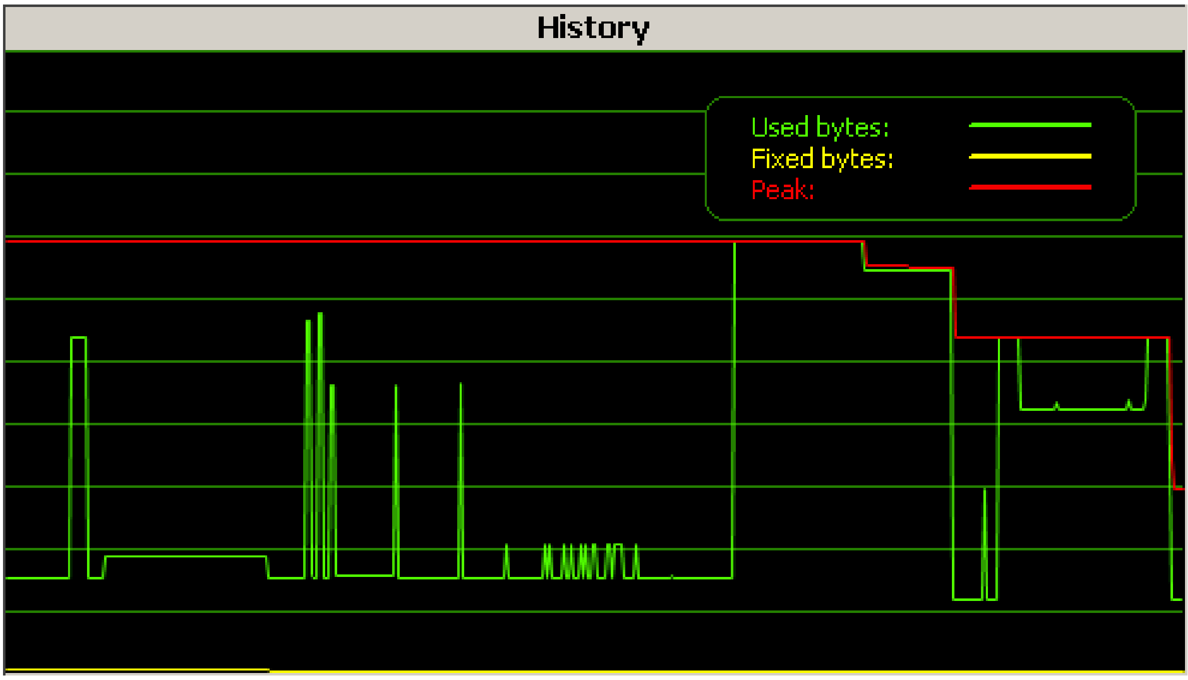
Item	Description
GUI-Version	Used GUI-Version of emWin application
Total bytes	Total bytes of memory available for emWin
Free bytes	Remaining free bytes
Dynamic bytes	Number of bytes currently allocated dynamically
Fixed bytes	Number of bytes used by fixed allocation
Peak	Maximum number of bytes used (fixed + dynamic)
Max layers	Maximum number of layers configured by `GUI_NUM_LAYERS`
Used layers	Number of layers used with current configuration.
Max tasks	Maximum number of GUI-tasks

Table 5.2: Items shown in status area of emWinSPY

Fixed bytes / Dynamic bytes

The memory management of emWin uses 2 kinds of memory allocation: dynamic allocated memory and fixed blocks. Dynamically allocated memory can be freed and reused for further dynamic allocation operations. A fixed memory block is no longer available for dynamic memory allocation. Once a fixed memory block is allocated, that block could not be used for dynamic allocation. Examples for fixed memory blocks are driver caches or conversion buffers.

5.3.1.2 History area



It shows the changes of used bytes, fixed bytes and memory peak in the curse of time. The history remains after disconnecting and reconnecting. A context menu available with a right click allows clearing the history.

5.3.1.3 Windows area

Windows

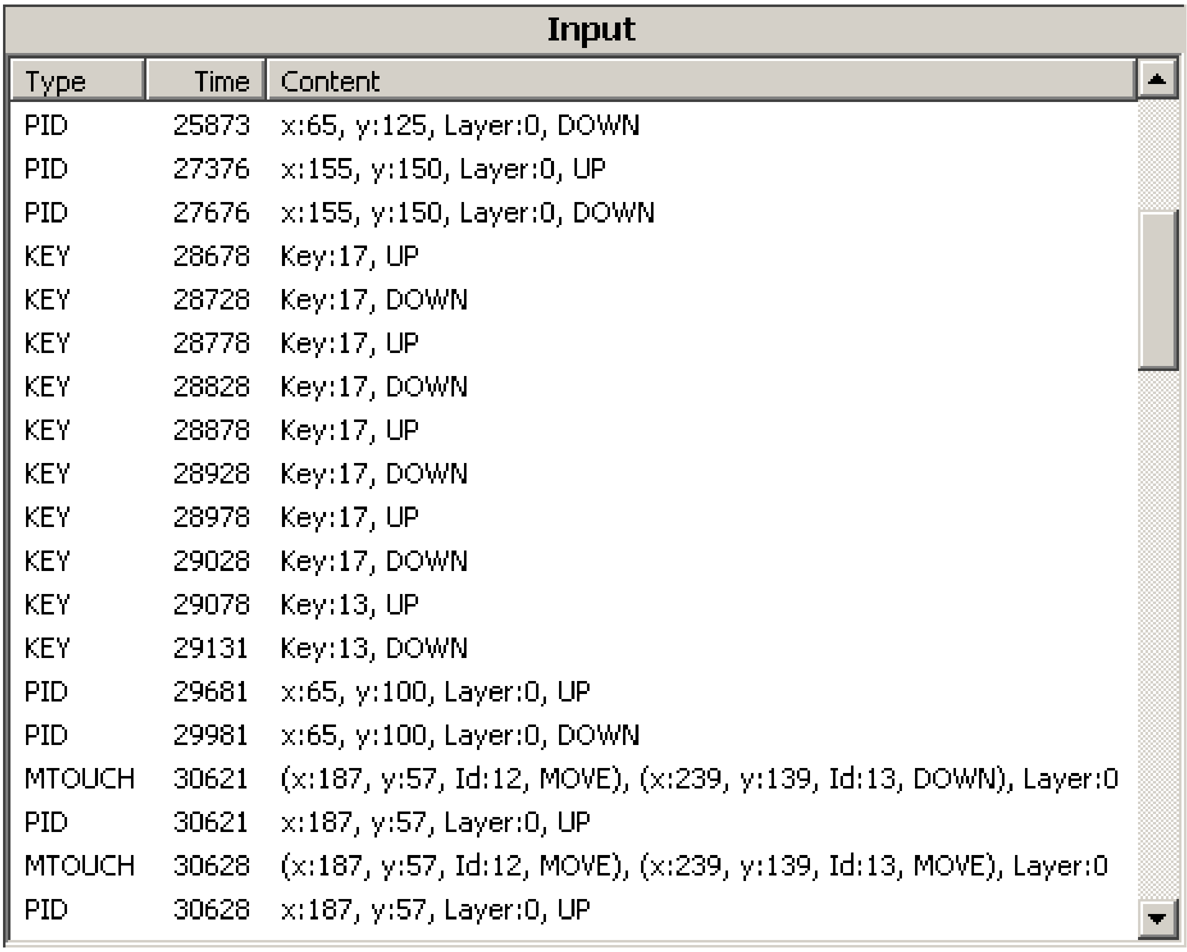
Window	Handle	x0	y0	Width	Height	Visbl.	Trans	MDev	Enbl.
(Desktop)	2	0	0	16383	16383	■	-	■	■
FRAMEWIN	5	240	179	80	61	■	-	-	■
FRAMEWIN-CLIENT	6	245	192	70	44	■	-	■	■
BUTTON	4	247	216	32	20	■	■	■	■
BUTTON	10	281	216	32	20	■	■	■	■
PROGBAR	12	247	203	66	12	■	-	■	■
TEXT	14	247	194	69	7	■	■	■	■
FRAMEWIN	16	160	0	160	65	-	-	-	■
FRAMEWIN-CLIENT	17	165	13	150	48	■	-	■	■
TEXT	19	168	16	150	48	■	■	■	■

That screen contains a tree of all currently existing windows with some additional information about their current states. The following table shows the available information:

Column	Description
Window	Type of window.
Handle	The handle of the window.
x0/y0	Position of the window in screen coordinates.
Width/Height	Size of the window.
Visbl.	Shows if the window (and its children) is visible or not.
Trans	Transparency flag of the window.
MDev	Shows if automatic use of memory devices is enabled for that window.
Enbl.	Shows if the window is enabled or disabled.

Table 5.3: Items shown in windows area of emWinSPY

5.3.1.4 Input area



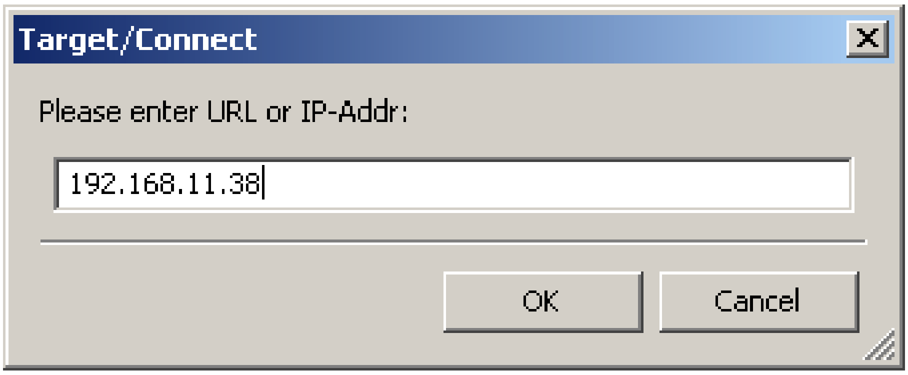
That window shows the user interface input recognized by emWinSPY. The following table shows the available information:

Column	Description
Type	PID: Pointer interface input KEY: Keyboard input MTOUCH: MultiTouch input
Time	Timestamp created on target side
Content	PID: X- and Y-position, Layer, UP or DOWN KEY: Keycode and UP or DOWN MTOUCH: X- and Y-position, TP-Id and DOWN, MOVE or UP for each TP (TP: Touchpoint)

Table 5.4: Items shown in input area of emWinSPY

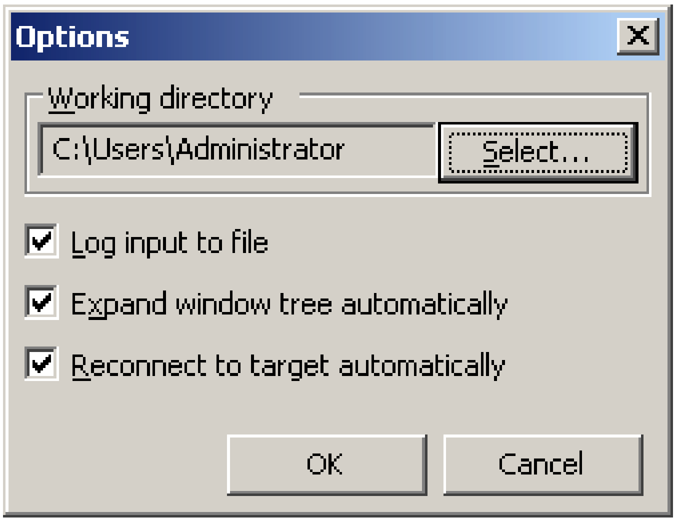
5.3.2 Connecting to target

Once a server is available the command 'Target\Connect' could be used to connect:



An URL or an IP-address could be used here.

5.3.3 Configuration options



Working directory

That folder is used for LOG-files and screenshots. The default value is the home directory of the current user.

Logging

If logging is activated (default), all user interface input is written into a file. The filename is created automatically by using the current local time of the PC. The format is YYYY_MM_DD_HH_MM_SS_MSEC.log. For example 2014_12_16_15_04_44_0943.log means the file was created at 12/2014 at 16:15 and 4 seconds. The last 4 digits contain the milliseconds. Each time a connection is closed (or aborted) the file is be closed and a new one is created once the connection is established again.

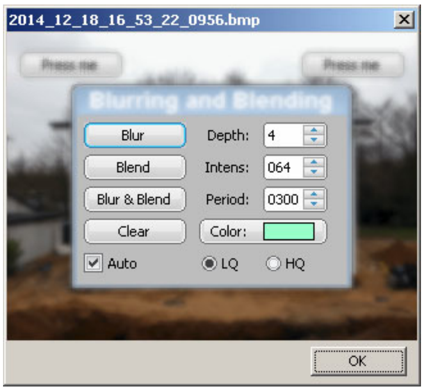
Auto-Expand

If that option is active the nodes of the windows tree are automatically expanded.

Auto-Connect

If activated emWinSPY automatically tries to reconnect to a target after the connection has been closed.

5.3.4 Taking a screenshot from the target



With the command 'Target\Get screenshot' or by pressing <CTRL>+<G> BMP-file containing the current content of the screen is created. The name of the file is created automatically by the same way as described under 'Logging'. Only the file extension is 'bmp' instead of 'log'. The screenshot can be found in the working directory.

5.4 emWinSPY API

The following table lists the emWinSPY related API functions.

Routine	Description
GUI_SPY_Process()	Actual emWinSPY-server to be called from the server thread.
GUI_SPY_SetMemHandler()	Could be used to set a different memory manager for the server thread.
GUI_SPY_StartServer()	Starts the server thread by calling GUI_SPY_X_StartServer().
GUI_SPY_X_StartServer()	Responsible for creating the actual server thread, establishing connections and calling GUI_SPY_Process() from the server thread.

Table 5.5: emWinSPY API list

GUI_SPY_Process()

Description

That function is the actual server which supplies the emWinSPY with the requested information. Simply call that function from the server thread after establishing a connection.

Prototype

```
int GUI_SPY_Process(GUI_tSend pfSend,
                    GUI_tRecv pfRecv, void * pConnectInfo);
```

Parameter	Description
pfSend	Pointer to the function to be used by the server to send data to the emWinSPY.
pfRecv	Pointer to the function to be used by the server to read data from the emWinSPY.
pConnectInfo	Pointer to be passed to the send and receive function.

Table 5.6: GUI_SPY_Process() parameter list

Additional information

The sample folder `Sample\GUI_X` contains a sample implementation of a server thread. It is located in the file `GUI_SPY_X_StartServer.c`.

Return value

The function returns 0 after the connection has been closed properly and 1 on error.

Example

```
static int _Send(const U8 * buf, int len, void * p) {
```

```
  ...
}

static int _Recv(U8 * buf, int len, void * p) {
  ...
}

static int _ServerTask(void * p) {
  int Sock;
  ...
  GUI_SPY_Process(_Send, _Recv, (void *)Sock);
}
```

GUI_SPY_SetMemHandler()

Description

This function could be used to set a memory handler for the emWinSPY. Some operations, especially collecting the windows information requires dynamic memory. That memory normally is allocated by using the emWin memory management system. With a separate memory manager (for example malloc and free) the server thread would not affect the dynamic memory manager.

Prototype

```
void GUI_SPY_SetMemHandler(void * (* pMalloc)(unsigned int),
                           void (* pFree)(void *));
```

Parameter	Description
pMalloc	Pointer to memory allocation routine.
pFree	Pointer to memory release function.

Table 5.7: () parameter list

Additional information

Using a separate memory manager is optional and not required.

GUI_SPY_StartServer()

Description

That function starts the server by calling GUI_SPY_X_StartServer() explained later and sets the required hook functions for gathering information.

Prototype

```
int GUI_SPY_StartServer(void);
```

Return value

0 on success, 1 on error.

GUI_SPY_X_StartServer()

Description

That function actually is responsible for creating the server thread and establishing a connection. When running the simulation it already contains an implementation of that function. When running on hardware that function has to be supplied by the customer.

Prototype

```
int GUI_SPY_X_StartServer(void);
```

Additional information

As already mentioned earlier the sample folder `Sample\GUI_X` contains a sample implementation of a server thread. It is located in the file `GUI_SPY_X_StartServer.c` and is written to be used with embOS/IP. In case of using different tools it should be an easy task to adapt that sample.

Return value

The function should return 0 on success and 1 on error.

Chapter 6

Displaying Text

It is very easy to display text with emWin. Knowledge of only a few routines already allows you to write any text, in any available font, at any point on the display. We first provide a short introduction to displaying text, followed by more detailed descriptions of the individual routines that are available.

6.1 Basic routines

Text can be displayed just by calling `GUI_DispString()`. For example:

```
GUI_DispString("Hello world!");
```

The above code will display the text "Hello world" at the current text position. However, there are functions to display text using different fonts or at certain positions. Even when using byte-oriented displays the position can be specified pixel accurate. In addition to that, it is also possible to display decimal, hexadecimal and binary values. Details on how values can be displayed using emWin can be found in the chapter “Displaying Values” on page 107.

Control characters

Control characters are characters with a character code of less than 32. The control characters are defined as part of ASCII. emWin ignores all control characters except the following:

Char. Code	ASCII code	C	Description
10	LF	\n	Line feed. The current text position is changed to the beginning of the next line. Per default, this is: X = 0. Y += font-distance in pixels (as delivered by `GUI_GetFontDistY()`).
13	CR	\r	Carriage return. The current text position is changed to the beginning of the current line. Per default, this is: X = 0.

Table 6.1: Control characters description table

Usage of the line feeds can be very convenient in strings. They can be part of a string so that strings spanning multiple lines can be displayed with a single function call.

6.2 Drawing modes

Text is displayed using the foreground and background color which are set using the functions `GUI_SetColor()` and `GUI_SetBkColor()`. described below. In order to set a certain drawing mode to display strings or single characters the function `GUI_SetTextMode()` can be called using the flags described below.

Normal text

Displaying normal text is the default behavior. The characters are displayed using the foreground color. The background color is used to clear the background according to its width of the text and height of the currently selected font.
Text can be displayed normally by specifying just `GUI_TM_NORMAL` or 0.

Reverse text

Text can be displayed in reverse mode by specifying `GUI_TM_REV`. This causes characters to be displayed using the background color and the background to be cleared using the foreground color.

Transparent text

Text can be displayed in transparent mode by specifying `GUI_TM_TRANS`. This causes characters to be displayed without the background to be cleared. In this case whatever was drawn before the text was displayed can still be seen.

XOR text

Text can be displayed in XOR mode by specifying `GUI_TM_XOR`. This causes characters to be displayed using the inverted colors of the background. This is done pixelwise. This is also a transparent drawing mode, so the background remains unchanged. This mode is often used to in 1bpp-configurations to ensure readability, since black is inverted to white and vice versa. In case colors are used a single pixel is inverted as follows:

```
New pixel color = number of colors - actual pixel color - 1
```

Transparent reversed text

Text can be displayed in reverse and transparent mode by specifying `(GUI_TM_TRANS | GUI_TM_REV)`. According to the transparent mode, the background is not cleared. According to the reverse mode, the characters are displayed using the background color.

Example

Displays normal, reverse, transparent, XOR, and transparent reversed text:

```
GUI_SetFont(&GUI_Font8x16);
GUI_SetBkColor(GUI_BLUE);
GUI_Clear();
GUI_SetPenSize(10);
GUI_SetColor(GUI_RED);
GUI_DrawLine(80, 10, 240, 90);
GUI_DrawLine(80, 90, 240, 10);
GUI_SetBkColor(GUI_BLACK);
GUI_SetColor(GUI_WHITE);
GUI_SetTextMode(GUI_TM_NORMAL);
GUI_DispStringHCenterAt("GUI_TM_NORMAL"           , 160, 10);
GUI_SetTextMode(GUI_TM_REV);
GUI_DispStringHCenterAt("GUI_TM_REV"              , 160, 26);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringHCenterAt("GUI_TM_TRANS"            , 160, 42);
GUI_SetTextMode(GUI_TM_XOR);
GUI_DispStringHCenterAt("GUI_TM_XOR"              , 160, 58);
GUI_SetTextMode(GUI_TM_TRANS | GUI_TM_REV);
GUI_DispStringHCenterAt("GUI_TM_TRANS | GUI_TM_REV", 160, 74);
```

Screenshot of above example

6.3 Position

Every task has a current text position. This is the position relative to the origin of the display. This position is used by text displaying functions to place the next characters. Initially this position is (0,0) which is the upper left corner of the display. When using the Window Manager the position is used according to the current window. In order to set the text position the function `GUI_GotoX()`, `GUI_GotoY()` and `GUI_GotoXY()` can be used.

6.4 Text API

The table below lists the available text-related functions in alphabetical order within their respective categories. Detailed function descriptions can be found in the following sections.

Routine	Description
Displaying text	
GUI_DispCEOL()	Clears the current line from the current position to the end.
GUI_DispChar()	Displays a single character.
GUI_DispCharAt()	Displays a single character at the specified position.
GUI_DispChars()	Displays a character a specified number of times.
GUI_DispString()	Displays a string.
GUI_DispStringAt()	Displays a string at the specified position.
GUI_DispStringAtCEOL()	Displays a string at the specified position and clears the current line to the end.
GUI_DispStringHCenterAt()	Displays a string centered horizontally at the given position.
GUI_DispStringInRect()	Displays a string in the specified rectangle.
GUI_DispStringInRectEx()	Displays a string rotated in the specified rectangle.
GUI_DispStringInRectWrap()	Displays a string wrapped in the specified rectangle.
GUI_DispStringLen()	Displays a string at the current position with specified number of characters.
GUI_WrapGetNumLines()	Returns the number lines required to display the given string using the given wrap mode at the given size.
Drawing modes	
GUI_GetTextMode()	Returns the currently set drawing mode.
GUI_SetTextMode()	Sets the drawing mode.
GUI_SetTextStyle()	Sets the style to be used.
Alignment	
GUI_GetTextAlign()	Returns the currently set text alignment.
GUI_SetLBorder()	Sets the size of the left border to be used after line feeds.
GUI_SetTextAlign()	Sets the text alignment.
Position	
GUI_DispNextLine()	Moves the cursor to the beginning of the next line.
GUI_GotoX()	Sets the X-position.
GUI_GotoXY()	Sets the X- and Y-position.
GUI_GotoY()	Sets the Y-position.
GUI_GetDispPosX()	Returns the current X-position.
GUI_GetDispPosY()	Returns the current Y-position.

Table 6.2: Text API list

6.4.1 Displaying text

GUI_DispCEOL()

Description

Clears the current line in the current window (or the display) from the current text position to the end of the window using the height of the current font.

Prototype

```
void GUI_DispCEOL(void);
```

Example

Shows "Hello world" on the display, waits 1 second and then displays "Hi" in the same place, replacing the old string:

```
GUI_DispStringAt("Hello world", 0, 0);
GUI_Delay(1000);
GUI_DispStringAt("Hi", 0, 0);
GUI_DispCEOL();
```

GUI_DispChar()

Description

Displays a single character at the current text position in the current window using the current font.

Prototype

```
void GUI_DispChar(U16 c);
```

Parameter	Description
c	Character to display.

Table 6.3: GUI_DispChar() parameter list

Additional information

This is the basic routine for displaying a single character. All other display routines (GUI_DispCharAt(), GUI_DispString(), etc.) call this routine to output the individual characters.
Which characters are available depends on the selected font. If the character is not available in the current font, nothing is displayed.

Example

Shows a capital A on the display:

```
GUI_DispChar('A');
```

Related topics

GUI_DispChars(), GUI_DispCharAt()

GUI_DispCharAt()

Description

Displays a single character at the specified position in the current window using the current font.

Prototype

```
void GUI_DispCharAt(U16 c, I16P x, I16P y);
```

Parameter	Description
c	Character to display.
x	X-position to write to in pixels of the client window.
y	Y-position to write to in pixels of the client window.

Table 6.4: GUI_DispCharAt() parameter list

Add information

Displays the character with its upper left corner at the specified (X,Y) position.
Writes the character using the routine GUI_DispChar().
If the character is not available in the current font, nothing is displayed.

Example

Shows a capital A on the display in the upper left corner:

```
GUI_DispCharAt('A',0,0);
```

Related topics

GUI_DispChar(), GUI_DispChars()

GUI_DispChars()

Description

Displays a character a specified number of times at the current text position in the current window using the current font.

Prototype

```
void GUI_DispChars(U16 c, int Cnt);
```

Parameter	Description
c	Character to display.
Cnt	Number of repetitions (0 <= Cnt <= 32767).

Table 6.5: GUI_DispChars() parameter list

Additional information

Writes the character using the routine GUI_DispChar().
If the character is not available in the current font, nothing is displayed.

Example

Shows the line "******************************" on the display:

```
GUI_DispChars('*', 30);
```

Related topics

GUI_DispChar(), GUI_DispCharAt()

GUI_DispString()

Description

Displays the string passed as parameter at the current text position in the current window using the current font.

Prototype

```
void GUI_DispString(const char * s);
```

Parameter	Description
s	String to display.

Table 6.6: GUI_DispString() parameter list

Additional information

The string can contain the control character `\n`. This control character moves the current text position to the beginning of the next line.

Example

Shows "Hello world" on the display and "Next line" on the next line:

```
GUI_DispString("Hello world");  //Disp text
GUI_DispString("\nNext line");  //Disp text
```

Related topics

`GUI_DispStringAt()`, `GUI_DispStringAtCEOL()`, `GUI_DispStringLen()`

GUI_DispStringAt()

Description

Displays the string passed as parameter at a specified position in the current window using the current font.

Prototype

```
void GUI_DispStringAt(const char * s, int x, int y);
```

Parameter	Description
`s`	String to display.
`x`	X-position to write to in pixels of the client window.
`y`	Y-position to write to in pixels of the client window.

Table 6.7: GUI_DispStringAt() parameter list

Example

Shows "Position 50,20" at position 50,20 on the display:

```
GUI_DispStringAt("Position 50,20", 50, 20);  // Disp text
```

Related topics

`GUI_DispString()`, `GUI_DispStringAtCEOL()`, `GUI_DispStringLen()`,

GUI_DispStringAtCEOL()

Description

This routine uses the exact same parameters as `GUI_DispStringAt()`. It does the same thing: displays a given string at a specified position. However, after doing so, it clears the remaining part of the line to the end by calling the routine `GUI_DispCEOL()`. This routine can be handy if one string is to overwrite another, and the overwriting string is or may be shorter than the previous one.

GUI_DispStringHCenterAt()

Description

Displays the string passed as parameter horizontally centered at a specified position in the current window using the current font.

Prototype

```
void GUI_DispStringHCenterAt(const char * s, int x, int y);
```

Parameter	Description
`s`	String to display.
`x`	X-position to write to in pixels of the client window.
`y`	Y-position to write to in pixels of the client window.

Table 6.8: GUI_DispStringHCenterAt() parameter list

GUI_DispStringInRect()

Description

Displays the string passed as parameter at a specified position within a specified rectangle, in the current window using the current font.

Prototype

```
void GUI_DispStringInRect(const char * s,
                          GUI_RECT   * pRect,
                          int          TextAlign);
```

Parameter	Description
s	String to display.
pRect	Rectangle to write to in pixels of the client window.
TextAlign	OR-combination of text alignment flags. See table below.

Table 6.9: GUI_DispStringInRect() parameter list

Permitted values for parameter TextAlign (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right.
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

Example

Shows the word "Text" centered horizontally and vertically in the current window:

```
GUI_RECT rClient;
GUI_GetClientRect(&rClient);
GUI_DispStringInRect("Text", &rClient, GUI_TA_HCENTER | GUI_TA_VCENTER);
```

Additional information

If the specified rectangle is too small, the text will be clipped.

Related topics

`GUI_DispString(), GUI_DispStringAtCEOL(), GUI_DispStringLen(),`

GUI_DispStringInRectEx()

Description

Displays the string passed as parameter at a specified position within a specified rectangle, in the current window using the current font and (optionally) rotates it.

Prototype

```
void GUI_DispStringInRectEx(const char *          s,
                            GUI_RECT *            pRect,
                            int                   TextAlign,
                            int                   MaxLen,
                            const GUI_ROTATION *  pLCD_Api);
```


Parameter	Description
s	String to display.
pRect	Rectangle to write to in pixels of the client window.
TextAlign	Alignment flags; "OR" combinable. A flag for horizontal and a flag for vertical alignment should be combined. Available flags are: GUI_TA_TOP, GUI_TA_BOTTOM, GUI_TA_VCENTER for vertical alignment. GUI_TA_LEFT, GUI_TA_RIGHT, GUI_TA_HCENTER for horizontal alignment.
MaxLen	Maximum number of characters to be shown.
pLCD_Api	See table below.

Table 6.10: GUI_DispStringInRectEx() parameter list

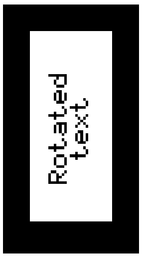
Permitted values for parameter pLCD_Api	
GUI_ROTATE_0	Does not rotate the text. Shows it from left to right.
GUI_ROTATE_180	Rotates the text by 180 degrees.
GUI_ROTATE_CCW	Rotates the text counter clockwise.
GUI_ROTATE_CW	Rotates the text clockwise.

Example

Shows the word "Text" centered horizontally and vertically in the given rectangle:

```
GUI_RECT Rect = {10, 10, 40, 80};
char acText[] = "Rotated\ntext";
GUI_SetTextMode(GUI_TM_XOR);
GUI_FillRectEx(&Rect);
GUI_DispStringInRectEx(acText, &Rect, GUI_TA_HCENTER | GUI_TA_VCENTER,
                       strlen(acText), GUI_ROTATE_CCW);
```

Screenshot of above example



Additional information

If the specified rectangle is too small, the text will be clipped.
To make the function available the configuration switch GUI_SUPPORT_ROTATION must be activated (default).

GUI_DispStringInRectWrap()

Description

Displays a string at a specified position within a specified rectangle, in the current window using the current font and (optionally) wraps the text.

Prototype

```
void GUI_DispStringInRectWrap(const char    * s,
                              GUI_RECT      * pRect,
                              int             TextAlign,
                              GUI_WRAPMODE    WrapMode);
```


Parameter	Description
s	String to display.
pRect	Rectangle to write to in pixels of the client window.
TextAlign	Alignment flags; "OR" combinable. A flag for horizontal and a flag for vertical alignment should be combined. Available flags are: GUI_TA_TOP, GUI_TA_BOTTOM, GUI_TA_VCENTER for vertical alignment. GUI_TA_LEFT, GUI_TA_RIGHT, GUI_TA_HCENTER for horizontal alignment.
WrapMode	See table below.

Table 6.11: GUI_DispStringInRectWrap() parameter list

Permitted values for parameter WrapMode	
GUI_WRAPMODE_NONE	No wrapping will be performed.
GUI_WRAPMODE_WORD	Text is wrapped word wise.
GUI_WRAPMODE_CHAR	Text is wrapped char wise.

Additional information

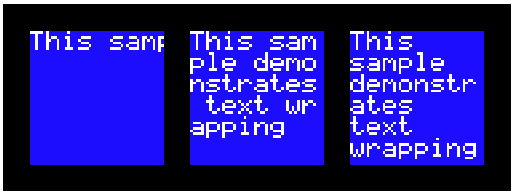
If word wrapping should be performed and the given rectangle is too small for a word char wrapping is executed at this word.

Example

Shows a text centered horizontally and vertically in the given rectangle with word wrapping:

```
GUI_WRAPMODE aWm[]    = { GUI_WRAPMODE_NONE, GUI_WRAPMODE_CHAR, GUI_WRAPMODE_WORD};
GUI_RECT     Rect     = {10, 10, 59, 59};
char         acText[] = "This example demonstrates text wrapping";
int          i;

GUI_SetTextMode(GUI_TM_TRANS);
for (i = 0; i < 3; i++) {
  GUI_SetColor(GUI_BLUE);
  GUI_FillRectEx(&Rect);
  GUI_SetColor(GUI_WHITE);
  GUI_DispStringInRectWrap(acText, &Rect, GUI_TA_LEFT, aWm[i]);
  Rect.x0 += 60;
  Rect.x1 += 60;
}
```

Screenshot of above example



GUI_DispStringLen()

Description

Displays the string passed as parameter with a specified number of characters at the current text position, in the current window using the current font.

Prototype

```
void GUI_DispStringLen(const char * s, int Len);
```

Parameter	Description
s	String to display. Should be a \0 terminated array of 8-bit character. Passing NULL as parameter is permitted.
Len	Number of characters to display.

Table 6.12: GUI_DispStringLen() parameter list

Additional information

If the string has less characters than specified (is shorter), it is padded with spaces.
If the string has more characters than specified (is longer), then only the given number of characters is actually displayed.
This function is especially useful if text messages can be displayed in different languages (and will naturally differ in length), but only a certain number of characters can be displayed.

Related topics

`GUI_DispString(), GUI_DispStringAt(), GUI_DispStringAtCEOL(),`

GUI_WrapGetNumLines()

Description

Returns the number of lines required to display the given text with the given wrap mode at the given size using the current font.

Prototype

```
int GUI_WrapGetNumLines(const char    * pText,     int xSize,
                        GUI_WRAPMODE    WrapMode);
```

Parameter	Description
pText	String to display. Should be a \0-terminated array of 8-bit characters.
xSize	X-size to be used to draw the text.
WrapMode	See table below.

Table 6.13: GUI_WrapGetNumLines() parameter list

Permitted values for parameter WrapMode	
GUI_WRAPMODE_NONE	No wrapping will be performed.
GUI_WRAPMODE_WORD	Text is wrapped word wise.
GUI_WRAPMODE_CHAR	Text is wrapped char wise.

Return value

The number of lines which is required to display the given text.

6.4.2 Drawing modes

GUI_GetTextMode()

Description

Returns the currently selected text mode.

Prototype

```
int GUI_GetTextMode(void);
```

Return value

The currently selected text mode.

GUI_SetTextMode()

Description

Sets the text mode to the parameter specified.

Prototype

```
int GUI_SetTextMode(int TextMode);
```

Parameter	Description
TextMode	Text mode to set. May be any combination of the TEXTMODE flags.

Table 6.14: GUI_SetTextMode() parameter list

Permitted values for parameter TextMode (OR-combinable)	
GUI_TEXTMODE_NORMAL	Causes text to be displayed normally. This is the default setting; the value is identical to 0.
GUI_TEXTMODE_REV	Causes text to be displayed reverse.
GUI_TEXTMODE_TRANS	Causes text to be displayed transparent.
GUI_TEXTMODE_XOR	Causes text to invert the background.

Return value

The previous selected text mode.

GUI_SetTextStyle()

Description

Sets the text style to the parameter specified.

Prototype

```
char GUI_SetTextStyle(char Style);
```

Parameter	Description
Style	Text style to set. See table below.

Table 6.15: GUI_SetTextStyle() parameter list

Permitted values for parameter Style	
GUI_TS_NORMAL	Renders text normal (default).
GUI_TS_UNDERLINE	Renders text underlined.
GUI_TS_STRIKETHRU	Renders text in strike through type.
GUI_TS_OVERLINE	Renders text in overline type.

Return value

The previous selected text style.

6.4.3 Alignment

GUI_GetTextAlign()

Description

Returns the current text alignment mode.

Prototype

```
int GUI_GetTextAlign(void);
```

GUI_SetLBorder()

Description

Sets the left border for line feeds in the current window.

Prototype

```
void GUI_SetLBorder(int x);
```

Parameter	Description
x	New left border (in pixels, 0 is left border).

Table 6.16: GUI_SetLBorder() parameter list

GUI_SetTextAlign()

Description

Sets the text alignment for the next displayed string in the current window.

Prototype

```
int GUI_SetTextAlign(int TextAlign);
```

Parameter	Description
TextAlign	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

Table 6.17: GUI_SetTextAlign() parameter list

Permitted values for parameter TextAlign (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right.
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

Return value

The selected text alignment mode.

Additional information

Setting the text alignment does not affect GUI_DispChar...()-functions. Text alignment is valid only for the current window.

Example

Displays the value 1234 with the center of the text at x = 100, y = 100:

```
GUI_SetTextAlign(GUI_TA_HCENTER | GUI_TA_VCENTER);
GUI_DispDecAt(1234, 100, 100, 4);
```


6.4.4 Position

GUI_DispNextLine()

Description

Moves the cursor to the beginning of the next line which can be adjusted using the function `GUI_SetLBorder()`.

Prototype

```
void GUI_DispNextLine(void);
```

GUI_GotoXY()

GUI_GotoX()

GUI_GotoY()

Description

Set the current text write position.

Prototypes

```
char GUI_GotoXY(int x, int y);
char GUI_GotoX(int x);
char GUI_GotoY(int y);
```

Parameter	Description
x	New X-position (in pixels, 0 is left border).
y	New Y-position (in pixels, 0 is top border).

Table 6.18: GUI_GotoXY() / GUI_GotoX() / GUI_GotoY() parameter list

Return value

0, on success. != 0, if the set text position is right or below outside the window. Consequtive drawing operations can be omitted in this case.

Example

Shows a string at position (20, 20) on the display:

```
GUI_GotoXY(20,20);
GUI_DispString("The value is");
```

GUI_GetDispPosX()

Description

Returns the current X-position.

Prototype

```
int GUI_GetDispPosX(void);
```

GUI_GetDispPosY()

Description

Returns the current Y-position.

Prototype

```
int GUI_GetDispPosY(void);
```


Chapter 7

Displaying Values

The preceding chapter explained how to show strings on the display. Of course you may use strings and the functions of the standard C library to display values. However, this can sometimes be a difficult task. It is usually much easier (and much more efficient) to call a routine that displays the value in the form that you want. emWin supports different decimal, hexadecimal and binary outputs. The individual routines are explained in this chapter.
All functions work without the usage of a floating-point library and are optimized for both speed and size. Of course `sprintf` may also be used on any system. Using the routines in this chapter can sometimes simplify things and save both ROM space and execution time.

7.1 Value API

The table below lists the available value-related routines in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Description
Displaying decimal values	
GUI_DispDec()	Displays the given value in decimal form with the specified number of characters.
GUI_DispDecAt()	Displays the given value in decimal form at the specified position with specified number of characters.
GUI_DispDecMin()	Displays the given value in decimal form with minimum number of characters.
GUI_DispDecShift()	Displays long value in decimal form with decimal point at current position with specified number of characters.
GUI_DispDecSpace()	Display value in decimal form at current position with specified number of characters, replace leading zeros with spaces.
GUI_DispSDec()	Display value in decimal form at current position with specified number of characters and sign.
GUI_DispSDecShift()	Display long value in decimal form with decimal point at current position with specified number of characters and sign.
Displaying floating-point values	
GUI_DispFloat()	Display floating-point value with specified number of characters.
GUI_DispFloatFix()	Display floating-point value with fixed no. of digits to the right of decimal point.
GUI_DispFloatMin()	Display floating-point value with minimum number of characters.
GUI_DispSFloatFix()	Display floating-point value with fixed no. of digits to the right of decimal point and sign.
GUI_DispSFloatMin()	Display floating-point value with minimum number of characters and sign.
Displaying binary values	
GUI_DispBin()	Display value in binary form at current position.
GUI_DispBinAt()	Display value in binary form at specified position.
Displaying hexadecimal values	
GUI_DispHex()	Display value in hexadecimal form at current position.
GUI_DispHexAt()	Display value in hexadecimal form at specified position.
Version of emWin	
GUI_GetVersionString()	Return the current version of emWin.

Table 7.1: Value API list

7.1.1 Displaying decimal values

GUI_DispDec()

Description

Displays a value in decimal form with a specified number of characters at the current text position, in the current window using the current font.

Prototype

```
void GUI_DispDec(I32 v, U8 Len);
```

Parameter	Description
v	Value to display. Minimum -2147483648 (= -2^31). Maximum 2147483647 (= 2^31 -1).
Len	No. of digits to display (max. 10).

Table 7.2: GUI_DispDec() parameter list

Additional information

Leading zeros are not suppressed (are shown as 0).
If the value is negative, a minus sign is shown.

Example

```
// Display time as minutes and seconds
GUI_DispString("Min:");
GUI_DispDec(Min, 2);
GUI_DispString(" Sec:");
GUI_DispDec(Sec, 2);
```

Related topics

GUI_DispSDec(), GUI_DispDecAt(), GUI_DispDecMin(), GUI_DispDecSpace()

GUI_DispDecAt()

Description

Displays a value in decimal form with a specified number of characters at a specified position, in the current window using the current font.

Prototype

```
void GUI_DispDecAt(I32 v, I16P x, I16P y, U8 Len);
```

Parameter	Description
v	Value to display. Minimum -2147483648 (= -2^31). Maximum 2147483647 (= 2^31 -1).
x	X-position to write to in pixels of the client window.
y	Y-position to write to in pixels of the client window.
Len	Number of digits to display (max. 10).

Table 7.3: GUI_DispDecAt() parameter list

Additional information

Leading zeros are not suppressed.
If the value is negative, a minus sign is shown.

Example

```
// Update seconds in upper right corner
GUI_DispDecAT(Sec, 200, 0, 2);
```

Related topics

GUI_DispDec(), GUI_DispSDec(), GUI_DispDecMin(), GUI_DispDecSpace()

GUI_DispDecMin()

Description

Displays a value in decimal form at the current text position in the current window using the current font. The length of the value does not require to be specified. The minimum length will automatically be used.

Prototype

```
void GUI_DispDecMin(I32 v);
```

Parameter	Description
v	Value to display. Minimum: -2147483648 (= -2^31); maximum 2147483647 (= 2^31 -1).

Table 7.4: GUI_DispDecMin() parameter list

Additional information

The maximum number of displayed digits is 10. This function should not be used if values have to be aligned but differ in the number of digits. Try one of the functions which require specification of the number of digits to use in this case.

Example

```
// Show result
GUI_DispString("The result is :");
GUI_DispDecMin(Result);
```

Related topics

GUI_DispDec(), GUI_DispDecAt(), GUI_DispSDec(), GUI_DispDecSpace()

GUI_DispDecShift()

Description

Displays a `long` value in decimal form with a specified number of characters and with decimal point at the current text position, in the current window using the current font.

Prototype

```
void GUI_DispDecShift(I32 v, U8 Len, U8 Shift);
```

Parameter	Description
`v`	Value to display. Minimum: -2147483648 (= -2^31); maximum: 2147483647 (= 2^31 -1).
`Len`	No. of digits to display (max. 10).
`Shift`	No. of digits to show to right of decimal point.

Table 7.5: GUI_DispDecShift() paramter list

Additional information

Watch the maximum number of 9 characters (including sign and decimal point).

GUI_DispDecSpace()

Description

Displays a value in decimal form at the current text position in the current window using the current font. Leading zeros are suppressed (replaced by spaces).

Prototype

```
void DispDecSpace(I32 v, U8 MaxDigits);
```

Parameter	Description
`v`	Value to display. Minimum: -2147483648 (= -2^31); maximum: 2147483647 (= 2^31 -1).
`MaxDigits`	No. of digits to display, including leading spaces. Maximum no. of digits displayed is 10 (excluding leading spaces).

Table 7.6: GUI_DispDecSpace() parameter list

Additional information

If values have to be aligned but differ in the number of digits, this function is a good choice.

Example

```
// Show result
GUI_DispString("The result is :");
GUI_DispDecSpace(Result, 200);
```

Related topics

GUI_DispDec(), GUI_DispDecAt(), GUI_DispSDec(), GUI_DispDecMin()

GUI_DispSDec()

Description

Displays a value in decimal form (with sign) with a specified number of characters at the current text position, in the current window using the current font.

Prototype

```
void GUI_DispSDec(I32 v, U8 Len);
```

Parameter	Description
`v`	Value to display. Minimum: -2147483648 (= -2^31); maximum: 2147483647 (= 2^31 -1).
`Len`	No. of digits to display (max. 10).

Table 7.7: GUI_DispSDec() paramter list

Additional information

Leading zeros are not suppressed.
This function is similar to `GUI_DispDec`, but a sign is always shown in front of the value, even if the value is positive.

Related topics

`GUI_DispDec()`, `GUI_DispDecAt()`, `GUI_DispDecMin()`, `GUI_DispDecSpace()`

GUI_DispSDecShift()

Description

Displays a `long` value in decimal form (with sign) with a specified number of characters and with decimal point at the current text position, in the current window using the current font.

Prototype

```
void GUI_DispSDecShift(I32 v, U8 Len, U8 Shift);
```

Parameter	Description
`v`	Value to display. Minimum: -2147483648 (= -2^31); maximum: 2147483647 (= 2^31 -1).
`Len`	No. of digits to display. (max. 8, if Shift is set; max. 9, if Shift is not set)
`Shift`	No. of digits to show to right of decimal point.

Table 7.8: GUI_DispSDecShift() parameter list

Additional information

A sign is always shown in front of the value.
Watch the maximum number of 9 characters (including sign and decimal point).

Example

```
long Value = 12345;

GUI_Init();
GUI_Clear();
GUI_SetFont(&GUI_Font8x8);
GUI_DispStringAt("GUI_DispSDecShift:\n",0,0);
GUI_DispSDecShift(Value, 7, 3);
```

Screenshot of above example

7.1.2 Displaying floating point values

GUI_DispFloat()

Description

Displays a floating point value with a specified number of characters at the current text position in the current window using the current font.

Prototype

```
void GUI_DispFloat(float v, char Len);
```

Parameter	Description
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	Number of digits to display (max. 10).

Table 7.9: GUI_DispFloat() parameter list

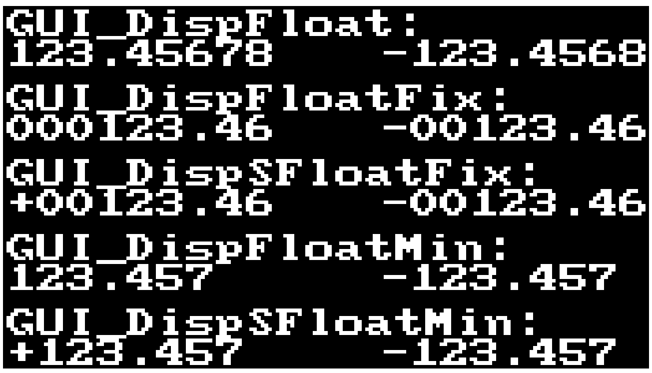
Additional information

Leading zeros are suppressed. The decimal point counts as one character.
If the value is negative, a minus sign is shown.

Example

```
// Shows different possibilities to display floating point values.
float f = 123.45678;

GUI_Clear();
GUI_SetFont(&GUI_Font8x8);
GUI_DispStringAt("GUI_DispFloat:\n", 0, 0);
GUI_DispFloat(f, 9);
GUI_GotoX(100);
GUI_DispFloat(-f, 9);
GUI_DispStringAt("GUI_DispFloatFix:\n", 0, 20);
GUI_DispFloatFix(f, 9, 2);
GUI_GotoX(100);
GUI_DispFloatFix(-f, 9, 2);
GUI_DispStringAt("GUI_DispSFloatFix:\n", 0, 40);
GUI_DispSFloatFix(f, 9, 2);
GUI_GotoX(100);
GUI_DispSFloatFix(-f, 9, 2);
GUI_DispStringAt("GUI_DispFloatMin:\n", 0, 60);
GUI_DispFloatMin(f, 3);
GUI_GotoX(100);
GUI_DispFloatMin(-f, 3);
GUI_DispStringAt("GUI_DispSFloatMin:\n", 0, 80);
GUI_DispSFloatMin(f, 3);
GUI_GotoX(100);
GUI_DispSFloatMin(-f, 3);
```

Screenshot of above example



GUI_DispFloatFix()

Description

Displays a floating-point value with specified number of total characters and a specified number of characters to the right of the decimal point, at the current text position in the current window using the current font.

Prototype

```
void GUI_DispFloatFix(float v, char Len, char Decs);
```

Parameter	Description
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	Number of digits to display (max. 10).
Decs	Number of digits to show to the right of the decimal point.

Table 7.10: GUI_DispFloatFix() parameter list

Additional information

Leading zeros are not suppressed.
If the value is negative, a minus sign is shown.

GUI_DispFloatMin()

Description

Displays a floating-point value with a minimum number of decimals to the right of the decimal point, at the current text position in the current window using the current font.

Prototype

```
void GUI_DispFloatMin(float v, char Fract);
```

Parameter	Description
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Fract	Minimum number of characters to display.

Table 7.11: GUI_DispFloatMin() paramter list

Additional information

Leading zeros are suppressed. If the value is negative, a minus sign is shown. The length does not need to be specified. The minimum length will automatically be used. If values have to be aligned but differ in the number of digits, one of the "...Fix()"-functions should be used instead.

GUI_DispSFloatFix()

Description

Displays a floating-point value (with sign) with a specified number of total characters and a specified number of characters to the right of the decimal point, in the current window using the current font.

Prototype

```
void GUI_DispSFloatFix(float v, char Len, char Decs);
```

Parameter	Description
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	Number of digits to display (max. 10).
Decs	Number of digits to show to the right of the decimal point.

Table 7.12: GUI_DispSFloatFix() parameter list

Additional information

Leading zeros are not suppressed. A sign is always shown in front of the value.

GUI_DispSFloatMin()

Description

Displays a floating-point value (with sign) with a minimum number of decimals to the right of the decimal point, at the current text position in the current window using the current font.

Prototype

```
void GUI_DispSFloatMin(float v, char Fract);
```

Parameter	Description
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Fract	Minimum number of digits to display.

Table 7.13: GUI_DispSFloatMin() parameter list

Additional information

Leading zeros are suppressed. A sign is always shown in front of the value. The length does not need to be specified. The minimum length will automatically be used. If values have to be aligned but differ in the number of digits, one of the "...Fix()"-functions should be used instead.

7.1.3 Displaying binary values

GUI_DispBin()

Description

Displays a value in binary form at the current text position in the current window using the current font.

Prototype

```
void GUI_DispBin(U32 v, U8 Len);
```

Parameter	Description
v	Value to display, 32-bit.
Len	No. of digits to display (including leading zeros).

Table 7.14: GUI_DispBin() parameter list

Additional information

As with decimal and hexadecimal values, the least significant bit is rightmost.

Example

```
//
// Show binary value 7, result: 000111
//
U32 Input = 0x7;
GUI_DispBin(Input, 6);
```

Related topics

GUI_DispBinAt()

GUI_DispBinAt()

Description

Displays a value in binary form at a specified position in the current window using the current font.

Prototype

```
void GUI_DispBinAt(U32 v, I16P x, I16P y, U8 Len);
```

Parameter	Description
v	Value to display, 16-bit.
x	X-position to write to in pixels of the client window.
y	Y-position to write to in pixels of the client window.
Len	No. of digits to display (including leading zeroes).

Table 7.15: GUI_DispBinAt() parameter list

Additional information

As with decimal and hexadecimal values, the least significant bit is rightmost.

Example

```
//
// Show binary input status
//
GUI_DispBinAt(Input, 0, 0, 8);
```

Related topics

GUI_DispBin(), GUI_DispHex()

7.1.4 Displaying hexadecimal values

GUI_DispHex()

Description

Displays a value in hexadecimal form at the current text position in the current window using the current font.

Prototype

```
void GUI_DispHex(U32 v, U8 Len);
```

Parameter	Description
v	Value to display, 16-bit.
Len	No. of digits to display.

Table 7.16: GUI_DispHex() parameter list

Additional information

As with decimal and binary values, the least significant bit is rightmost.

Example

```
//
// Show value of AD-converter
//
GUI_DispHex(Input, 4);
```

Related topics

GUI_DispDec(), GUI_DispBin(), GUI_DispHexAt()

GUI_DispHexAt()

Description

Displays a value in hexadecimal form at a specified position in the current window using the current font.

Prototype

```
void GUI_DispHexAt(U32 v, I16P x, I16P y, U8 Len);
```

Parameter	Description
v	Value to display, 16-bit.
x	X-position to write to in pixels of the client window.
y	Y-position to write to in pixels of the client window.
Len	No. of digits to display.

Table 7.17: GUI_DispHexAt() parameter list

Additional information

As with decimal and binary values, the least significant bit is rightmost.

Example

```
//
// Show value of AD-converter at specified position
//
GUI_DispHexAt(Input, 0, 0, 4);
```

Related topics

GUI_DispDec(), GUI_DispBin(), GUI_DispHex()

7.1.5 Version of emWin

GUI_GetVersionString()

Description

Returns a string containing the current version of emWin.

Prototype

```
const char * GUI_GetVersionString(void);
```

Example

```
//
// Displays the current version at the current cursor position
//
GUI_DispString(GUI_GetVersionString());
```

Chapter 8

2-D Graphic Library

emWin contains a complete 2-D graphic library which should be sufficient for most applications. The routines supplied with emWin can be used with or without clipping (refer to the chapter "The Window Manager (WM)" on page 375) and are based on fast and efficient algorithms. Currently, only the `GUI_DrawArc()` function requires floating-point calculations.

8.1 Graphic API

The table below lists the available graphic-related routines in alphabetical order within their respective categories. Detailed descriptions can be found in the sections that follow.

Routine	Description
Drawing related functions	
GUI_GetClientRect()	Returns the current available drawing area.
GUI_GetDrawMode()	Returns the current drawing mode.
GUI_GetPenSize()	Returns the current pen size in pixels.
GUI_GetPixelIndex()	Returns the color index of a given position.
GUI_SetClipRect()	Sets the rectangle used for clipping.
GUI_SetDrawMode()	Sets the drawing mode.
GUI_SetPenSize()	Sets the pen size in pixels.
Basic drawing routines	
GUI_Clear()	Fills the display / the active window with the background color.
GUI_ClearRect()	Fills a rectangular area with the background color.
GUI_CopyRect()	Copies a rectangle area on the display
GUI_DrawGradientH()	Draws a rectangle filled with a horizontal color gradient.
GUI_DrawGradientV()	Draws a rectangle filled with a vertical color gradient.
GUI_DrawGradientRoundedH()	Draws a rectangle with rounded corners filled with a horizontal color gradient.
GUI_DrawGradientRoundedV()	Draws a rectangle with rounded corners filled with a vertical color gradient.
GUI_DrawPixel()	Draws a single pixel.
GUI_DrawPoint()	Draws a point.
GUI_DrawRect()	Draws a rectangle.
GUI_DrawRectEx()	Draws a rectangle.
GUI_DrawRoundedFrame()	Draws a frame with rounded corners.
GUI_DrawRoundedRect()	Draws a rectangle with rounded corners.
GUI_FillRect()	Draws a filled rectangle.
GUI_FillRectEx()	Draws a filled rectangle.
GUI_FillRoundedRect()	Draws a filled rectangle with rounded corners.
GUI_InvertRect()	Invert a rectangular area.
Alpha blending	
GUI_EnableAlpha()	Enables/disables automatic alpha blending
GUI_PreserveTrans()	Makes sure that alpha channel remains after drawing operations.
GUI_RestoreUserAlpha()	Restores the previous state of user alpha blending
GUI_SetAlpha()	Sets the current alpha blending value. (Obsolete)
GUI_SetUserAlpha()	Sets an additional value which is used to calculate the actual alpha blending value to be used.
Drawing bitmaps	
GUI_DrawBitmap()	Draws a bitmap.
GUI_DrawBitmapEx()	Draws a scaled bitmap.
GUI_DrawBitmapHWAlpha()	Draws a bitmap with alpha blending information on a system with hardware alpha blending support.
GUI_DrawBitmapMag()	Draws a magnified bitmap.

Table 8.1: Graphic API list

Routine	Description
GUI_SetAlphaMask8888()	Could be used for setting an additional AND and OR mask to be used for drawing the pixels of 32bpp bitmaps.
Drawing streamed bitmaps	
GUI_CreateBitmapFromStream()	Creates a bitmap from a given stream of any type.
GUI_CreateBitmapFromStreamIDX()	Creates a bitmap from an index based bitmap stream.
GUI_CreateBitmapFromStreamRLE4()	Creates a bitmap from an RLE4 bitmap stream.
GUI_CreateBitmapFromStreamRLE8()	Creates a bitmap from an RLE8 bitmap stream.
GUI_CreateBitmapFromStream444_12()	Creates a bitmap from a 12bpp (444_12) bitmap stream.
GUI_CreateBitmapFromStream444_12_1()	Creates a bitmap from a 12bpp (444_12_1) bit-map stream.
GUI_CreateBitmapFromStreamM444_12()	Creates a bitmap from a 12bpp (M444_12) bit-map stream.
GUI_CreateBitmapFromStreamM444_12_1()	Creates a bitmap from a 12bpp (M444_12_1 bit-map stream.
GUI_CreateBitmapFromStream444_16()	Creates a bitmap from a 12bpp (444_16) bitmap stream.
GUI_CreateBitmapFromStreamM444_16()	Creates a bitmap from a 12bpp (444_16_1) bit-map stream.
GUI_CreateBitmapFromStreamA555()	Creates a bitmap with an alpha channel from a 16bpp (A555) bitmap stream.
GUI_CreateBitmapFromStreamAM555()	Creates a bitmap with an alpha channel from a 16bpp (AM555) bitmap stream.
GUI_CreateBitmapFromStreamA565()	Creates a bitmap with an alpha channel from a 16bpp (A565) bitmap stream.
GUI_CreateBitmapFromStreamAM565()	Creates a bitmap with an alpha channel from a 16bpp (AM565) bitmap stream.
GUI_CreateBitmapFromStream565()	Creates a bitmap from a 16bpp (565) bitmap stream.
GUI_CreateBitmapFromStreamM565()	Creates a bitmap from a 16bpp (M565) bitmap stream with red and blue swapped.
GUI_CreateBitmapFromStream555()	Creates a bitmap from a 16bpp (555) bitmap stream.
GUI_CreateBitmapFromStreamM555()	Creates a bitmap from a 16bpp (M555) bitmap stream with red and blue swapped.
GUI_CreateBitmapFromStreamRLE16()	Creates a bitmap from an RLE16 (565) bitmap stream.
GUI_CreateBitmapFromStreamRLEM16()	Creates a bitmap from an RLEM16 (M565) bit-map stream with red and blue swapped.
GUI_CreateBitmapFromStream24()	Creates a bitmap from a 24 bit bitmap stream.
GUI_CreateBitmapFromStreamAlpha()	Creates a bitmap from a 32 bit bitmap stream.
GUI_CreateBitmapFromStreamRLEAlpha()	Creates a bitmap from an RLE compressed 8 bit alpha bitmap stream.
GUI_CreateBitmapFromStreamRLE32()	Creates a bitmap from an RLE32 bitmap stream.
GUI_DrawStreamedBitmap()	Draws a bitmap from an indexed based bitmap stream (1 - 8bpp).
GUI_DrawStreamedBitmapAuto()	Draws a bitmap from a bitmap stream of any supported format.
GUI_DrawStreamedBitmapEx()	Draws a bitmap from an indexed based bitmap stream (1 - 8bpp) without loading the complete image.
GUI_DrawStreamedBitmapExAuto()	Draws a bitmap from a bitmap stream of any supported format without loading the complete image.
GUI_DrawStreamedBitmapA555Ex()	Draws a bitmap from a 16bpp (A555) bitmap stream with alpha channel without loading the complete image.
GUI_DrawStreamedBitmapAM555Ex()	Draws a bitmap from a 16bpp (AM555) bitmap stream with alpha channel without loading the complete image.

Table 8.1: Graphic API list

Routine	Description
GUI_DrawStreamedBitmapA565Ex()	Draws a bitmap from a 16bpp (AM565) bitmap stream with alpha channel without loading the complete image.
GUI_DrawStreamedBitmapAM565Ex()	Draws a bitmap from a 16bpp (AM565) bitmap stream with alpha channel without loading the complete image.
GUI_DrawStreamedBitmap555Ex()	Draws a bitmap from a 16bpp (555) bitmap stream without loading the complete image.
GUI_DrawStreamedBitmapM555Ex()	Draws a bitmap from a 16bpp (M555) bitmap stream without loading the complete image.
GUI_DrawStreamedBitmap565Ex()	Draws a bitmap from a 16bpp (565) bitmap stream without loading the complete image.
GUI_DrawStreamedBitmapM565Ex()	Draws a bitmap from a 16bpp (M565) bitmap stream without loading the complete image.
GUI_DrawStreamedBitmap24Ex()	Draws a bitmap from a 24bpp bitmap stream without loading the complete image.
GUI_GetStreamedBitmapInfo()	Returns information about the given stream.
GUI_GetStreamedBitmapInfoEx()	Returns information about the given stream which can be located on any kind of media.
GUI_SetStreamedBitmapHook()	Sets a hook function for GUI_DrawStreamedBitmapEx().
Drawing lines	
GUI_DrawHLine()	Draws a horizontal line.
GUI_DrawLine()	Draws a line from a specified start point to a specified end point (absolute coordinates).
GUI_DrawLineRel()	Draws a line from the current position to an endpoint specified by X- and Y-distances (relative coordinates).
GUI_DrawLineTo()	Draws a line from the current position to a specified endpoint.
GUI_DrawPolyLine()	Draws a polyline.
GUI_DrawVLine()	Draws a vertical line.
GUI_GetLineStyle()	Returns the current line style.
GUI_MoveRel()	Moves the line pointer relative to its current position.
GUI_MoveTo()	Moves the line pointer to the given position.
GUI_SetLineStyle()	Sets the current line style.
Drawing polygons	
GUI_DrawPolygon()	Draws the outline of a polygon.
GUI_EnlargePolygon()	Enlarges a polygon.
GUI_FillPolygon()	Draws a filled polygon.
GUI_MagnifyPolygon()	Magnifies a polygon.
GUI_RotatePolygon()	Rotates a polygon by a specified angle.
Drawing circles	
GUI_DrawCircle()	Draws the outline of a circle.
GUI_FillCircle()	Draws a filled circle.
Drawing ellipses	
GUI_DrawEllipse()	Draws the outline of an ellipse.
GUI_FillEllipse()	Draws a filled ellipse.
Drawing arcs	
GUI_DrawArc()	Draws an arc.
Drawing a graph	
GUI_DrawGraph()	Draws a graph.
Drawing a pie chart	
GUI_DrawPie()	Draws a circle sector.
Saving and restoring the GUI-context	
GUI_RestoreContext()	Restores the GUI-context.

Table 8.1: Graphic API list

Routine	Description
GUI_SaveContext()	Saves the GUI-context.
Info about screen changes	
GUI_DIRTYDEVICE_Create()	Creates a DIRTYDEVICE object.
GUI_DIRTYDEVICE_CreateEx()	Creates a DIRTYDEVICE object in the given layer.
GUI_DIRTYDEVICE_Delete()	Deletes a DIRTYDEVICE object.
GUI_DIRTYDEVICE_DeleteEx()	Deletes a DIRTYDEVICE object from the given layer.
GUI_DIRTYDEVICE_Fetch()	Fetches information from a DIRTYDEVICE.
GUI_DIRTYDEVICE_FetchEx()	Fetches information from a DIRTYDEVICE of the given layer.
Avoiding tearing effects	
GUI_SetRefreshHook()	Sets a callback function which waits until the vertical non display period has been reached before updating the screen.

Table 8.1: Graphic API list

8.1.1 Drawing related functions

GUI_GetClientRect()

Description

The current client rectangle depends on using the Window Manager or not. If using the Window Manager the function uses WM_GetClientRect to retrieve the client rectangle. If not using the Window Manager the client rectangle corresponds to the complete LCD display.

Prototype

```
void GUI_GetClientRect(GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to the GUI_RECT-structure which is filled with the coordinates of the client rectangle.

Table 8.2: GUI_GetClientRect() parameter list

GUI_GetDrawMode()

Description

Returns the current drawing mode.

Prototype

```
GUI_DRAWMODE GUI_GetDrawMode(void);
```

Return value

The currently selected drawing mode.

GUI_GetPenSize()

Description

Returns the current pen size.

Prototype

```
U8 GUI_GetPenSize(void);
```


GUI_GetPixelIndex()

Description

Returns the color index of a given position.

Prototype

```
unsigned GUI_GetPixelIndex(int x, int y);
```

Parameter	Description
x	absolute x-position of the pixel
y	absolute y-position of the pixel

Table 8.3: GUI_GetPixelIndex() parameter list

GUI_SetClipRect()

Description

Sets the clipping rectangle used for limiting the output.

Prototype

```
void GUI_SetClipRect(const GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to the rectangle which should be used for clipping. A NULL pointer should be used to restore the default value.

Table 8.4: GUI_SetClipRect() parameter list

Additional information

The clipping area is limited to the configured (virtual) display size per default. Under some circumstances it can be useful to use a smaller clipping rectangle, which can be set using this function. The rectangle referred to should remain unchanged until the function is called again with a NULL pointer.

Example

The following example shows how to use the function:

```
GUI_RECT Rect = {10, 10, 100, 100};
GUI_SetClipRect(&Rect);
.
. // Draw something...
.
GUI_SetClipRect(NULL);
```

GUI_SetDrawMode()

Description

Selects the specified drawing mode.

Prototype

```
GUI_DRAWMODE GUI_SetDrawMode(GUI_DRAWMODE dm);
```

Parameter	Description
dm	Drawing mode to set. Permitted values are listed below.

Table 8.5: GUI_SetDrawMode() parameter list

Permitted values for parameter dm	
GUI_DM_NORMAL	Default: The content of the display is overdrawn by the graphic.
GUI_DM_XOR	The content of the display is inverted when it is overdrawn. Restrictions are listed below.

Restrictions

- XOR mode is useful only when using two displayed colors inside the active window or screen.
- Functions which make use of the pen size might not work properly if the drawing mode is XOR and the pen size is unequal to 1. So before using one of those functions either the drawing mode should be set to NORMAL or the pen size should be set to 1. The functions which regard the pen size are listed in the description of “GUI_SetPenSize()” on page 123.
- When drawing bitmaps with a color depth greater than 1 bit per pixel (bpp) this drawing mode takes no effect.
- When using drawing functions such as GUI_DrawPolyLine() or multiple calls of GUI_DrawLineTo(), the fulcrums are inverted twice. The result is that these pixels remain in the background color.

Return value

The previously set drawing mode.

Additional information

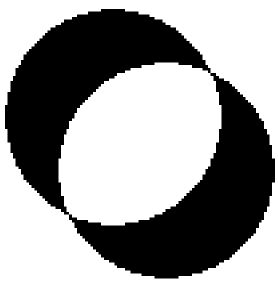
If using colors, an inverted pixel is calculated as follows:

```
New pixel color = number of colors - actual pixel color - 1
```

Example

```
//
// Showing two circles, the second one XOR-combined with the first:
//
GUI_Clear();
GUI_SetDrawMode(GUI_DRAWMODE_NORMAL);
GUI_FillCircle(120, 64, 40);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(140, 84, 40);
```

Screenshot of above example



GUI_SetPenSize()

Description

Sets the pen size to be used for further drawing operations.

Prototype

```
U8 GUI_SetPenSize(U8 PenSize);
```

Parameter	Description
PenSize	Pen size in pixels to be used.

Table 8.6: GUI_SetPenSize() parameter list

Return value

Previous pen size.

Additional information

The pen size should be >= 1. It is not possible to combine line styles with a pen size > 1. The following vector drawing operations are affected by the pen size:

- GUI_DrawPoint()
- GUI_DrawLine()
- GUI_DrawLineRel()
- GUI_DrawLineTo()
- GUI_DrawPolyLine()
- GUI_DrawPolygon()
- GUI_DrawEllipse()
- GUI_DrawArc()

8.1.2 Basic drawing routines

The basic drawing routines allow drawing of individual points, horizontal and vertical lines and shapes at any position on the display. Any available drawing mode can be used. Since these routines are called frequently in most applications, they are optimized for speed as much as possible. For example, the horizontal and vertical line functions do not require the use of single-dot routines.

GUI_Clear()

Description

Clears the current window.

Prototype

```
void GUI_Clear(void);
```

Additional information

If no window has been defined, the current window is the entire display. In this case, the entire display is cleared.

Example

Shows "Hello world" on the display, waits 1 second and then clears the display:

```
GUI_DispStringAt("Hello world", 0, 0);  // Display text.
GUI_Delay(1000);                        // Wait 1 second.
GUI_Clear();                            // Clear screen.
```

GUI_ClearRect()

Description

Clears a rectangular area at a specified position in the current window by filling it with the background color.

Prototype

```
void GUI_ClearRect(int x0, int y0, int x1, int y1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.

Table 8.7: GUI_ClearRect() parameter list

Related topics

GUI_InvertRect(), GUI_FillRect()

GUI_CopyRect()

Description

Copies the content of the given rectangular area to the specified position.

Prototype

```
void GUI_CopyRect(int x0, int y0, int x1, int y1, int xSize, int ySize);
```

Parameter	Description
x0	Upper left X-position of the source rectangle.
y0	Upper left Y-position of the source rectangle.
x1	Upper left X-position of the destination rectangle.
y1	Upper left Y-position of the destination rectangle.
xSize	X-size of the rectangle.
ySize	Y-size of the rectangle.

Table 8.8: GUI_CopyRect() parameter list

Additional information

The source and destination rectangle may overlap each other.

GUI_DrawGradientH()

Description

Draws a rectangle filled with a horizontal color gradient.

Prototype

```
void GUI_DrawGradientH(int x0, int y0, int x1, int y1,
                       GUI_COLOR Color0, GUI_COLOR Color1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
Color0	Color to be drawn on the leftmost side of the rectangle.
Color1	Color to be drawn on the rightmost side of the rectangle.

Table 8.9: GUI_DrawGradientH() parameter list

Example

```
GUI_DrawGradientH(0, 0, 99, 99, 0x0000FF, 0x00FFFF);
```

Screenshot of above example

GUI_DrawGradientV()

Description

Draws a rectangle filled with a vertical color gradient.

Prototype

```
void GUI_DrawGradientV(int x0, int y0, int x1, int y1,
```



```
GUI_COLOR Color0, GUI_COLOR Color1);
```

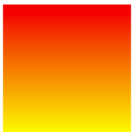
Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
Color0	Color to be drawn on the topmost side of the rectangle.
Color1	Color to be drawn on the bottommost side of the rectangle.

Table 8.10: GUI_DrawGradientV() parameter list

Example

```
GUI_DrawGradientV(0, 0, 99, 99, 0x0000FF, 0x00FFFF);
```

Screenshot of above example



GUI_DrawGradientRoundedH()

Description

Draws a rectangle with rounded corners filled with a horizontal color gradient.

Prototype

```
void GUI_DrawGradientRoundedH(int x0, int y0, int x1, int y1, int rd
                              GUI_COLOR Color0, GUI_COLOR Color1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
rd	Radius to be used for the rounded corners.
Color0	Color to be drawn on the leftmost side of the rectangle.
Color1	Color to be drawn on the rightmost side of the rectangle.

Table 8.11: GUI_DrawGradientRoundedH() parameter list

Example

```
GUI_DrawGradientRoundedH(0, 0,  99, 99, 25, 0x0000FF, 0x00FFFF);
```

Screenshot of above example

GUI_DrawGradientRoundedV()

Description

Draws a rectangle with rounded corners filled with a vertical color gradient.

Prototype

```
void GUI_DrawGradientRoundedV(int x0,   int y0, int x1,   int y1,
```



```
GUI_COLOR Color0, GUI_COLOR Color1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
Color0	Color to be drawn on the leftmost side of the rectangle.
Color1	Color to be drawn on the rightmost side of the rectangle.

Table 8.12: GUI_DrawGradientRoundedV() parameter list

Example

```
GUI_DrawGradientRoundedV(0, 0, 99, 99, 25, 0x0000FF, 0x00FFFF);
```

Screenshot of above example

GUI_DrawPixel()

Description

Draws a pixel at a specified position in the current window.

Prototype

```
void GUI_DrawPixel(int x, int y);
```

Parameter	Description
x	X-position of pixel.
y	Y-position of pixel.

Table 8.13: GUI_DrawPixel() parameter list

Related topics

GUI_DrawPoint()

GUI_DrawPoint()

Description

Draws a point with the current pen size at a specified position in the current window.

Prototype

```
void GUI_DrawPoint(int x, int y);
```

Parameter	Description
x	X-position of point.
y	Y-position of point.

Table 8.14: GUI_DrawPoint() parameter list

Related topics

GUI_DrawPixel()

GUI_DrawRect()

Description

Draws a rectangle at a specified position in the current window.

Prototype

```
void GUI_DrawRect(int x0, int y0, int x1, int y1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.

Table 8.15: GUI_DrawRect() parameter list

GUI_DrawRectEx()

Description

Draws a rectangle at a specified position in the current window.

Prototype

```
void GUI_DrawRectEx(const GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to a GUI_RECT-structure containing the coordinates of the rectangle

Table 8.16: GUI_DrawRectEx() parameter list

GUI_DrawRoundedFrame()

Description

Draws a frame at a specified position in the current window with rounded corners an a specified width.

Prototype

```
void GUI_DrawRoundedFrame(int x0, int y0, int x1, int y1, int r, int w);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
r	Radius to be used for the rounded corners.
w	Width in which the frame is drawn.

Table 8.17: GUI_DrawRoundedFrame() parameter list

GUI_DrawRoundedRect()

Description

Draws a rectangle at a specified position in the current window with rounded corners.

Prototype

```
void GUI_DrawRoundedRect(int x0, int y0, int x1, int y1, int r);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
r	Radius to be used for the rounded corners.

Table 8.18: GUI_DrawRoundedRect() parameter list

GUI_FillRect()

Description

Draws a filled rectangular area at a specified position in the current window.

Prototype

```
void GUI_FillRect(int x0, int y0, int x1, int y1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.

Table 8.19: GUI_FillRect() parameter list

Additional information

Uses the current drawing mode, which normally means all pixels inside the rectangle are set.

Related topics

GUI_InvertRect(), GUI_ClearRect()

GUI_FillRectEx()

Description

Draws a filled rectangle at a specified position in the current window.

Prototype

```
void GUI_FillRectEx(const GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to a GUI_RECT-structure containing the coordinates of the rectangle

Table 8.20: GUI_FillRectEx() parameter list

GUI_FillRoundedRect()

Description

Draws a filled rectangle at a specified position in the current window with rounded corners.

Prototype

```
void GUI_FillRoundedRect(int x0, int y0, int x1, int y1, int r);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.
r	Radius to be used for the rounded corners.

Table 8.21: GUI_FillRoundedRect() parameter list

GUI_InvertRect()

Description

Draws an inverted rectangular area at a specified position in the current window.

Prototype

```
void GUI_InvertRect(int x0, int y0, int x1, int y1);
```

Parameter	Description
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.

Table 8.22: GUI_InvertRect() parameter list

Related topics

GUI_FillRect(), GUI_ClearRect()

8.1.3 Alpha blending

Alpha blending is a method of combining a foreground image with the background to create the appearance of semi transparency. An alpha value determines how much of a pixel should be visible and how much of the background should show through.

Color information

emWin internally works with 32 bits of color information. It is important to know that emWin is able to use 2 different color formats. For details please refer to “Logical colors” on page 292. That chapter explains the differences between both logical color modes. Important here is to know that when using the default logical color format (ABGR) an an alpha value of 0 means opaque and a value of 255 means completely transparent. In case of using ARGB the meaning is vice versa: 0 means completely transparent and 255 means opaque. The documentation assumes using the default format (ABGR).

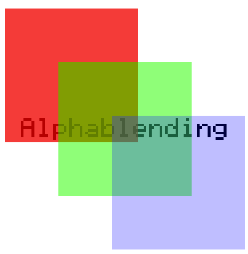
How it works

The alpha blending is done completely automatically once it is enabled by using the function GUI_EnableAlpha(). This makes emWin regard the upper 8 bits of the color information as alpha value. Enabling alpha blending is required only for functions which use the background or foreground color. Bitmaps which already contain alpha values (32bpp) are automatically displayed properly, so enabling alpha blending is not required in this case.

Example

The following small example shows how it works:

```
GUI_EnableAlpha(1);
GUI_SetBkColor(GUI_WHITE);
GUI_Clear();
GUI_SetColor(GUI_BLACK);
GUI_DispStringHCenterAt("Alphablending", 45, 41);
GUI_SetColor((0x40uL << 24) | GUI_RED);
GUI_FillRect(0, 0, 49, 49);
GUI_SetColor((0x80uL << 24) | GUI_GREEN);
GUI_FillRect(20, 20, 69, 69);
GUI_SetColor((0xC0uL << 24) | GUI_BLUE);
GUI_FillRect(40, 40, 89, 89);
```



Older versions

In older versions it was required to use the function GUI_SetAlpha() for blending the foreground with the current background color information. This also works but is no longer required.

GUI_EnableAlpha()

Description

Enables or disables automatic alpha blending.

Prototype

```
unsigned GUI_EnableAlpha(unsigned OnOff);
```

Parameter	Description
OnOff	1 enables automatic alpha blending, 0 disables it.

Table 8.23: GUI_EnableAlpha() parameter list

Return value

Old state.

Additional information

After enabling automatic alpha blending the color information of each object automatically determines its transparency.

GUI_PreserveTrans()

Description

Drawing items using an alpha value normally requires mixing the content of the framebuffer with the item color. Mixing is done by using the alpha channel of the item color as intensity information for mixing the colors. After the drawing operation the alpha value is normally lost.
But there could be situations where mixing is not wanted, for example when working with a multi layer hardware. If the alpha values of a bitmap or set by GUI_SetAlpha() should remain as alpha channel in the frame buffer this function should be called immediately before the drawing operation. To avoid unwanted behavior and side effects on other drawing operations it is recommended to disable that function after use.

Prototype

```
unsigned GUI_PreserveTrans(unsigned OnOff);
```

Parameter	Description
OnOff	1 enables transparency preserving, 0 disables it.

Table 8.24: GUI_PreserveTrans() parameter list

Return value

Old state.

GUI_SetAlpha()

Description

Enables software alpha blending for all subsequent drawing operations.

Prototype

```
unsigned GUI_SetAlpha(U8 Value);
```

Parameter	Description
Alpha	Alpha value to be used for all subsequent drawing operations. Default is 0 which means no alpha blending.

Table 8.25: GUI_SetAlpha() parameter list

Return value

Previous value used for alpha blending.

Additional information

The function sets the alpha value to be used for all subsequent drawing operations. A value of 0 for parameter `Alpha` means opaque (alpha blending disabled) and a value of 255 means completely transparent (invisible).

Note that software alpha blending increases the CPU load. Further it is strongly recommended to set the alpha value back to the default value after finishing the drawing operations.

Example

```
extern const GUI_BITMAP _LogoBitmap;

GUI_SetColor(GUI_BLUE);
GUI_FillCircle(100, 50, 49);
GUI_SetColor(GUI_YELLOW);
for (i = 0; i < 100; i++) {
  U8 Alpha;
  Alpha = (i * 255 / 100);
  GUI_SetAlpha(Alpha);
  GUI_DrawHLine(i, 100 - i, 100 + i);
}
GUI_SetAlpha(0x80);
GUI_DrawBitmap(&_LogoBitmap, 30, 30);
GUI_SetColor(GUI_MAGENTA);
GUI_SetFont(&GUI_Font24B_ASCII);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringHCenterAt("Alphablending", 100, 3);
GUI_SetAlpha(0);                                        /* Set back to default (opaque) */
```

Screenshot of above example

GUI_SetUserAlpha()

Description

Sets an additional value which is used to calculate the actual alpha value to be used. The actual alpha value is calculated as follows:

Alpha = AlphaFromObject + ((255 - AlphaFromObject) * UserAlpha) / 255

Prototype

```
U32 GUI_SetUserAlpha(GUI_ALPHA_STATE * pAlphaState, U32 UserAlpha);
```

Parameter	Description
pAlphaState	Pointer to an GUI_ALPHA_STATE structure to be used to save the current state.
UserAlpha	Value to be used.

Table 8.26: GUI_SetUserAlpha() parameter list

Elements of GUI_ALPHA_STATE

Data type	Element	Description
U32	UserAlpha	Alpha value to be used.

Table 8.27: GUI_ALPHA_STATE element list

Return value

Previous user alpha value.

Additional information

The following function `GUI_RestoreUserAlpha()` can be used to restore the previous state of the function.

GUI_RestoreUserAlpha()

Description

Restores the previous state of user alpha blending. saved in the structure pointed by.

Prototype

```
U32 GUI_RestoreUserAlpha(GUI_ALPHA_STATE * pAlphaState);
```

Parameter	Description
pAlphaState	Pointer to an GUI_ALPHA_STATE structure containing information of the previous state to be restored.

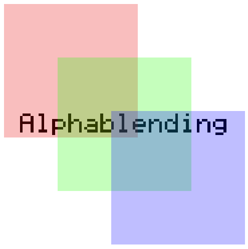
Table 8.28: GUI_RestoreUserAlpha() parameter list

Return value

Current user alpha value.

Example

```
{
  GUI_ALPHA_STATE AlphaState;

  GUI_EnableAlpha(1);
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetColor(GUI_BLACK);
  GUI_DispStringHCenterAt("Alphablending", 45, 41);
  GUI_SetUserAlpha(&AlphaState, 0xC0);
  GUI_SetColor(GUI_RED);
  GUI_FillRect(0, 0, 49, 49);
  GUI_SetColor(GUI_GREEN);
  GUI_FillRect(20, 20, 69, 69);
  GUI_SetColor(GUI_BLUE);
  GUI_FillRect(40, 40, 89, 89);
  GUI_RestoreUserAlpha(&AlphaState);
}
```



8.1.4 Drawing bitmaps

Generally emWin is able to display any bitmap image at any display position. On 16 bit CPUs (sizeof(int) == 2), the size of one bitmap per default is limited to 64 kb. If larger bitmaps should be displayed with a 16 bit CPU, refer to the chapter “Configuration” on page 1203.

GUI_DrawBitmap()

Description

Draws a bitmap image at a specified position in the current window.

Prototype

```
void GUI_DrawBitmap(const GUI_BITMAP * pBM, int x, int y);
```

Parameter	Description
pBM	Pointer to the bitmap to display.
x	X-position of the upper left corner of the bitmap in the display.
y	Y-position of the upper left corner of the bitmap in the display.

Table 8.29: GUI_DrawBitmap() parameter list

Additional information

The picture data is interpreted as bit stream starting with the most significant bit (msb) of the first byte.
A new line always starts at an even byte address, as the `n`th line of the bitmap starts at offset (n * BytesPerLine). The bitmap can be shown at any point in the client area. Usually, the Bitmap Converter is used to generate bitmaps. Detailed information can be found in the chapter “Bitmap Converter” on page 189.

Example

```
extern const GUI_BITMAP bmSeggerLogoBlue;  /* declare external Bitmap */

void main() {
  GUI_Init();
  GUI_DrawBitmap(&bmSeggerLogoBlue, 45, 20);
}
```

Screenshot of above example

GUI_DrawBitmapEx()

Description

This routine makes it possible to scale and/or to mirror a bitmap on the display.

Prototype

```
void GUI_DrawBitmapEx(const GUI_BITMAP * pBitmap,
                      int x0,       int y0,
                      int xCenter, int yCenter,
                      int xMag,     int yMag);
```

Parameter	Description
pBM	Pointer to the bitmap to display.
x0	X-position of the anchor point in the display.
y0	Y-position of the anchor point in the display.
xCenter	X-position of the anchor point in the bitmap. Specifies the bitmap pixel of the bitmap which should be displayed at x0 on the screen independent of scaling or mirroring.
yCenter	Y-position of the anchor point in the bitmap. Specifies the bitmap pixel of the bitmap which should be displayed at y0 on the screen independent of scaling or mirroring.
xMag	Scale factor of X-direction in the unit 1/1000. A negative value mirrors the x-axis.
yMag	Scale factor of Y-direction in the unit 1/1000. A negative value mirrors the y-axis.

Table 8.30: GUI_DrawBitmapEx() parameter list

Additional information

This function can not be used to draw RLE-compressed bitmaps.

GUI_DrawBitmapHWAlpha()

Description

Draws a bitmap with alpha information on a MultiLayer system with hardware alpha blending support.

Prototype

```
void GUI_DrawBitmapHWAlpha(const GUI_BITMAP * pBM, int x0, int y0);
```

Parameter	Description
pBM	Pointer to the bitmap to display.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 8.31: GUI_DrawBitmapHWAlpha() parameter list

Additional information

In emWin logical colors are handled as 32 bit values. The lower 24 bits are used for the color information and the upper 8 bits are used to manage the alpha value. An alpha value of 0 means the image is opaque and a value of 0xFF means completely transparent (invisible).
On systems with hardware support for alpha blending the alpha values need to be written to the display controller which does the alpha blending.
Normally the alpha format of the hardware is not the same as the alpha definition in emWin described above. Mostly a value of 0 means fully transparent and higher values means the pixel becomes more visible.
Because of this in the most cases custom color conversion routines are required to translate a logical color to the required hardware format. The `Sample` folder contains the example `ALPHA_DrawBitmapHWAlpha` which shows how to consider the requirement of custom color conversion.

GUI_DrawBitmapMag()

Description

This routine makes it possible to magnify a bitmap on the display.

Prototype

```
void GUI_DrawBitmapMag(const GUI_BITMAP * pBM,
                       int x0,   int y0,
                       int XMul, int YMul);
```

Parameter	Description
`pBM`	Pointer to the bitmap to display.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.
`XMul`	Magnification factor of X-direction.
`YMul`	Magnification factor of Y-direction.

Table 8.32: GUI_DrawBitmapMag() parameter list

GUI_SetAlphaMask8888()

Description

That routine takes only effect on drawing non compressed true color bitmaps (`GUI_DRAW_BMP8888` and `GUI_DRAW_BMPM8888I`). It can be used to set additional masks used for drawing the pixels of a non compressed true color bitmap.

Prototype

```
void GUI_SetAlphaMask8888(U32 OrMask, U32 AndMask);
```

Parameter	Description
`OrMask`	Value to be OR combined with each pixel.
`AndMask`	Value to be AND combined with each pixel.

Table 8.33: GUI_SetAlphaMask8888() parameter list

8.1.5 Drawing streamed bitmaps

Streamed bitmaps can be located in addressable area (RAM or ROM) as well as external memory (e.g. on removable devices).

Drawing from addressable memory

There are 2 possibilities to display streamed bitmaps which are located on addressable memory. The first one is to use the function GUI_DrawStreamedBitmap() or the function GUI_DrawStreamedBitmapAuto(). The second one is to create a GUI_BITMAP according to the streamed bitmap and use it for a regular call of e.g. GUI_DrawBitmap().

Drawing from external memory

Streamed bitmaps which are located on external memory can be drawn using the ...Ex() functions. ...Ex() functions require a pointer to a user defined GetData() function (see “Getting data with the ...Ex() functions” on page 187) in order to have emWin retrieve the stream self-dependently. If the format of the streamed bitmap is unknown at run-time, the function GUI_DrawStreamedBitmapExAuto() should be used.

Requirements

The ...Ex() functions require to have enough free memory which is assigned to emWin to store at least one line of pixel data. If there is not enough free memory, the function will return immediately without having anything drawn.
Using the ...Auto() function causes the linker to add all functions referenced by the ...Auto() function. If there is not enough memory the according function for the specific format should be used (e.g. GUI_DrawStreamedBitmap565Ex()).

Available bitmap formats

The following table shows the currently supported formats and the availability of according ...Ex() functions:

Format	Description	...Ex() function available
IDX	Index based* bitmaps 1-8bpp.	Yes
444_12	12bpp Bitmaps, 4 bits blue, 4 bits green, 4 bits red.	Yes
444_12_1	12bpp Bitmaps, 4 bits blue, 4 bits green, 4 bits red.	Yes
444_16	12bpp Bitmaps, 4 bits blue, 4 bits green, 4 bits red.	Yes
M444_16	12bpp Bitmaps, 4 bits red, 4 bits green, 4 bits blue.	Yes
M444_12	12bpp Bitmaps, 4 bits red, 4 bits green, 4 bits blue.	Yes
M444_12_1	12bpp Bitmaps, 4 bits red, 4 bits green, 4 bits blue.	Yes
555	16bpp high color bitmaps, 5 bits blue, 5 bits green, 5 bits red.	Yes
M555	16bpp high color bitmaps, 5 bits red, 5 bits green, 5 bits blue.	Yes
565	16bpp high color bitmaps, 5 bits blue, 6 bits green, 5 bits red.	Yes
M565	16bpp high color bitmaps, 5 bits red, 6 bits green, 5 bits blue.	Yes
A555	16bpp high color bitmaps, 5 bits blue, 5 bits green, 5 bits red, 8 bit alpha channel	Yes
AM555	16bpp high color bitmaps, 5 bits red, 5 bits green, 5 bits blue, 8 bit alpha channel	Yes
A565	16bpp high color bitmaps, 5 bits blue, 6 bits green, 5 bits red, 8 bit alpha channel	Yes
AM565	16bpp high color bitmaps, 5 bits red, 6 bits green, 5 bits blue, 8 bit alpha channel	Yes
24	24bpp true color bitmaps, 8 bits blue, 8 bits green, 8 bits red.	Yes
Alpha	32bpp true color bitmaps, 8 bits alpha, 8 bits blue, 8 bits green, 8 bits red.	No
RLEAlpha	8bpp alpha channel bitmaps, compressed.	No
RLE4	4bpp index based bitmaps, RLE compressed.	Yes
RLE8	8bpp index based bitmaps, RLE compressed.	Yes
RLE16	16bpp (565) high color bitmaps, RLE compressed.	Yes
RLEM16	16bpp (M565) high color bitmaps, RLE compressed.	Yes
RLE32	32bpp (8888) true color bitmaps with alpha channel, RLE compressed.	Yes

Table 8.34: Streamed bitmap format description table

* Index based bitmaps consist of a palette of colors stated as 32bit values. All other bitmaps do not have a palette and therefore have the bitmap data stored in the format specified in the table.

GUI_CreateBitmapFromStream()

Description

The function creates a bitmap structure by passing any type of bitmap stream.

Prototype

```
int GUI_CreateBitmapFromStream(GUI_BITMAP      * pBMP,
                               GUI_LOGPALETTE * pPAL,
                               const void     * p);
```

Parameter	Description
pBMP	Pointer to a GUI_BITMAP structure to be initialized by the function.
pPAL	Pointer to a GUI_LOGPALETTE structure to be initialized by the function.
p	Pointer to the data stream.

Table 8.35: GUI_CreateBitmapFromStream() parameter list

Return value

0 on success, 1 on error.

Additional information

This function should be used if the data stream can consist of several kinds of bitmap formats or unknown. Disadvantage of using this function is that it has a significant memory footprint. If memory usage (ROM) is a concern, it may be better to use the format specific functions below.

Example

The following example shows how the GUI_CreateBitmapFromStream() - functions can be used to create and draw a bitmap:

```
void DrawBitmap(const void * pData, int xPos, int yPos) {
  GUI_BITMAP     Bitmap;
  GUI_LOGPALETTE Palette;

  GUI_CreateBitmapFromStream(&Bitmap, &Palette, pData);
  GUI_DrawBitmap(&Bitmap, xPos, yPos);
}
```


GUI_CreateBitmapFromStreamIDX()

GUI_CreateBitmapFromStreamRLE4()

GUI_CreateBitmapFromStreamRLE8()

GUI_CreateBitmapFromStream444_12()

GUI_CreateBitmapFromStream444_12_1()

GUI_CreateBitmapFromStream444_16()

GUI_CreateBitmapFromStreamM444_12()

GUI_CreateBitmapFromStreamM444_12_1()

GUI_CreateBitmapFromStreamM444_16()

GUI_CreateBitmapFromStreamA555()

GUI_CreateBitmapFromStreamAM555()

GUI_CreateBitmapFromStreamA565()

GUI_CreateBitmapFromStreamAM565()

GUI_CreateBitmapFromStream565()

GUI_CreateBitmapFromStreamM565()

GUI_CreateBitmapFromStream555()

GUI_CreateBitmapFromStreamM555()

GUI_CreateBitmapFromStreamRLE16()

GUI_CreateBitmapFromStreamRLEM16()

GUI_CreateBitmapFromStream24()

GUI_CreateBitmapFromStreamAlpha()

GUI_CreateBitmapFromStreamRLEAlpha()

GUI_CreateBitmapFromStreamRLE32()

Description

These functions create bitmap structures by passing bitmap streams of a known format.

Prototype

```
int  GUI_CreateBitmapFromStream<FORMAT>(GUI_BITMAP     * pBMP,
                                        GUI_LOGPALETTE * pPAL,
```

```
const void * p);
```

Parameter	Description
pBMP	Pointer to a GUI_BITMAP structure to be initialized by the function.
pPAL	Pointer to a GUI_LOGPALETTE structure to be initialized by the function.
p	Pointer to the data stream.

Table 8.36: GUI_CreateBitmapFromStream...() parameter list

Supported data stream formats

The following table shows the supported data stream formats for each function:

Function	Supported stream format
GUI_CreateBitmapFromStreamIDX()	Streams of index based bitmaps.
GUI_CreateBitmapFromStreamRLE4()	Streams of RLE4 compressed bitmaps.
GUI_CreateBitmapFromStreamRLE8()	Streams of RLE8 compressed bitmaps.
GUI_CreateBitmapFromStreamA555()	Streams of high color bitmaps with alpha channel (A555).
GUI_CreateBitmapFromStreamAM555()	Streams of high color bitmaps with alpha channel (AM555).
GUI_CreateBitmapFromStreamA565()	Streams of high color bitmaps with alpha channel (A565).
GUI_CreateBitmapFromStreamAM565()	Streams of high color bitmaps with alpha channel (AM565).
GUI_CreateBitmapFromStream444_12()	Streams 12bpp bitmaps (444_12).
GUI_CreateBitmapFromStream444_12_1()	Streams 12bpp bitmaps (444_12_1).
GUI_CreateBitmapFromStream444_16()	Streams 12bpp bitmaps (444_16).
GUI_CreateBitmapFromStreamM444_12()	Streams 12bpp bitmaps (M444_12).
GUI_CreateBitmapFromStreamM444_12_1()	Streams 12bpp bitmaps (M444_12_1).
GUI_CreateBitmapFromStreamM444_16()	Streams 12bpp bitmaps (M444_16).
GUI_CreateBitmapFromStream565()	Streams of high color bitmaps (565).
GUI_CreateBitmapFromStreamM565()	Streams of high color bitmaps (M565).
GUI_CreateBitmapFromStream555()	Streams of high color bitmaps (555).
GUI_CreateBitmapFromStreamM555()	Streams of high color bitmaps (M565).
GUI_CreateBitmapFromStreamRLE16()	Streams of RLE16 compressed bitmaps.
GUI_CreateBitmapFromStreamRLEM16()	Streams of RLE16 compressed bitmaps, red and blue swapped.
GUI_CreateBitmapFromStream24()	Streams of 24bpp bitmaps (true color).
GUI_CreateBitmapFromStreamAlpha()	Streams of 32bpp bitmaps (true color with alpha channel).
GUI_CreateBitmapFromStreamRLEAlpha()	Streams of RLE compressed 8bpp alpha bitmaps.
GUI_CreateBitmapFromStreamRLE32()	Streams of RLE32 compressed bitmaps (true color with alpha channel).

Table 8.37: Supported stream formats

Return value

0 on success, 1 on error.

Additional information

These functions should be used if the data stream consists of a known format. This avoids linking of unused code and keeps the binary code small.

GUI_DrawStreamedBitmap()

Description

Draws a bitmap from an indexed based bitmap data stream.

Prototype

```
void GUI_DrawStreamedBitmap(const void * p, int x, int y);
```

Parameter	Description
p	Pointer to the data stream.
x	X-position of the upper left corner of the bitmap in the display.
y	Y-position of the upper left corner of the bitmap in the display.

Table 8.38: GUI_DrawStreamedBitmap() parameter list

Additional information

The Bitmap Converter can be used to create bitmap data streams. The format of these streams does not equal the format of a `bmp` file. Details can be found in the chapter “Bitmap Converter” on page 189.

GUI_DrawStreamedBitmapAuto()

Description

Draws a bitmap from a bitmap data stream of any supported format.

Prototype

```
void GUI_DrawStreamedBitmapAuto(const void * p, int x, int y);
```

Parameter	Description
p	Pointer to the data stream.
x	X-position of the upper left corner of the bitmap in the display.
y	Y-position of the upper left corner of the bitmap in the display.

Table 8.39: GUI_DrawStreamedBitmapAuto() parameter list

Additional information

Additional information can be found unter “GUI_DrawStreamedBitmap()” on page 140.

GUI_DrawStreamedBitmapEx()

Description

This function can be used for drawing index based bitmap data streams if not enough RAM or ROM is available to keep the whole file within the addressable memory (RAM or ROM). The GUI library calls the function pointed by the parameter `pfGetData` to read the data. This `GetData` function needs to return the number of read bytes.

Prototype

```
int GUI_DrawStreamedBitmapEx(GUI_GET_DATA_FUNC * pfGetData,
```



```
const void * p, int x, int y);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to “Getting data with the ...Ex() functions” on page 187.
p	Void pointer passed to the function pointed by pfGetData.
x	X-position of the upper left corner of the bitmap in the display.
y	Y-position of the upper left corner of the bitmap in the display.

Table 8.40: GUI_DrawStreamedBitmapEx() parameter list

Return value

0 on success, 1 on error.

Additional information

The function requires at least memory for one line of bitmap data.
For more details please also refer to the function GUI_SetStreamedBitmapHook().

GUI_DrawStreamedBitmapExAuto()

Description

This function can be used for drawing bitmap data streams of any supported format if not enough RAM or ROM is available to keep the whole file within the addressable memory (RAM or ROM). The GUI library calls the function pointed by the parameter pfGetData to read the data. This GetData function needs to return the number of read bytes.

Prototype

```
int GUI_DrawStreamedBitmapExAuto(GUI_GET_DATA_FUNC * pfGetData,
                                 const void * p, int x, int y);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to “Getting data with the ...Ex() functions” on page 187.
p	Void pointer passed to the function pointed by pfGetData.
x	X-position of the upper left corner of the bitmap in the display.
y	Y-position of the upper left corner of the bitmap in the display.

Table 8.41: GUI_DrawStreamedBitmapExAuto() parameter list

Return value

0 on success, 1 on error.

Additional information

The function requires at least memory for one line of bitmap data.

GUI_DrawStreamedBitmapA555Ex()

GUI_DrawStreamedBitmapAM555Ex()

GUI_DrawStreamedBitmapA565Ex()

GUI_DrawStreamedBitmapAM565Ex()

GUI_DrawStreamedBitmap555Ex()

GUI_DrawStreamedBitmapM555Ex()

GUI_DrawStreamedBitmap565Ex()

GUI_DrawStreamedBitmapM565Ex()

GUI_DrawStreamedBitmap24Ex()

Description

This function can be used for drawing bitmap data streams of the respective format if not enough RAM or ROM is available to keep the whole file within the addressable memory (RAM or ROM). The GUI library calls the function pointed by the parameter `pfGetData` to read the data. This `GetData` function needs to return the number of read bytes.

Prototype

```
int GUI_DrawStreamedBitmap<XXX>Ex(GUI_GET_DATA_FUNC * pfGetData,
                                  const void * p, int x, int y);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by `pfGetData`.
x	X-position of the upper left corner of the bitmap in the display.
y	Y-position of the upper left corner of the bitmap in the display.

Table 8.42: GUI_DrawStreamedBitmap..() parameter list

Return value

0 on success, 1 on error.

Additional information

The functions require at least memory for one line of bitmap data.

GUI_GetStreamedBitmapInfo()

Description

Returns a structure with information about the given data stream.

Prototype

```
void GUI_GetStreamedBitmapInfo(const void             * p,
                               GUI_BITMAPSTREAM_INFO * pInfo);
```

Parameter	Description
p	Pointer to the data stream.
pInfo	Pointer to a `GUI_BITMAPSTREAM_INFO` structure to be filled by the function.

Table 8.43: GUI_GetStreamedBitmapInfo() parameter list

Elements of structure GUI_BITMAPSTREAM_INFO

Data type	Element	Description
int	XSize	Pixel size in X of the image.
int	YSize	Pixel size in Y of the image.
int	BitsPerPixel	Number of bits per pixel.
int	NumColors	Number of colors in case of an index based image.
int	HasTrans	In case of an index based image 1 if transparency exist, 0 if not.

Table 8.44: GUI_BITMAPSTREAM_INFO element list

GUI_GetStreamedBitmapInfoEx()

Description

Returns a structure with information about the given data stream which does not need to be located in the addressable ROM or RAM area of the CPU.

Prototype

```
int GUI_GetStreamedBitmapInfoEx(GUI_GET_DATA_FUNC      * pfGetData,
                                const void             * p,
                                GUI_BITMAPSTREAM_INFO * pInfo);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a GUI_BITMAPSTREAM_INFO structure to be filled by the function.

Table 8.45: GUI_GetStreamedBitmapInfoEx() parameter list

Return value

0 on success, 1 on error.

GUI_BITMAPSTREAM_INFO

The elements of the structure GUI_BITMAPSTREAM_INFO are listed under GUI_GetStreamedBitmapInfo().

GUI_SetStreamedBitmapHook()

Description

Sets a hook function to be able to manipulate the palette of a streamed bitmap which is not located in the addressable area of the CPU. The hook function is called when executing GUI_DrawStreamedBitmapEx().

Prototype

```
void GUI_SetStreamedBitmapHook(
                          GUI_BITMAPSTREAM_CALLBACK pfStreamedBitmapHook);
```

Parameter	Description
pfStreamedBitmapHook	Hook function to be called by GUI_DrawStreamedBitmapEx().

Table 8.46: GUI_SetStreamedBitmapHook() parameter list

Prototype of hook function

```
void * Hook(GUI_BITMAPSTREAM_PARAM * pParam);
```

Parameter	Description
pParam	Pointer to a GUI_BITMAPSTREAM_PARAM structure

Table 8.47: Hook function parameter list

Elements of structure GUI_BITMAPSTREAM_PARAM

Data type	Element	Description
int	Cmd	Command to be executed.
U32	v	Depends on the command to be executed.
void *	p	Depends on the command to be executed.

Table 8.48: GUI_BITMAPSTREAM_PARAM element list

Supported values for parameter Cmd	
GUI_BITMAPSTREAM_GET_BUFFER	When receiving this command the application can spend a buffer for the palette of a bitmap stream. Parameters: p - Pointer to the buffer or NULL v - Requested buffer size
GUI_BITMAPSTREAM_RELEASE_BUFFER	If the application has spend a buffer for the palette here the buffer should be released. Parameters: p - Pointer to buffer to be released v - not used
GUI_BITMAPSTREAM_MODIFY_PALETTE	This command is sent after loading the palette and before drawing the image to be able to modify the palette of the streamed image. Parameters: p - Pointer to palette data v - Number of colors in palette

Example

```
static void * _cbStreamedBitmapHook(GUI_BITMAPSTREAM_PARAM * pParam) {
  void * p = NULL;
  int    i, NumColors;
  U32    Color;
  U32  * pColor;

  switch (pParam->Cmd) {
  case GUI_BITMAPSTREAM_GET_BUFFER:
    //
    // Allocate buffer for palette data
    //
    p = malloc(pParam->v);
    break;
  case GUI_BITMAPSTREAM_RELEASE_BUFFER:
    //
    // Release buffer
    //
    free(pParam->p);
    break;
  case GUI_BITMAPSTREAM_MODIFY_PALETTE:
    //
    // Do something with the palette...
    //
    NumColors = pParam->v;
    pColor    = (U32 *)pParam->p;
    Color     = *(pColor + pParam->v - 1);
    for (i = NumColors - 2; i >= 0; i--) {
      *(pColor + i + 1) = *(pColor + i);
    }
    *pColor = Color;
    break;
  }
  return p;
}
```

8.1.6 Drawing lines

The most frequently used drawing routines are those that draw a line from one point to another.

GUI_DrawHLine()

Description

Draws a horizontal line one pixel thick from a specified starting point to a specified endpoint in the current window.

Prototype

```
void GUI_DrawHLine(int y, int x0, int x1);
```

Parameter	Description
y	Y-position.
x0	X-starting position.
x1	X-end position.

Table 8.49: GUI_DrawHLine() parameter list

Additional information

If x1 < x0, nothing will be displayed.
With most LCD controllers, this routine is executed very quickly because multiple pixels can be set at once and no calculations are needed. If it is clear that horizontal lines are to be drawn, this routine executes faster than the GUI_DrawLine() routine.

GUI_DrawLine()

Description

Draws a line from a specified starting point to a specified endpoint in the current window (absolute coordinates).

Prototype

```
void GUI_DrawLine(int x0, int y0, int x1, int y1);
```

Parameter	Description
x0	X-starting position.
y0	Y-starting position.
x1	X-end position.
y1	Y-end position.

Table 8.50: GUI_DrawLine() parameter list

Additional information

If part of the line is not visible because it is not in the current window or because part of the current window is not visible, this is due to clipping.

GUI_DrawLineRel()

Description

Draws a line from the current (x, y) position to an endpoint specified by X-distance and Y-distance in the current window (relative coordinates).

Prototype

```
void GUI_DrawLineRel(int dx, int dy);
```

Parameter	Description
dx	Distance in X-direction to end of line to draw.
dy	Distance in Y-direction to end of line to draw.

Table 8.51: GUI_DrawLineRel() parameter list

GUI_DrawLineTo()

Description

Draws a line from the current (X,Y) position to an endpoint specified by X- and Y-coordinates in the current window.

Prototype

```
void GUI_DrawLineTo(int x, int y);
```

Parameter	Description
x	X-end position.
y	Y-end position.

Table 8.52: GUI_DrawLineTo() parameter list

GUI_DrawPolyLine()

Description

Connects a predefined list of points with lines in the current window.

Prototype

```
void GUI_DrawPolyLine(const GUI_POINT * pPoint, int NumPoints,
                      int x, int y);
```

Parameter	Description
pPoint	Pointer to the polyline to display.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

Table 8.53: GUI_DrawPolyLine() parameter list

Additional information

The starting point and endpoint of the polyline need not be identical.

GUI_DrawVLine()

Description

Draws a vertical line one pixel thick from a specified starting point to a specified end-point in the current window.

Prototype

```
void GUI_DrawVLine(int x, int y0, int y1);
```

Parameter	Description
x	X-position.
y0	Y-starting position.
y1	Y-end position.

Table 8.54: GUI_DrawVLine() parameter list

Additional information

If `y1` < `y0`, nothing will be displayed.
With most LCD controllers, this routine is executed very quickly because multiple pixels can be set at once and no calculations are needed. If it is clear that vertical lines are to be drawn, this routine executes faster than the `GUI_DrawLine()` routine.

GUI_GetLineStyle()

Description

Returns the current line style used by the function GUI_DrawLine.

Prototype

```
U8 GUI_GetLineStyle(void);
```

Return value

Current line style used by the function GUI_DrawLine.

GUI_MoveRel()

Description

Moves the current line pointer relative to its current position.

Prototype

```
void GUI_MoveRel(int dx, int dy);
```

Parameter	Description
dx	Distance to move in X.
dy	Distance to move in Y.

Table 8.55: GUI_MoveRel() parameter list

Related topics

GUI_DrawLineTo(), GUI_MoveTo()

GUI_MoveTo()

Description

Moves the current line pointer to the given position.

Prototype

```
void GUI_MoveTo(int x, int y);
```

Parameter	Description
x	New position in X.
y	New position in Y.

Table 8.56: GUI_MoveTo() parameter list

GUI_SetLineStyle()

Description

Sets the current line style used by the function GUI_DrawLine.

Prototype

```
U8 GUI_SetLineStyle(U8 LineStyle);
```

Parameter	Description
LineStyle	New line style to be used. See table below.

Table 8.57: GUI_SetLineStyle() parameter list

Permitted values for parameter LineStyle	
GUI_LS_SOLID	Lines would be drawn solid (default).
GUI_LS_DASH	Lines would be drawn dashed.

Permitted values for parameter LineStyle	
GUI_LS_DOT	Lines would be drawn dotted.
GUI_LS_DASHDOT	Lines would be drawn alternating with dashes and dots.
GUI_LS_DASHDOTDOT	Lines would be drawn alternating with dashes and double dots.

Return value

Previous line style used by the function GUI_DrawLine.

Additional information

This function sets only the line style used by GUI_DrawLine. The style will be used only with a pen size of 1.

8.1.7 Drawing polygons

The polygon drawing routines can be helpful when drawing vectorized symbols.

GUI_DrawPolygon()

Description

Draws the outline of a polygon defined by a list of points in the current window.

Prototype

```
void GUI_DrawPolygon(const GUI_POINT * pPoint, int NumPoints,
                     int x, int y);
```

Parameter	Description
pPoint	Pointer to the polygon to display.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

Table 8.58: GUI_DrawPolygon() parameter list

Additional information

The polyline drawn is automatically closed by connecting the endpoint to the starting point.

GUI_EnlargePolygon()

Description

Enlarges a polygon on all sides by a specified length in pixels.

Prototype

```
void GUI_EnlargePolygon(GUI_POINT       * pDest,
                        const GUI_POINT * pSrc,
                        int               NumPoints,
                        int               Len);
```

Parameter	Description
pDest	Pointer to the destination polygon.
pSrc	Pointer to the source polygon.
NumPoints	Number of points specified in the list of points.
Len	Length (in pixels) by which to enlarge the polygon.

Table 8.59: GUI_EnlargePolygon() parameter list

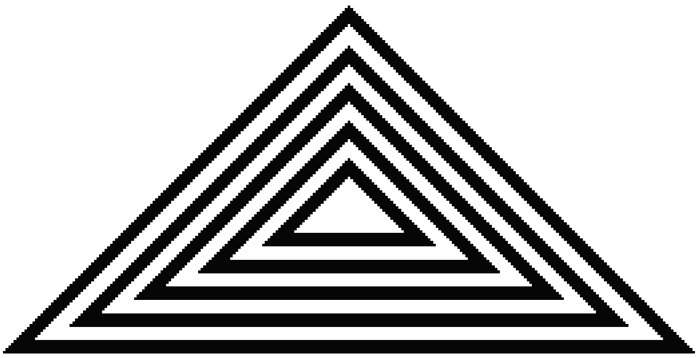
Additional information

Make sure the destination array of points is equal to or larger than the source array.

Example

```
const GUI_POINT aPoints[] = {
  { 40, 20},
  {  0, 20},
  { 20,  0}
};

GUI_POINT aEnlargedPoints[GUI_COUNTOF(aPoints)];

void Sample(void) {
  int i;
  GUI_Clear();
  GUI_SetDrawMode(GUI_DM_XOR);
  GUI_FillPolygon(aPoints, GUI_COUNTOF(aPoints), 140, 110);
  for (i = 1; i < 10; i++) {
    GUI_EnlargePolygon(aEnlargedPoints, aPoints, GUI_COUNTOF(aPoints), i * 5);
    GUI_FillPolygon(aEnlargedPoints, GUI_COUNTOF(aPoints), 140, 110);
  }
}
```

Screenshot of above example



GUI_FillPolygon()

Description

Draws a filled polygon defined by a list of points in the current window.

Prototype

```
void GUI_FillPolygon(const GUI_POINT * pPoint, int NumPoints, int x, int y);
```

Parameter	Description
pPoint	Pointer to the polygon to display and to fill.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

Table 8.60: GUI_FillPolygon() parameter list

Additional information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. It is not required that the endpoint touches the outline of the polygon.
Rendering a polygon is done by drawing one or more horizontal lines for each y-position of the polygon. Per default the maximum number of points used to draw the horizontal lines for one y-position is 12 (which means 6 lines per y-position). If this value needs to be increased, the macro `GUI_FP_MAXCOUNT` can be used to set the maximum number of points.

Example

```
#define GUI_FP_MAXCOUNT 50
```


GUI_MagnifyPolygon()

Description

Magnifies a polygon by a specified factor.

Prototype

```
void GUI_MagnifyPolygon(GUI_POINT *         pDest,
                        const GUI_POINT * pSrc,
                        int               NumPoints,
                        int               Mag);
```

Parameter	Description
pDest	Pointer to the destination polygon.
pSrc	Pointer to the source polygon.
NumPoints	Number of points specified in the list of points.
Mag	Factor used to magnify the polygon.

Table 8.61: GUI_MagnifyPolygon() parameter list

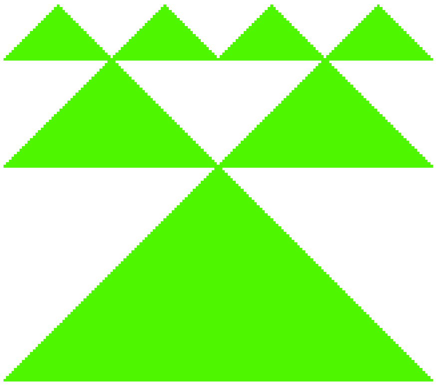
Additional information

Make sure the destination array of points is equal to or larger than the source array. Note the difference between enlarging and magnifying a polygon. Calling the function GUI_EnlargePolygon() with the parameter Len = 1 will enlarge the polygon by one pixel on all sides, whereas the call of GUI_MagnifyPolygon() with the parameter Mag = 1 will have no effect.

Example

```
const GUI_POINT aPoints[] = {
  {  0, 20},
  { 40, 20},
  { 20,  0}
};

GUI_POINT aMagnifiedPoints[GUI_COUNTOF(aPoints)];

void Sample(void) {
  int Mag, y = 0, Count = 4;
  GUI_Clear();
  GUI_SetColor(GUI_GREEN);
  for (Mag = 1; Mag <= 4; Mag *= 2, Count /= 2) {
    int i, x = 0;
    GUI_MagnifyPolygon(aMagnifiedPoints, aPoints, GUI_COUNTOF(aPoints), Mag);
    for (i = Count; i > 0; i--, x += 40 * Mag) {
      GUI_FillPolygon(aMagnifiedPoints, GUI_COUNTOF(aPoints), x, y);
    }
    y += 20 * Mag;
  }
}
```

Screenshot of above example



GUI_RotatePolygon()

Description

Rotates a polygon by a specified angle.

Prototype

```
void GUI_RotatePolygon(GUI_POINT *       pDest,
                       const GUI_POINT * pSrc,
                       int               NumPoints,
                       float             Angle);
```

Parameter	Description
pDest	Pointer to the destination polygon.
pSrc	Pointer to the source polygon.
NumPoints	Number of points specified in the list of points.
Angle	Angle in radian used to rotate the polygon.

Table 8.62: GUI_RotatePolygon() parameter list

Additional information

Make sure the destination array of points is equal to or larger than the source array.

Example

The following example shows how to draw a polygon. It is available as `2DGL_DrawPolygon.c` in the examples shipped with emWin.

```
#include "gui.h"
/*******************************************************************
*
*            The points of the arrow
*/
static const GUI_POINT aPointArrow[] = {
  {  0,  -5},
  {-40, -35},
  {-10, -25},
  {-10, -85},
  { 10, -85},
  { 10, -25},
  { 40, -35},
};

/*******************************************************************
*
*                Draws a polygon
*/
static void DrawPolygon(void) {
  int Cnt =0;
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetFont(&GUI_Font8x16);
  GUI_SetColor(0x0);
  GUI_DispStringAt("Polygons of arbitrary shape ", 0, 0);
```



```
  GUI_DispStringAt("in any color", 120, 20);
  GUI_SetColor(GUI_BLUE);
  /* Draw filled polygon */
  GUI_FillPolygon (&aPointArrow[0],7,100,100);
}

/*******************************************************************
*
*                  main
*/
void main(void) {
  GUI_Init();
  DrawPolygon();
  while(1)
    GUI_Delay(100);
}
```

Screenshot of above example

8.1.8 Drawing circles

GUI_DrawCircle()

Description

Draws the outline of a circle of specified dimensions, at a specified position in the current window.

Prototype

```
void GUI_DrawCircle(int x0, int y0, int r);
```

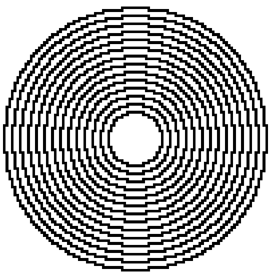
Parameter	Description
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
r	Radius of the circle (half the diameter). Must be a positive value.

Table 8.63: GUI_DrawCircle() parameter list

Example

```
for (i = 10; i < 50; i += 3) {
  GUI_DrawCircle(120, 60, i);
}
```

Screenshot of above example



GUI_FillCircle()

Description

Draws a filled circle of specified dimensions at a specified position in the current window.

Prototype

```
void GUI_FillCircle(int x0, int y0, int r);
```

Parameter	Description
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
r	Radius of the circle (half the diameter). Must be a positive value.

Table 8.64: GUI_FillCircle() parameter list

Example

```
GUI_FillCircle(120,60,50);
```

Screenshot of above example

8.1.9 Drawing ellipses

GUI_DrawEllipse()

Description

Draws the outline of an ellipse of specified dimensions, at a specified position in the current window.

Prototype

```
void GUI_DrawEllipse(int x0, int y0, int rx, int ry);
```

Parameter	Description
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
rx	X-radius of the ellipse (half the diameter). Must be a positive value.
ry	Y-radius of the ellipse (half the diameter). Must be a positive value.

Table 8.65: GUI_DrawEllipse() parameter list

Example

See the `GUI_FillEllipse()` example.

GUI_FillEllipse()

Description

Draws a filled ellipse of specified dimensions at a specified position in the current window.

Prototype

```
void GUI_FillEllipse(int x0, int y0, int rx, int ry);
```

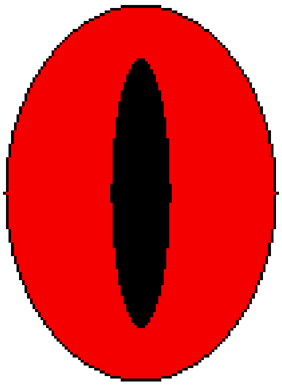
Parameter	Description
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
rx	X-radius of the ellipse (half the diameter). Must be a positive value.
ry	Y-radius of the ellipse (half the diameter). Must be a positive value.

Table 8.66: GUI_FillEllipse() parameter list

Example

```
// Demo ellipses
GUI_SetColor(0xff);
GUI_FillEllipse(100, 180, 50, 70);
GUI_SetColor(0x0);
GUI_DrawEllipse(100, 180, 50, 70);
GUI_SetColor(0x000000);
GUI_FillEllipse(100, 180, 10, 50);
```

Screenshot of above example



8.1.10 Drawing arcs

GUI_DrawArc()

Description

Draws an arc of specified dimensions at a specified position in the current window. An arc is a section of the outline of a circle.

Prototype

```
void GUI_DrawArc(int xCenter, int yCenter, int rx, int ry, int a0, int a1);
```

Parameter	Description
xCenter	Horizontal position of the center in pixels of the client window.
yCenter	Vertical position of the center in pixels of the client window.
rx	X-radius (pixels).
ry	Y-radius (pixels).
a0	Starting angle (degrees).
a1	Ending angle (degrees).

Table 8.67: GUI_DrawArc() parameter list

Limitations

Currently the ry parameter is not used. The rx parameter is used instead.

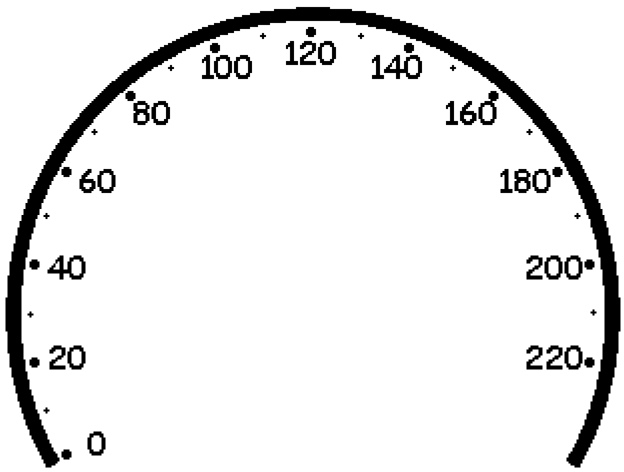
Example

```
void DrawArcScale(void) {
  int x0 = 160;
```



```
  int y0 = 180;
  int i;
  char ac[4];
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetPenSize( 5 );
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_SetFont(&GUI_FontComic18B_ASCII);
  GUI_SetColor( GUI_BLACK );
  GUI_DrawArc( x0,y0,150, 150,-30, 210 );
  GUI_Delay(1000);
  for (i=0; i<= 23; i++) {
    float a = (-30+i*10)*3.1415926/180;
    int x = -141*cos(a)+x0;
    int y = -141*sin(a)+y0;
    if (i%2 == 0)
      GUI_SetPenSize( 5 );
    else
      GUI_SetPenSize( 4 );
    GUI_DrawPoint(x,y);
    if (i%2 == 0) {
      x = -123*cos(a)+x0;
      y = -130*sin(a)+y0;
      sprintf(ac, "%d", 10*i);
      GUI_SetTextAlign(GUI_TA_VCENTER);
      GUI_DispStringHCenterAt(ac,x,y);
    }
  }
}
```

Screenshot of above example



8.1.11 Drawing graphs

GUI_DrawGraph()

Description

Draws a graph at once.

Prototype

```
void GUI_DrawGraph(I16 * paY, int NumPoints, int x0, int y0);
```

Parameter	Description
paY	Pointer to an array containing the Y-values of the graph.
NumPoints	Number of Y-values to be displayed.
x0	Starting point in x.
y0	Starting point in y.

Table 8.68: GUI_DrawGraph() parameter list

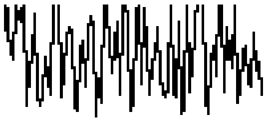
Additional information

The function first sets the line-cursor to the position specified with x0, y0 and the first Y-value of the given array. Then it starts drawing lines to x0 + 1, y0 + *(paY + 1), x0 + 2, y0 + *(paY + 2) and so on.

Example

```
#include "GUI.h"
#include <stdlib.h>

I16 aY[100];

void MainTask(void) {
  int i;
  GUI_Init();
  for (i = 0; i < GUI_COUNTOF(aY); i++) {
    aY[i] = rand() % 50;
  }
  GUI_DrawGraph(aY, GUI_COUNTOF(aY), 0, 0);
}
```

Screenshot of above example



8.1.12 Drawing pie charts

GUI_DrawPie()

Description

Draws a circle sector.

Prototype

```
void GUI_DrawPie(int x0, int y0, int r, int a0, int a1, int Type);
```

Parameter	Description
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
r	Radius of the circle (half the diameter).
a0	Starting angle (degrees).
a1	End angle (degrees).
Type	(reserved for future use, should be 0)

Table 8.69: GUI_DrawPie() parameter list

Example

```
int i, a0, a1;
const unsigned aValues[]  = { 100, 135, 190, 240, 340, 360};
const GUI_COLOR aColors[] = { GUI_BLUE, GUI_GREEN,   GUI_RED,
                              GUI_CYAN, GUI_MAGENTA, GUI_YELLOW };
for (i = 0; i < GUI_COUNTOF(aValues); i++) {
  a0 = (i == 0) ? 0 : aValues[i - 1];
  a1 = aValues[i];
  GUI_SetColor(aColors[i]);
  GUI_DrawPie(100, 100, 50, a0, a1, 0);
}
```


Screenshot of above example

8.1.13 Saving and restoring the GUI-context

GUI_RestoreContext()

Description

The function restores the GUI-context.

Prototype

```
void GUI_RestoreContext(const GUI_CONTEXT * pContext);
```

Parameter	Description
pContext	Pointer to a GUI_CONTEXT structure containing the new context.

Table 8.70: GUI_RestoreContext() parameter list

Additional information

The GUI-context contains the current state of the GUI like the text cursor position, a pointer to the current font and so on. Sometimes it could be useful to save the current state ant to restore it later. For this you can use these functions.

GUI_SaveContext()

Description

The function saves the current GUI-context. (See also GUI_RestoreContext)

Prototype

```
void GUI_SaveContext(GUI_CONTEXT * pContext);
```

Parameter	Description
pContext	Pointer to a GUI_CONTEXT structure for saving the current context.

Table 8.71: GUI_SaveContext() parameter list

8.1.14 Info about screen changes

GUI_DIRTYDEVICE_Create()

Description

A DITRTYDEVICE is an object which makes it possible to monitor the changed area of the screen. In combination with `GUI_DIRTYDEVICE_Fetch()` it makes it possible, to check if the content of the screen has been changed. If changes have been detected the function `GUI_DIRTYDEVICE_Fetch()` returns 1 and fills up an information structure with size and position of the changed area. If nothing has been changed `GUI_DIRTYDEVICE_Fetch()` returns 0.
In case of working with multiple layers the function does not monitor all layers simultaneously. For each layer a separate DIRTYDEVICE needs to be created (if monitoring is required).
Calling `GUI_DIRTYDEVICE_Create()` creates such an monitoring object in the currently selected layer which then automatically monitors all screen drawing operations. If no longer used it can be deleted with `GUI_DIRTYDEVICE_Delete()`.

Prototype

```
int GUI_DIRTYDEVICE_Create(void);
```


Return value

0 on success, 1 on error.

Additional information

A DIRTYDEVICE is also able to return advanced information like a pointer to the first changed pixel, the number of bytes used per pixel and the stride value in pixels from one line of data to the next line. To be able to use those features a linear addressable driver is required. Further the DIRTYDEVICE needs to be created in LCD_X_Config() before the driver of the layer will be created. If advanced information is not required the DIRTYDEVICE can be created and deleted anytime in the application.

GUI_DIRTYDEVICE_CreateEx()

Description

Creates a DIRTYDEVICE in the given layer. For details please refer to the function "GUI_DIRTYDEVICE_Create()" on page 157.

Prototype

```
int GUI_DIRTYDEVICE_CreateEx(int LayerIndex);
```

Parameter	Description
LayerIndex	LayerIndex to be used.

Table 8.72: GUI_DIRTYDEVICE_CreateEx() parameter list

Return value

0 on success, 1 on error.

GUI_DIRTYDEVICE_Delete()

Description

Removes the DIRTYDEVICE of the currently selected layer. If not possible the function returns an error.

Prototype

```
int GUI_DIRTYDEVICE_Delete(void);
```

Return value

0 on success, 1 on error.

GUI_DIRTYDEVICE_DeleteEx()

Description

Removes the DIRTYDEVICE of the given layer. If not possible the function returns an error.

Prototype

```
int GUI_DIRTYDEVICE_DeleteEx(int LayerIndex);
```

Parameter	Description
LayerIndex	LayerIndex to be used.

Table 8.73: GUI_DIRTYDEVICE_DeleteEx() parameter list

Return value

0 on success, 1 on error.

GUI_DIRTYDEVICE_Fetch()

Description

The function fills the given structure with the coordinates and the size of the changed screenarea of the current layer. If no changes have been detected since the last call the function returns 0.

Prototype

```
int GUI_DIRTYDEVICE_Fetch(GUI_DIRTYDEVICE_INFO * pInfo);
```

Parameter	Description
pInfo	Pointer to the GUI_DIRTYDEVICE_INFO-structure which is filled with size and coordinates of the changed area.

Table 8.74: GUI_DIRTYDEVICE_Fetch() parameter list

Elements of structure GUI_DIRTYDEVICE_INFO

Data type	Element	Description
int	x0	Leftmost position of changed area.
int	y0	Topmost position of changed area.
int	xSize	Size in X of changed area.
int	xSize	Size in Y of changed area.
int	BytesPerPixel	Number of bytes per pixel.*1
int	LineOff	Number of pixels (stride) from one line to the next line.*1
void *	pData	Pointer to the first changed pixel.*1

Table 8.75: GUI_DIRTYDEVICE_INFO element list

Note:
1. Only available if the DIRTYDEVICE is created before the driver has been created.

Return value

0 if no changes have been detected, 1 if changes are detected.

GUI_DIRTYDEVICE_FetchEx()

Description

The function fills the given structure with the coordinates and the size of the changed screenarea of the given layer.

Prototype

```
int GUI_DIRTYDEVICE_FetchEx(GUI_DIRTYDEVICE_INFO * pInfo, int LayerIndex);
```

Parameter	Description
pInfo	Pointer to the GUI_DIRTYDEVICE_INFO-structure which is filled with size and coordinates of the changed area.
LayerIndex	Layer index to be used.

Table 8.76: GUI_DIRTYDEVICE_FetchEx() parameter list

Elements of structure GUI_DIRTYDEVICE_INFO

Data type	Element	Description
int	x0	Leftmost position of changed area.
int	y0	Topmost position of changed area.
int	xSize	Size in X of changed area.
int	xSize	Size in Y of changed area.
int	BytesPerPixel	Number of bytes per pixel.*1
int	LineOff	Number of pixels (stride) from one line to the next line.*1
void *	pData	Pointer to the first changed pixel.*1

Table 8.77: GUI_BITMAPSTREAM_INFO element list

Note:
2. Only available if the DIRTYDEVICE is created before the driver has been created.

Return value

0 if no changes have been detected, 1 if changes are detected.

8.1.15 Avoiding tearing effects

Tearing effects could occur if the content of the frame buffer changes during the display controller is not in the vertical non display period and currently updating the screen. A detailed description of tearing effects could be found in Chapter 23.

GUI_SetRefreshHook()

Description

If tearing effects should not occur and the following assumptions are fulfilled that function could be used to avoid those effects:

- Display uses an indirect interface
- Display provides a tearing signal (TE)
- Display communication is fast enough for updating the frame buffer within the vertical non display period.

The function sets a callback function which is called immediately before sending any content to the display controller. That gives the application the chance to wait until the display controller is within the vertical non display period. That could be achieved by polling the TE-pin of the display (if available). The function should return immediately after reaching the non display period. After that the driver sends its (dirty) content to the display controller.

Prototype

```
void GUI_SetRefreshHook(void (* pFunc)(void));
```

Parameter	Description
pFunc	Callback function to be called for waiting until vertical non display period is reached.

Table 8.78: GUI_SetRefreshHook() parameter list

Additional information

When using this function it is important to avoid writing to the display controller for each drawing operation. This can be achieved by the cache locking mechanism. Details can be found in the section “Cache group” on page 1168.

Chapter 9

Displaying bitmap files

The recommended and most efficient way to display a bitmap known at compile time is to use the Bitmap Converter to convert it into a C file and add it to the project / makefile. Details about the Bitmap Converter can be found in the chapter “Bitmap Converter” on page 189.
If the application needs to display images not known at compile time, the image needs to be available in a graphic file format supported by emWin. In this case, the image file can reside in memory or on an other storage device; it can be displayed even if the amount of available memory is less than the size of the image file.
emWin currently supports BMP-, JPEG- and GIF-files. PNG-file support can be achieved by adding the PNG-library available under `www.segger.com/link/emwin_png.zip` which comes with its own BSD style license. More details about PNG support can be found under “PNG file support” on page 183.

9.1 BMP file support

Although bitmaps which can be used with emWin are normally compiled and linked as C files with the application, there may be situations when using these types of structures is not desirable. A typical example would be an application that continuously references new images, such as bitmaps downloaded by the user. The following functions support `bmp` files which have been loaded into memory.
For images that you plan to re-use (that is, a company logo) it is much more efficient to compile and link it as C file which can be used directly by emWin. This may be easily done with the Bitmap Converter.

9.1.1 Supported formats

The BMP file format has been defined by Microsoft. There are a number of different formats as shown in the table below:

Bits per pixel	Indexed	Compression	Supported
1	yes	no	yes
4	yes	no	yes
4	yes	yes	yes
8	yes	no	yes
8	yes	yes	yes
16	no	no	yes
24	no	no	yes
32	no	no	yes

Table 9.1: Supported BMP formats

9.1.2 BMP API

The table below lists the available BMP file related routines in alphabetical order. Detailed function descriptions follow:

Routine	Description
GUI_BMP_Draw()	Draws a BMP file which has been loaded into memory.
GUI_BMP_DrawEx()	Draws a BMP file which needs not to be loaded into memory.
GUI_BMP_DrawScaled()	Draws a BMP file with scaling which has been loaded into memory.
GUI_BMP_DrawScaledEx()	Draws a BMP file with scaling which needs not to be loaded into memory.
GUI_BMP_GetXSize()	Returns the X-size of a BMP file loaded into memory.
GUI_BMP_GetXSizeEx()	Returns the X-size of a BMP file which needs not to be loaded into memory.
GUI_BMP_GetYSize()	Returns the Y-size of a bitmap loaded into memory.
GUI_BMP_GetYSizeEx()	Returns the Y-size of a BMP file which needs not to be loaded into memory.
GUI_BMP_Serialize()	Creates a BMP file.
GUI_BMP_SerializeEx()	Creates a BMP file from the given rectangle.
GUI_BMP_SerializeExBpp()	Creates a BMP file from the given rectangle using the specified color depth.

Table 9.2: BMP API list

GUI_BMP_Draw()

Description

Draws a Windows bmp file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_BMP_Draw(const void * pFileData, int x0, int y0);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the bmp file resides.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.3: GUI_BMP_Draw() parameter list

Additional information

The table at the beginning of the chapter shows the supported BMP file formats. The example 2DGL_DrawBMP.c shows how to use the function.

GUI_BMP_DrawEx()

Description

Draws a bmp file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_BMP_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.4: GUI_BMP_DrawEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

This function is used for drawing bmp files if not enough RAM is available to load the whole file into memory. The GUI library then calls the function pointed by the parameter pfGetData to read the data. The GetData function needs to return the number of requested bytes. The maximum number of bytes requested by the GUI is the number of bytes needed for drawing one line of the image.

GUI_BMP_DrawScaled()

Description

Draws a bmp file, which has been loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int GUI_BMP_DrawScaled(const void * pFileData,
```



```
int x0, int y0, int Num, int Denom);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the bmp file resides.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Table 9.5: GUI_BMP_DrawScaled() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrunk to 2/3 of size the parameter `Num` should be 2 and `Denom` should be 3.

GUI_BMP_DrawScaledEx()

Description

Draws a `bmp` file, which does not have to be loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int  GUI_BMP_DrawScaledEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                          int                 x0,        int    y0,
                          int                 Num,       int    Denom);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by `pfGetData`.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Table 9.6: GUI_BMP_DrawScaledEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrunk to 2/3 of size the parameter `Num` should be 2 and `Denom` should be 3.
For more details, refer to "GUI_BMP_DrawEx()" on page 163.

GUI_BMP_GetXSize()

Description

Returns the X-size of a specified bitmap which has been loaded into memory.

Prototype

```
int GUI_BMP_GetXSize(const void * pFileData);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the bmp file resides.

Table 9.7: GUI_BMP_GetXSize() parameter list

Return value

X-size of the bitmap.

GUI_BMP_GetXSizeEx()

Description

Returns the X-size of a specified bmp file which does not have to be loaded into memory.

Prototype

```
int GUI_BMP_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.

Table 9.8: GUI_BMP_GetXSizeEx() parameter list

Return value

X-size of the bitmap.

GUI_BMP_GetYSize()

Description

Returns the Y-size of a specified bitmap which has been loaded into memory.

Prototype

```
int GUI_BMP_GetYSize(const void * pFileData);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the bmp file resides.

Table 9.9: GUI_BMP_GetYSize() parameter list

Return value

Y-size of the bitmap.

GUI_BMP_GetYSizeEx()

Description

Returns the Y-size of a specified bmp file which does not have to be loaded into memory.

Prototype

```
int GUI_BMP_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.

Table 9.10: GUI_BMP_GetYSizeEx() parameter list

Return value

Y-size of the bitmap.

GUI_BMP_Serialize()

Description

The function creates a BMP file containing the complete content of the LCD. The BMP file is created using the color depth which is used in emWin at a maximum of 24 bpp. In case of using a color depth of less than 8bpp the color depth of the BMP file will be 8bpp.
The currently selected device is used for reading the pixel data. If a Memory Device is selected it's content is written to the file.

Prototype

```
void GUI_BMP_Serialize(GUI_CALLBACK_VOID_U8_P * pfSerialize, void * p);
```

Parameter	Description
pfSerialize	Pointer to serialization function
p	Pointer to user defined data passed to serialization function

Table 9.11: GUI_BMP_Serialize() parameter list

Example

The following example shows how to create a BMP file under windows.

```
static void _DrawSomething(void) {
  /* Draw something */
  GUI_DrawLine(10, 10, 100, 100);
}

static void _WriteByte2File(U8 Data, void * p) {
  U32 nWritten;
  WriteFile(*((HANDLE *)p), &Data, 1, &nWritten, NULL);
}

static void _ExportToFile(void) {
  HANDLE hFile = CreateFile("C:\\GUI_BMP_Serialize.bmp", GENERIC_WRITE, 0, 0,
                            CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
  GUI_BMP_Serialize(_WriteByte2File, &hFile);
  CloseHandle(hFile);
}

void MainTask(void) {
  GUI_Init();
  _DrawSomething();
  _ExportToFile();
}
```

GUI_BMP_SerializeEx()

Description

The function creates a BMP file containing the given area. The BMP file is created using the color depth which is used in emWin at a maximum of 24 bpp. In case of using a color depth of less than 8bpp the color depth of the BMP file will be 8bpp.
The currently selected device is used for reading the pixel data. If a Memory Device is selected it's content is written to the file.

Prototype

```
void GUI_BMP_SerializeEx(GUI_CALLBACK_VOID_U8_P * pfSerialize,
                         int    x0,    int y0,
                         int    xSize, int ySize,
```



```
                               void * p);
```

Parameter	Description
pfSerialize	Pointer to user defined serialization function. See prototype below.
x0	Start position in X to create the BMP file.
y0	Start position in Y to create the BMP file.
xSize	Size in X.
ySize	Size in Y.
p	Pointer to user defined data passed to serialization function.

Table 9.12: GUI_BMP_SerializeEx() parameter list

Prototype of GUI_CALLBACK_VOID_U8_P

```
void GUI_CALLBACK_VOID_U8_P(U8 Data, void * p);
```

Additional information

An example can be found in the description of `GUI_BMP_Serialize()`.

GUI_BMP_SerializeExBpp()

Description

The function creates a BMP file containing the given area using the specified color depth. In case of using a color depth of less than 8bpp the color depth of the BMP file will be 8bpp. The color depth should be a multiple of 8. In case of a system color depth of more than 8bpp the color depth needs to be 16bpp or more.
The currently selected device is used for reading the pixel data. If a Memory Device is selected it's content is written to the file.

Prototype

```
void GUI_BMP_SerializeExBpp(GUI_CALLBACK_VOID_U8_P * pfSerialize,
                            int     x0,     int y0,
                            int     xSize,  int ySize,
                            void * p,       int BitsPerPixel);
```

Parameter	Description
pfSerialize	Pointer to user defined serialization function. See prototype below.
x0	Start position in X to create the BMP file.
y0	Start position in Y to create the BMP file.
xSize	Size in X.
ySize	Size in Y.
p	Pointer to user defined data passed to serialization function.
BitsPerPixel	Color depth.

Table 9.13: GUI_BMP_SerializeExBpp() parameter list

Prototype of GUI_CALLBACK_VOID_U8_P

```
void GUI_CALLBACK_VOID_U8_P(U8 Data, void * p);
```

Additional information

An example can be found in the description of `GUI_BMP_Serialize()` above.

9.2 JPEG file support

JPEG (pronounced "jay-peg") is a standardized compression method for full-color and gray-scale images. JPEG is intended for compressing "real-world" scenes; line drawings, cartoons and other non-realistic images are not its strong suit. JPEG is lossy, meaning that the output image is not exactly identical to the input image. Hence you must not use JPEG if you have to have identical output bits. However, on typical photographic images, very good compression levels can be obtained with no visible change, and remarkably high compression levels are possible if you can tolerate a low-quality image.

9.2.1 Supported JPEG compression methods

This software implements JPEG baseline, extended-sequential and progressive compression processes. Provision is made for supporting all variants of these processes, although some uncommon parameter settings aren't implemented yet. For legal reasons, code for the arithmetic-coding variants of JPEG is not distributed. It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses. For this reason, support for arithmetic coding has not been included. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.)
The JPEG file support does not contain provision for the hierarchical or lossless processes defined in the standard. Further it supports only JPEG files based on the yCbCr color space and gray scale JPEGs.

9.2.2 Converting a JPEG file to C source

Under some circumstances it can be useful to add a JPEG file as C file to the project. In this case the JPEG file first needs to be converted to a C file. This can be done using the tool `Bin2C.exe` shipped with emWin. It can be found in the `Tools` subfolder. It converts the given binary file (in this case the JPEG file) to a C file. The filename of the C file is the same as the binary file name with the file extension '.c'.
The following steps will show how to embed a JPEG file using `Bin2C`:

- Start `Bin2C.exe` and select the JPEG file to be converted to a C file, for example 'Image.jpeg' and convert it to a C file.
- Add the C file to the project.

Example

The following example shows how to display the converted JPEG file:

```
#include "GUI.h"
#include "Image.c" /* Include the converted C file */

void MainTask(void) {
  GUI_Init();
  GUI_JPEG_Draw(acImage, sizeof(acImage), 0, 0);
  ...
}
```

9.2.3 Displaying JPEG files

The graphic library first decodes the graphic information. If the image has to be drawn the decoding process takes considerable time. If a JPEG file is used in a frequently called callback routine of the Window Manager, the decoding process can take a considerable amount of time. The calculation time can be reduced by the use of Memory Devices. The best way would be to draw the image first into a Memory Device. In this case the decompression would be executed only one time. For more information about Memory Devices, refer to chapter "Memory Devices" on page 319.

9.2.4 Memory usage

The JPEG decompression uses app. 33Kb RAM for decompression independent of the image size and a size dependent amount of bytes. The RAM requirement can be calculated as follows:

```
App. RAM requirement = X-Size of image * 80 bytes + 33 Kbytes
```

The X-size dependent amount depends on the compression type of the JPEG file. The following table shows some examples:

Compression	Size of image in pixels	RAM usage [Kbyte]	RAM usage, size dependent [Kbyte]
H1V1	160x120	45	12
H2V2	160x120	46	13
GRAY	160x120	38	4

Table 9.14: Memory usage for JPEG decompression

The memory required for the decompression is allocated dynamically by the emWin memory management system. After drawing the JPEG image the complete RAM will be released.

9.2.5 Progressive JPEG files

Contrary to baseline and extended-sequential JPEG files progressive JPEGs consist of multiple scans. Each of these scans is based on the previous scan(s) and refines the appearance of the JPEG image. This requires scanning the whole file even if only one line needs to be decompressed.
If enough RAM is configured for the whole image data, the decompression needs only be done one time. If less RAM is configured, the JPEG decoder uses 'banding' for drawing the image. The more bands required the more times the image needs to be decompressed and the slower the performance. With other words: The more RAM the better the performance.

9.2.6 JPEG API

The table below lists the available JPEG file related routines in alphabetical order. Detailed function descriptions follow:

Routine	Description
GUI_JPEG_Draw()	Draws a JPEG file which has been loaded into memory.
GUI_JPEG_DrawEx()	Draws a JPEG file which needs not to be loaded into memory.
GUI_JPEG_DrawScaled()	Draws a JPEG file with scaling which has been loaded into memory.
GUI_JPEG_DrawScaledEx()	Draws a JPEG file with scaling which needs not to be loaded into memory.
GUI_JPEG_GetInfo()	Fills a GUI_JPEG_INFO structure from a JPEG file which has been loaded into memory.
GUI_JPEG_GetInfoEx()	Fills a GUI_JPEG_INFO structure from a JPEG file which needs not to be loaded into memory.

Table 9.15: JPEG API list

GUI_JPEG_Draw()

Description

Draws a jpeg file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_JPEG_Draw(const void * pFileData, int DataSize, int x0, int y0);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the jpeg file resides.
DataSize	Number of bytes of the jpeg file.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.16: GUI_JPEG_Draw() parameter list

Return value

Zero on success, nonzero if the function fails. (The current implementation always returns 0)

Additional information

The Sample folder contains the example 2DGL_DrawJPG.c which shows how to use the function.

GUI_JPEG_DrawEx()

Description

Draws a jpeg file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_JPEG_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                    int                  x0,        int    y0);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.17: GUI_JPEG_DrawEx() parameter list

Return value

Zero on success, nonzero if the function fails. (The current implementation always returns 0)

Additional information

This function is used for drawing jpegs if not enough RAM is available to load the whole file into memory. The JPEG library then calls the function pointed by the parameter pfGetData to read the data.
The GetData function should return the number of available bytes. This could be less or equal the number of requested bytes. The function needs at least to return 1 new byte. The Sample folder contains the example 2DGL_DrawJPGScaled.c which shows how to use a GetData function.

GUI_JPEG_DrawScaled()

Description

Draws a `jpeg` file, which has been loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int  GUI_JPEG_DrawScaled(const void * pFileData, int DataSize,
                         int x0, int y0, int Num, int Denom);
```

Parameter	Description
`pFileData`	Pointer to the start of the memory area in which the jpeg file resides.
`DataSize`	Number of bytes of the jpeg file.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.
`Num`	Numerator to be used for scaling.
`Denom`	Denominator used for scaling.

Table 9.18: GUI_JPEG_DrawScaled() parameter list

Return value

Zero on success, nonzero if the function fails. (The current implementation always returns 0)

Additional information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrunk to 2/3 of size the parameter `Num` should be 2 and `Denom` should be 3.
The `Sample` folder contains the example `2DGL_DrawJPGScaled.c` which shows how to draw scaled JPEGs.

GUI_JPEG_DrawScaledEx()

Description

Draws a `jpeg` file, which does not have to be loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int  GUI_JPEG_DrawScaledEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                           int x0, int y0, int Num, int Denom);
```

Parameter	Description
`pfGetData`	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to "Getting data with the ...Ex() functions" on page 187.
`p`	Void pointer passed to the function pointed by `pfGetData`.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.
`Num`	Numerator to be used for scaling.
`Denom`	Denominator used for scaling.

Table 9.19: GUI_JPEG_DrawScaledEx() parameter list

Return value

Zero on success, nonzero if the function fails. (The current implementation always returns 0)

Additional information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrunk to 2/3 of size the parameter `Num` should be 2 and `Denom` should be 3.
For more details, refer to “GUI_JPEG_DrawEx()” on page 170.
The `Sample` folder contains the example `2DGL_DrawJPGScaled.c` which shows how to use the function.

GUI_JPEG_GetInfo()

Description

Fills a GUI_JPEG_INFO structure with information about a `jpeg` file, which has been loaded into memory.

Prototype

```
int GUI_JPEG_GetInfo(const void * pFileData, int DataSize,
                     GUI_JPEG_INFO * pInfo);
```

Parameter	Description
`pFileData`	Pointer to the start of the memory area in which the jpeg file resides.
`DataSize`	Number of bytes of the jpeg file.
`pInfo`	Pointer to a GUI_JPEG_INFO structure to be filled by the function.

Table 9.20: GUI_JPEG_GetInfo() parameter list

Return value

Zero on success, nonzero if the function fails.

Elements of structure GUI_JPEG_INFO

Data type	Element	Description
`int`	`XSize`	Pixel size in X of the image.
`int`	`YSize`	Pixel size in Y of the image.

Table 9.21: GUI_JPEG_INFO element list

Additional information

The `Sample` folder contains the example `2DGL_DrawJPG.c` which shows how to use the function.

GUI_JPEG_GetInfoEx()

Description

Fills a GUI_JPEG_INFO structure with information about a `jpeg` file, which does not have to be loaded into memory.

Prototype

```
int GUI_JPEG_GetInfoEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                       GUI_JPEG_INFO      * pInfo);
```

Parameter	Description
`pfGetData`	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to “Getting data with the ...Ex() functions” on page 187.
`p`	Void pointer passed to the function pointed by `pfGetData`.
`pInfo`	Pointer to a GUI_JPEG_INFO structure to be filled by the function.

Table 9.22: GUI_JPEG_GetInfoEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

For more details about the function and the parameters `pfGetData` and `p`, refer to “GUI_JPEG_GetInfo()” on page 172 and “GUI_JPEG_DrawEx()” on page 170.
The `Sample` folder contains the example `2DGL_DrawJPGScaled.c` which shows how to use the function.

9.3 GIF file support

The GIF file format (Graphic Interchange Format) has been developed by the CompuServe Information Service in the 1980s. It has been designed to transmit images across data networks.
The GIF standard supports interlacing, transparency, application defined data, animations and rendering of raw text. Unsupported data like raw text or application specific data will be ignored by emWin.
GIF files uses the LZW (Lempel-Zif-Welch) file compression method for compressing the image data. This compression method works without loosing data. The output image is exactly identical to the input image.

9.3.1 Converting a GIF file to C source

Under some circumstances it can be useful to add a GIF file as C file to the project. This can be done by exactly the same way as described before under 'JPEG file support'.

9.3.2 Displaying GIF files

The graphic library first decodes the graphic information. If the image has to be drawn the decoding process takes considerable time. If a GIF file is used in a frequently called callback routine of the Window Manager, the decoding process can take a considerable amount of time. The calculation time can be reduced by the use of Memory Devices. The best way would be to draw the image first into a Memory Device. In this case the decompression would be executed only one time. For more information about Memory Devices, refer to the chapter "Memory Devices" on page 319.

9.3.3 Memory usage

The GIF decompression routine of emWin needs about 16Kbytes of dynamically allocated RAM for decompression. After drawing an image the RAM which was used for decompression will be released.

9.3.4 GIF API

The table below lists the available GIF file related routines in alphabetical order. Detailed function descriptions follow:

Routine	Description
GUI_GIF_Draw()	Draws the first image of a GIF file which has been loaded into memory.
GUI_GIF_DrawEx()	Draws the first image of a GIF file which needs not to be loaded into memory.
GUI_GIF_DrawSub()	Draws the given sub image of a GIF file which has been loaded into memory.
GUI_GIF_DrawSubEx()	Draws the given sub image of a GIF file which needs not to be loaded into memory.
GUI_GIF_DrawSubScaled()	Draws the given sub image of a GIF file with scaling which has been loaded into memory.
GUI_GIF_DrawSubScaledEx()	Draws the given sub image of a GIF file with scaling which needs not to be loaded into memory.
GUI_GIF_GetComment()	Returns the given comment of a GIF file which has been loaded into memory.
GUI_GIF_GetCommentEx()	Returns the given comment of a GIF file which needs not to be loaded into memory.
GUI_GIF_GetImageInfo()	Returns information about the given sub image of a GIF file which has been loaded into memory.
GUI_GIF_GetImageInfoEx()	Returns information about the given sub image of a GIF file which needs not to be loaded into memory.

Table 9.23: GIF API list

Routine	Description
GUI_GIF_GetInfo()	Returns information about a GIF file which has been loaded into memory.
GUI_GIF_GetInfoEx()	Returns information about a GIF file which needs not to be loaded into memory.
GUI_GIF_GetXSize()	Returns the X-size of a bitmap loaded into memory.
GUI_GIF_GetXSizeEx()	Returns the X-size of a bitmap which needs not to be loaded into memory.
GUI_GIF_GetYSize()	Returns the Y-size of a bitmap loaded into memory.
GUI_GIF_GetYSizeEx()	Returns the Y-size of a bitmap which needs not to be loaded into memory.

Table 9.23: GIF API list

GUI_GIF_Draw()

Description

Draws the first image of a `gif` file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_Draw(const void * pGIF, U32 NumBytes, int x0, int y0);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the gif file resides.
NumBytes	Number of bytes of the gif file.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.24: GUI_GIF_Draw() parameter list

Return value

0 on success, != 0 on error.

Additional information

If the file contains more than one image, the function shows only the first image of the file. Transparency and interlaced images are supported.

GUI_GIF_DrawEx()

Description

Draws a `gif` file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to “Getting data with the ...Ex() functions” on page 187.
p	Void pointer passed to the function pointed by `pfGetData`.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.25: GUI_GIF_DrawEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

This function is used for drawing `gif` files if not enough RAM is available to load the whole file into memory. The library calls the function pointed by the parameter `pfGetData` to read the data.
The `GetData` function should return the number of available bytes. This could be less or equal the number of requested bytes. The function needs at least to return 1 new byte.

GUI_GIF_DrawSub()

Description

Draws the given sub image of a `gif` file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_DrawSub(const void * pGIF, U32 NumBytes,
                    int x0, int y0, int Index);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the gif file resides.
NumBytes	Number of bytes of the gif file.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Index	Zero-based index of sub image to be shown.

Table 9.26: GUI_GIF_DrawSub() parameter list

Return value

0 on success, != 0 on error.

Additional information

The function manages the background pixels between the current and the previous image. If for example sub image #3 should be drawn at offset x20/y20 with a size of w10/h10 and the previous sub image was shown at x15/y15 with a size of w20/h20 and the background needs to be redrawn, the function fills the pixels between the images with the background color.
The file `2DGL_DrawGIF.c` of the `Sample` folder shows how to use the function.

GUI_GIF_DrawSubEx()

Description

Draws the given sub image of a `gif` file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_DrawSubEx(GUI_GET_DATA_FUNC * pfGetData,
                      void * p, int x0, int  y0, int Index);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by `pfGetData`.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Index	Zero-based index of sub image to be shown.

Table 9.27: GUI_GIF_DrawSubEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

This function is used for drawing gif images if not enough RAM is available to load the whole file into memory. The GUI library then calls the function pointed by the parameter pfGetData to read the data.
For more details, refer to the "GUI_GIF_DrawEx()" on page 175.

GUI_GIF_DrawSubScaled()

Description

Draws the given sub image of a gif file, which has been loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int GUI_GIF_DrawSubScaled(const void * pGIF, U32 NumBytes, int x0, int y0,
                          int Index, int Num, int Denom);
```

Parameter	Description
pGif	Pointer to the start of the memory area in which the gif file resides.
NumBytes	Number of bytes of the gif file.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Index	Zero-based index of sub image to be shown.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Table 9.28: GUI_GIF_DrawSubScaled() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrunk to 2/3 of size the parameter Num should be 2 and Denom should be 3.

GUI_GIF_DrawSubScaledEx()

Description

Draws the given sub image of a gif file, which does not have to be loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int GUI_GIF_DrawSubScaledEx(GUI_GET_DATA_FUNC * pfGetData,
                            void * p,      int x0,   int y0,
                            int    Index, int Num, int Denom);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Table 9.29: GUI_GIF_DrawSubScaledEx() parameter list

Parameter	Description
Index	Zero-based index of sub image to be shown.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Table 9.29: GUI_GIF_DrawSubScaledEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrunk to 2/3 of size the parameter `Num` should be 2 and `Denom` should be 3.

GUI_GIF_GetComment()

Description

Returns the given comment from a GIF image, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetComment(const void * pGIF, U32 NumBytes,
                       U8 * pBuffer, int MaxSize, int Index);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the `gif` file resides.
NumBytes	Number of bytes of the gif file.
pBuffer	Pointer to a buffer to be filled with the comment.
MaxSize	Size of the buffer.
Index	Zero based index of comment to be returned.

Table 9.30: GUI_GIF_GetComment() parameter list

Return value

0 on success, != 0 on error.

Additional information

A GIF file can contain 1 or more comments. The function copies the comment into the given buffer. If the comment is larger than the given buffer only the bytes which fit into the buffer will be copied.
The file `2DGL_DrawGIF.c` of the `Sample` folder shows how to use the function.

GUI_GIF_GetCommentEx()

Description

Returns the given comment from a GIF image, which does not have to be loaded into memory.

Prototype

```
int GUI_GIF_GetCommentEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                         U8 * pBuffer, int MaxSize, int Index);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by `pfGetData`.

Table 9.31: GUI_GIF_GetCommentEx() parameter list

Parameter	Description
pBuffer	Pointer to a buffer to be filled with the comment.
MaxSize	Size of the buffer.
Index	Zero based index of comment to be returned.

Table 9.31: GUI_GIF_GetCommentEx() parameter list

Return value

0 on success, != 0 on error.

Additional information

For details, refer to "GUI_GIF_GetComment()" on page 178.

GUI_GIF_GetImageInfo()

Description

Returns information about the given sub image of a GIF file, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetImageInfo(const void * pGIF, U32 NumBytes,
                         GUI_GIF_IMAGE_INFO * pInfo, int Index);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the gif file resides.
NumBytes	Number of bytes of the gif file.
pInfo	Pointer to a GUI_GIF_IMAGE_INFO structure which will be filled by the function.
Index	Zero based index of sub image.

Table 9.32: GUI_GIF_GetImageInfo() parameter list

Return value

0 on success, != 0 on error.

Elements of structure GUI_GIF_IMAGE_INFO

Data type	Element	Description
int	xPos	X position of the last drawn image.
int	yPos	Y position of the last drawn image.
int	xSize	X size of the last drawn image.
int	ySize	Y size of the last drawn image.
int	Delay	Time in 1/100 seconds the image should be shown in a movie.

Table 9.33: GUI_GIF_IMAGE_INFO element list

Additional information

If an image needs be shown as a movie this function should be used to get the time the sub image should be visible and the next sub image should be shown.
If the delay member is 0 the image should be visible for 1/10 second.

GUI_GIF_GetImageInfoEx()

Description

Returns information about the given sub image of a GIF file, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetImageInfoEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
```



```
GUI_GIF_IMAGE_INFO * pInfo, int Index);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to “Getting data with the ...Ex() functions” on page 187.
p	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a GUI_GIF_IMAGE_INFO structure which will be filled by the function.
Index	Zero based index of sub image.

Table 9.34: GUI_GIF_GetImageInfoEx() parameter list

Return value

0 on success, != 0 on error.

Additional information

For more details, refer to “GUI_GIF_GetImageInfo()” on page 179.

GUI_GIF_GetInfo()

Description

Returns an information structure with information about the size and the number of sub images within the given GIF file, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetInfo(const void * pGIF, U32 NumBytes, GUI_GIF_INFO * pInfo);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the gif file resides.
NumBytes	Number of bytes of the gif file.
pInfo	Pointer to a GUI_GIF_INFO structure which will be filled by this function.

Table 9.35: GUI_GIF_GetInfo() parameter list

Return value

0 on success, != 0 on error.

Elements of structure GUI_GIF_INFO

Data type	Element	Description
int	XSize	Pixel size in X of the image.
int	YSize	Pixel size in Y of the image.
int	NumImages	Number of sub images in the file.

Table 9.36: GUI_GIF_INFO element list

GUI_GIF_GetInfoEx()

Description

Returns an information structure with information about the size and the number of sub images within the given GIF file, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetInfoEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
```



```
GUI_GIF_INFO * pInfo);;
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a GUI_GIF_INFO structure which will be filled by this function.

Table 9.37: GUI_GIF_GetInfoEx() parameter list

Return value

0 on success, != 0 on error.

Elements of structure GUI_GIF_INFO

Data type	Element	Description
int	XSize	Pixel size in X of the image.
int	YSize	Pixel size in Y of the image.
int	NumImages	Number of sub images in the file.

Table 9.38: GUI_GIF_INFO element list

GUI_GIF_GetXSize()

Description

Returns the X-size of a specified GIF image, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetXSize(const void * pGIF);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the gif file resides.

Table 9.39: GUI_GIF_GetXSize() parameter list

Return value

X-size of the GIF image.

GUI_GIF_GetXSizeEx()

Description

Returns the X-size of a specified GIF image, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.

Table 9.40: GUI_GIF_GetXSizeEx() parameter list

Return value

X-size of the GIF image.

GUI_GIF_GetYSize()

Description

Returns the Y-size of a specified GIF image, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetYSize(const void * pGIF);
```

Parameter	Description
pGIF	Pointer to the start of the memory area in which the bmp file resides.

Table 9.41: GUI_GIF_GetYSize() parameter list

Return value

Y-size of the GIF image.

GUI_GIF_GetYSizeEx()

Description

Returns the Y-size of a specified GIF image, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. Detailed information about the GetData() function can be found unter to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.

Table 9.42: GUI_GIF_GetYSizeEx() parameter list

Return value

Y-size of the GIF image.

9.4 PNG file support

The PNG (Portable Network Graphics) format is an image format which offers lossless data compression and alpha blending by using a non-patented data compression method. Version 1.0 of the PNG specification has been released in 1996. Since the end of 2003 PNG is an international standard (ISO/IEC 15948).
PNG support for emWin can be achieved by using the 'libpng' library from Glenn Randers-Pehrson, Guy Eric Schalnat and Andreas Dilger. An adapted version of this library ready to use with emWin is available under `www.segger.com/link/emwin_png.zip`. That library can be added to emWin in order to to use the PNG-API as explained later in this chapter.

Licensing

The use of 'libpng' library is subject to a BSD style license and copyright notice in the file `GUI\PNG\png.h` of the downloadable library. The original version of the library is available for free under `www.libpng.org`.

9.4.1 Converting a PNG file to C source

Under some circumstances it can be useful to add a PNG file as C file to the project. This can be done by exactly the same way as described before under 'JPEG file support'. Further the Bitmap Converter is able to load PNG files and can convert them into C bitmap files.

9.4.2 Displaying PNG files

The graphic library first decodes the graphic information. If the image has to be drawn the decoding process takes considerable time. If a PNG file is used in a frequently called callback routine of the Window Manager, the decoding process can take a considerable amount of time. The calculation time can be reduced by the use of Memory Devices. The best way would be to draw the image first into a Memory Device. In this case the decompression would be executed only one time. For more information about Memory Devices, refer to the chapter "Memory Devices" on page 319.

9.4.3 Memory usage

The PNG decompression uses app. 21Kbytes of RAM for decompression independent of the image size and a size dependent amount of bytes. The RAM requirement can be calculated as follows:

```
App. RAM requirement = (xSize + 1) * ySize * 4 + 54 Kbytes
```

9.4.4 PNG API

The table below lists the available PNG file related routines in alphabetical order. Detailed function descriptions follow:

Routine	Description
`GUI_PNG_Draw()`	Draws the PNG file which has been loaded into memory.
`GUI_PNG_DrawEx()`	Draws the PNG file which needs not to be loaded into memory.
`GUI_PNG_GetXSize()`	Returns the X-size of a bitmap loaded into memory.
`GUI_PNG_GetXSizeEx()`	Returns the X-size of a bitmap which needs not to be loaded into memory.
`GUI_PNG_GetYSize()`	Returns the Y-size of a bitmap loaded into memory.
`GUI_PNG_GetYSizeEx()`	Returns the Y-size of a bitmap which needs not to be loaded into memory.

Table 9.43: PNG API list

GUI_PNG_Draw()

Description

Draws a `png` file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_PNG_Draw(const void * pFileData, int FileSize, int x0, int y0);
```

Parameter	Description
`pFileData`	Pointer to the start of the memory area in which the png file resides.
`FileSize`	Number of bytes of the png file.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.

Table 9.44: GUI_PNG_Draw() parameter list

Return value

Zero on success, nonzero if the function fails. (The current implementation always returns 0)

Additional information

The `Sample` folder contains the example `2DGL_DrawPNG.c` which shows how to use the function.

GUI_PNG_DrawEx()

Description

Draws a `png` file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_PNG_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

Parameter	Description
`pfGetData`	Pointer to a function which is called for getting data. For details about the `GetData` function, refer to "Getting data with the ...Ex() functions" on page 187.
`p`	Void pointer passed to the function pointed by `pfGetData`.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.

Table 9.45: GUI_PNG_DrawEx() parameter list

Return value

Zero on success, nonzero if the function fails.

Additional information

This function is used for drawing `png` if not enough RAM is available to load the whole file into memory. The PNG library then calls the function pointed by the parameter `pfGetData` to read the data.
The `GetData` function should return the number of available bytes. This could be less or equal the number of requested bytes. The function needs at least to return 1 new byte. Note that the PNG library internally allocates a buffer for the complete image. This can not be avoided by using this function.

GUI_PNG_GetXSize()

Description

Returns the X-size of a specified PNG image, which has been loaded into memory.

Prototype

```
int GUI_PNG_GetXSize(const void * pFileData, int FileSize);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the png file resides.
FileSize	Size of the file in bytes.

Table 9.46: GUI_PNG_GetXSize() parameter list

Return value

X-size of the PNG image.

GUI_PNG_GetXSizeEx()

Description

Returns the X-size of a specified PNG image, which needs not to be loaded into memory.

Prototype

```
int GUI_PNG_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.

Table 9.47: GUI_PNG_GetXSizeEx() parameter list

Return value

X-size of the PNG image.

GUI_PNG_GetYSize()

Description

Returns the Y-size of a specified PNG image, which has been loaded into memory.

Prototype

```
int GUI_PNG_GetYSize(const void * pFileData, int FileSize);
```

Parameter	Description
pFileData	Pointer to the start of the memory area in which the png file resides.
FileSize	Size of the file in bytes.

Table 9.48: GUI_PNG_GetYSize() parameter list

Return value

Y-size of the PNG image.

GUI_PNG_GetYSizeEx()

Description

Returns the X-size of a specified PNG image, which needs not to be loaded into memory.

Prototype

```
int GUI_PNG_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. For details about the GetData function, refer to "Getting data with the ...Ex() functions" on page 187.
p	Void pointer passed to the function pointed by pfGetData.

Table 9.49: GUI_PNG_GetYSizeEx() parameter list

Return value

Y-size of the PNG image.

9.5 Getting data with the ...Ex() functions

As well as streamed bitmaps, using BMP, GIF, JPEG and PNG files also works without loading the whole image into RAM. For this case the ...Ex() functions can be used. Common for all of these functions is the use of a 'GetData' function. Please note that the 'GetData' function has to work slightly different depending on the actual task it is used for. See table of parameters and examples below.

Prototype of the 'GetData' function

```
int GUI_GET_DATA_FUNC(void * p, const U8 ** ppData, unsigned NumBytes,
                      U32 Off);
```

Parameter	Description
p	Application defined void pointer.
ppData	BMP, GIF & JPEG: The 'GetData' function has to set the pointer to the location the requested data resides in. Streamed bitmaps & PNG: The location the pointer points to has to be filled by the 'GetData' function.
NumBytes	Number of requested bytes.
Off	Defines the offset to use for reading the source data.

Table 9.50: GUI_GET_DATA_FUNC parameter list

Additional information

"...Ex()"-functions require the 'GetData'-function to fetch at least one pixel line of data. It is recommended to make sure that the 'GetData'-function is able to fetch at least one pixel line of the biggest image used by the application.

Internal use of the function

In general the 'GetData'-function is called one time at the beginning to retrieve overhead information and, after this, several times to retrieve the actual image data.

Return value

The number of bytes which were actually read. If the number of read bytes does not match, the drawing function will return immediately.

Example (BMP, GIF and JPEG)

The following code excerpt shows how to implement a 'GetData' function for usage with BMP, GIF and JPEG data:

```
int APP_GetData(void * p, const U8 ** ppData, unsigned NumBytes, U32 Off) {
  static char   _acBuffer[0x200];
  HANDLE      * phFile;
  DWORD         NumBytesRead;

  phFile = (HANDLE *)p;
  //
  // Check buffer size
  //
  if (NumBytes > sizeof(acBuffer)) {
    NumBytes = sizeof(acBuffer);
  }
  //
  // Set file pointer to the required position
  //
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  //
  // Read data into buffer
  //
  ReadFile(*phFile, acBuffer, NumBytes, &NumBytesRead, NULL);
  //
  // Set data pointer to the beginning of the buffer
  //
  *ppData = acBuffer;
  //
  // Return number of available bytes
  //
  return NumBytesRead;
```



```
}
```

Example (PNG and streamed bitmap)

The following code excerpt shows how to implement a 'GetData' function for usage with PNG and streamed bitmap data:

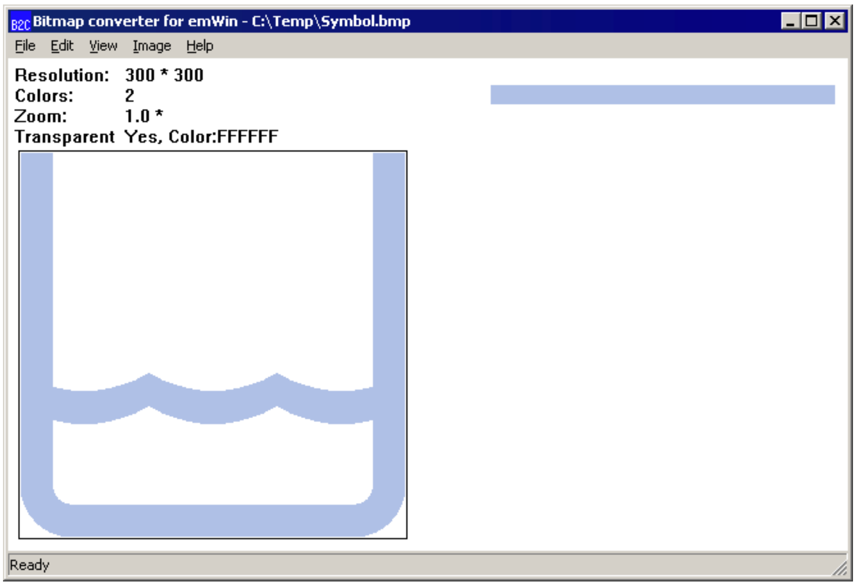
```
int APP_GetData(void * p, const U8 ** ppData, unsigned NumBytes, U32 Off) {
  HANDLE * phFile;
  DWORD    NumBytesRead;
  U8     * pData;

  pData  = (U8 *)*ppData;
  phFile = (HANDLE *)p;
  //
  // Set file pointer to the required position
  //
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  //
  // Read data into buffer
  //
  ReadFile(*phFile, pData, NumBytes, &NumBytesRead, NULL);
  //
  // Return number of available bytes
  //
  return NumBytesRead;
}
```


Chapter 10

Bitmap Converter

The Bitmap Converter is designed for converting common image file formats like BMP, PNG or GIF into the desired emWin bitmap format. That can be a C file which can directly be compiled and linked with the project or a binary format, which can be loaded at runtime. Simply load an image into the application. Convert the color format if you want or have to, and save it in the appropriate format.

Screenshot of the Bitmap Converter



10.1 What it does

The Bitmap Converter is primarily intended as a tool to convert bitmaps from a PC format to a C file. Bitmaps which can be used with emWin are normally defined as `GUI_BITMAP` structures in C. The structures - or rather the picture data which is referenced by these structures - can be quite large. It is time-consuming and inefficient to generate these bitmaps manually. We therefore recommend using the Bitmap Converter, which automatically generates C files from bitmaps.
An other useful feature is the ability to save images as C stream files. The advantage against a normal C file is, that these data streams can be located anywhere on any media whereas C files need to be located in the addressable CPU area.
It also features color conversion, so that the resulting C code is not unnecessarily large. You would typically reduce the number of bits per pixel in order to reduce memory consumption. The Bitmap Converter displays the converted image.
A number of simple functions can be performed with the Bitmap Converter, including scaling the size, flipping the bitmap horizontally or vertically, rotating it, and inverting the bitmap indices or colors (these features can be found under the `Image` menu).
Any further modifications to an image must be made in a bitmap manipulation program such as Adobe Photoshop or Corel Photopaint. It usually makes the most sense to perform any image modifications in such a program, using the Bitmap Converter for converting purposes only.

10.2 Loading a bitmap

10.2.1 Supported input file formats

The Bitmap Converter basically supports Windows bitmap files (*.bmp), "Graphic Interchange Format" (*.gif) and "Portable Network Graphics" (*.png):

Windows Bitmap Files (BMP)

The Bitmap Converter supports the most common bitmap file formats. Bitmap files of the following formats can be opened by the Bitmap Converter:

- 1, 4 or 8 bits per pixel (bpp) with palette;
- 16, 24 or 32 bpp without palette (full-color mode, in which each color is assigned an RGB value);
- RLE4 and RLE8.

Trying to read bitmap files of other formats will cause an error message of the Bitmap Converter.

Graphic Interchange Format (GIF)

The Bitmap Converter supports reading GIF files. For general editing only the first image of the GIF file is used. GIF image consisting of several images may be converted to animated sprites and animated cursors.
Transparency and interlaced GIF images are supported by the converter.

Portable Network Graphic (PNG)

The PNG format is the most recommended format to create images with alpha blending. The Bitmap Converter supports reading PNG images with alpha channel.

10.2.2 Loading from a file

An image file of one of the supported formats may be opened directly in the Bitmap Converter by selecting `File/Open`.

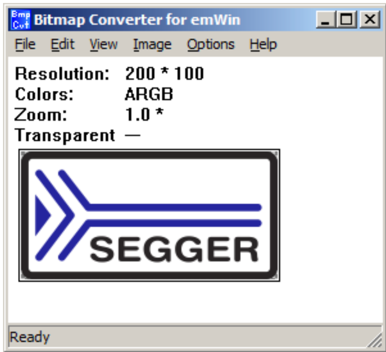
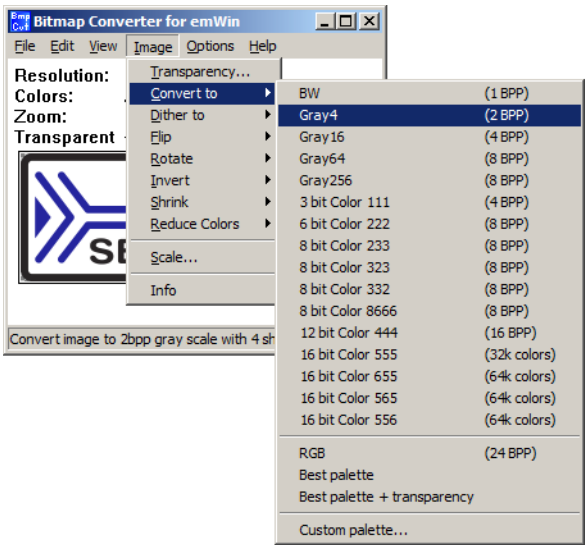
10.2.3 Using the clipboard

Any other type of bitmap (that is, `.jpg`, `.jpeg`, `.tif`) may be opened with another program, copied to the clipboard, and pasted into the Bitmap Converter. This process will achieve the same effect as loading directly from a file.

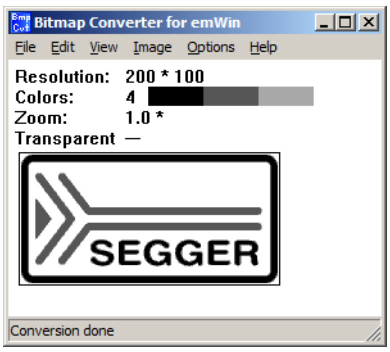
10.3 Color conversion

The primary reason for converting the color format of a bitmap is to reduce memory consumption. The most common way of doing this is by using the option `Best palette` as in the above example, which customizes the palette of a particular bitmap to include only the colors which are used in the image. It is especially useful with full-color bitmaps in order to make the palette as small as possible while still fully supporting the image. Once a bitmap file has been opened in the Bitmap Converter, simply select `Image/Convert Into/Best palette` from the menu. If it is necessary to keep transparency select `Image/Convert Into/Best palette + transparency`.
For certain applications, it may be more efficient to use a fixed color palette, chosen from the menu under `Image/Convert Into`. For example, suppose a bitmap in full-color mode is to be shown on a display which supports only four grayscales. It would be a waste of memory to keep the image in the original format, since it would only appear as four grayscales on the display. The full-color bitmap can be converted into a four-grayscale, 2bpp bitmap for maximum efficiency.
The procedure for conversion would be as follows:

<table>
<tr>
<td>The Bitmap Converter is opened and the same file is loaded as in steps 1 and 2 of the previous example.

The Bitmap Converter displays the loaded bitmap.</td>
<td>
</td>
</tr>
<tr>
<td>Choose "Image" -> "Convert Into" -> "Gray4"


Choose "Image" -> "Convert Into" -> "Gray4".

The Bitmap Converter displays the converted bitmap.</td>
<td>

In this example, the image uses less memory since a palette of only 4 grayscales is used instead of the full-color mode. If the target display supports only 4 grayscales, there is no use in having a higher pixel depth as it would only waste memory.</td>
</tr>
</table>

Table 10.1: Color conversion procedure

10.4 Dithering

Dithering is a method for showing more details on systems with only a few available colors than with a simple color conversion. It gives the illusion of a better color depth by using noise to randomize quantization error. If for example a photo needs to be drawn on a b/w system normally not much details would be visible after a simple conversion. However, dithering is able to show much more details:

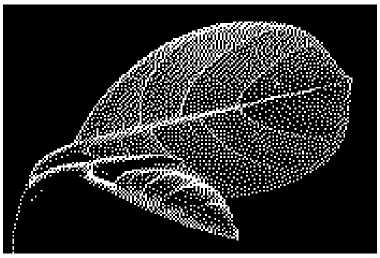
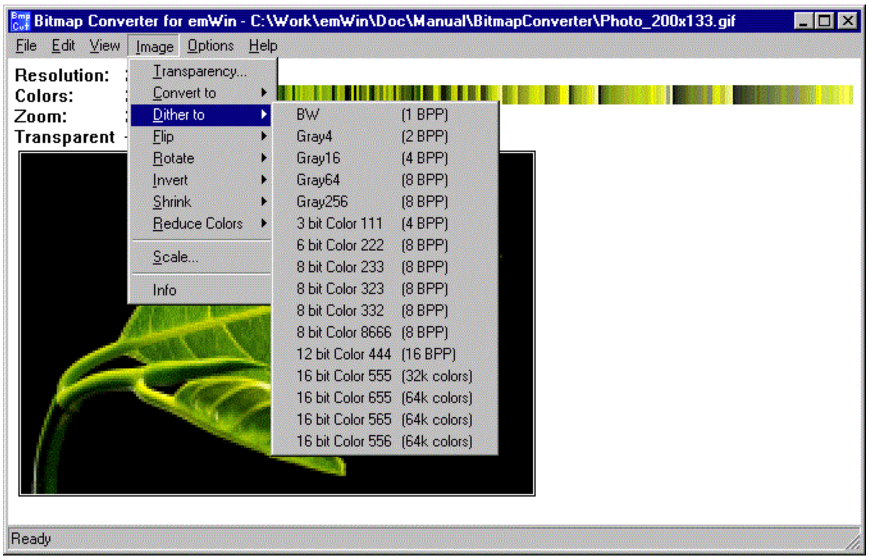
Original	black / white	black / white, dithered
		

Table 10.2: Dithering vs. simple conversion

The above table shows clearly the difference between dithering and a simple conversion. To dither a picture the command `Image/Dither to/...` should be used:



10.5 Using a custom palette

Converting bitmaps to a custom palette and saving them without palette information can save memory and can increase the performance of bitmap drawing operations.

More efficient memory utilization

Per default each bitmap contains its own palette. Even the smallest bitmaps can contain a large palette with up to 256 colors. In many cases only a small fraction of the palette is used by the bitmap. If using many of these bitmaps the amount of memory used by the palettes can grow rapidly.
So it can save much ROM if converting the bitmaps used by emWin to the available hardware palette and saving them as (D)evice (D)ependent (B)itmaps without palette information.

Better bitmap drawing performance

Before emWin draws a bitmap, it needs to convert each device independent bitmap palette to the available hardware palette. This is required because the pixel indices of the bitmap file are indices into the device independent bitmap palette and not to the available hardware palette.

Converting the bitmap to a DDB means that color conversion at run time is not required and speeds up the drawing.

10.5.1 Saving a palette file

The Bitmap Converter can save the palette of the currently loaded bitmap into a palette file which can be used for converting other bitmaps with the command `Image/Convert Into/Custom` palette. This requires that the current file is a palette based file and not a RGB file. To save the palette the command `File/Save palette...` can be used.

10.5.2 Palette file format

Custom palette files are simple files defining the available colors for conversion. They contain the following:

- Header (8 bytes)
- NumColors (U32, 4 bytes)
- 0 (4 bytes)
- U32 Colors[NumColors] (NumColors*4 bytes, type GUI_COLOR)

Total file size is therefore: 16 + (NumColors * 4) bytes. A custom palette file with 8 colors would be 16 + (8 * 4) = 48 bytes. At this point, a binary editor must be used in order to create such a file.
The maximum number of colors supported is 256; the minimum is 2.

Example

This example file would define a palette containing 2 colors -- red and white:

```
0000: 65 6d 57 69 6e 50 61 6c 02 00 00 00 00 00 00 00
0010: ff 00 00 00 ff ff ff 00
```

The 8 headers make up the first eight bytes of the first line. The U32 is stored lsb first (big endian) and represents the next four bytes, followed by the four 0 bytes. Colors are stored 1 byte per color, where the 4th byte is 0 as follows: RRGGBB00. The second line of code defines the two colors used in this example.

10.5.3 Palette files for fixed palette modes

Using the custom palette feature can even make sense with the most common used fixed palette modes, not only with custom hardware palettes. For the most palette based fixed palette modes a palette file can be found in the folder `Sample\Palette`.

10.5.4 Converting a bitmap

The command `Image/Convert Into/Custom palette` should be used for converting the currently loaded bitmap to a custom palette. The Bitmap Converter tries to find the nearest color of the palette file for each pixel of the currently loaded bitmap.

10.6 Generating C files from bitmaps

The main function of the Bitmap Converter is to convert PC-formatted bitmaps into C files which can be used by emWin. Before doing so, however, it is often desirable to modify the color palette of an image so that the generated C file is not excessively large.
The bitmap may be saved as a bmp, png or a gif file (which can be reloaded and used or loaded into other bitmap manipulation programs) or as a C file. A C file will serve as an input file for your C compiler. It may contain a palette (device-independent bitmap, or DIB) or be saved without (device-dependent bitmap, or DDB). DIBs are recommended, as they will display correctly on any display; a DDB will only display correctly on a display which uses the same palette as the bitmap.

C files may be generated as "C with palette", "C without palette", "C with palette, compressed" or "C without palette, compressed". For more information on compressed files, see the section "Compressed bitmaps" as well as the example at the end of the chapter.

10.6.1 Supported bitmap formats

The following table shows the currently available output formats for C and C stream files:

Format	Color depth [bpp]	Com-pression	Trans-parency	Palette
1 bit per pixel	1	no	yes	yes
2 bits per pixel	2	no	yes	yes
4 bits per pixel	4	no	yes	yes
8 bits per pixel	8	no	yes	yes
Compressed, RLE4	4	yes	yes	yes
Compressed, RLE8	8	yes	yes	yes
12 bits per pixel 444_12	12	no	no	no
12 bits per pixel M444_12, red and blue swapped	12	no	no	no
12 bits per pixel 444_12_1	12	no	no	no
12 bits per pixel M444_12_1, red and blue swapped	12	no	no	no
12 bits per pixel 444_16	12	no	no	no
12 bits per pixel M444_16, red and blue swapped	12	no	no	no
High color 555	15	no	no	no
High color 555, red and blue swapped	15	no	no	no
High color 565	16	no	no	no
High color 565, red and blue swapped	16	no	no	no
High color 565, compressed	16	yes	no	no
High color 565, red and blue swapped, compressed	16	yes	no	no
High color A555 with alpha channel	15	no	yes	no
High color AM555 with alpha channel, red and blue swapped	15	no	yes	no
High color A565 with alpha channel	16	no	yes	no
High color AM565 with alpha channel, red and blue swapped	16	no	yes	no
True color 888	24	no	no	no
True color 8888 with alpha channel	32	no	yes	no
True color 8888 with alpha channel, compressed	32	yes	yes	no
True color M8888I with alpha channel, red and blue swapped, alpha inverted	32	no	yes	no
Alpha channel, compressed	8	yes	yes	no

Table 10.3: Supported bitmap formats

10.6.2 Palette information

A bitmap palette is an array of 24 bit RGB color entries. Bitmaps with a color depth from 1 - 8 bpp can be saved with (device independent bitmap, DIB) or without palette information (device dependent bitmap DDB).

Device independent bitmaps (DIB)

The color information is stored in the form of an index into the color array. Before emWin draws a DIB, it converts the 24 bit RGB colors of the bitmap palette into color indices of the hardware palette. The advantage of using DIBs is that they are hardware independent and can be drawn correctly on systems with different color configurations. The disadvantages are the additional ROM requirement for the palette and the slower performance because of the color conversion.

Device dependent bitmaps (DDB)

The pixel information of a DDB is the index of the displays hardware palette. No conversion needs to be done before drawing a DDB. The advantages are less ROM requirement and a better performance. The disadvantage is that these bitmaps can not be displayed correctly on systems with other color configurations.

10.6.3 Transparency

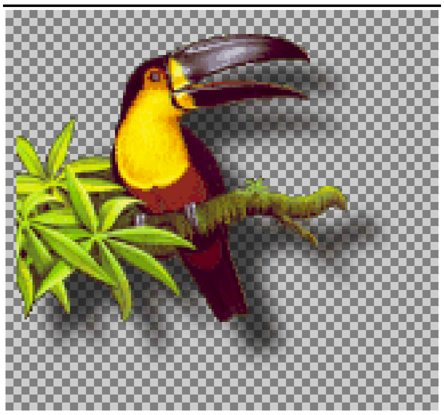
A palette based bitmap can be converted to a transparent bitmap. Transparency means each pixel with index 0 will not produce any output. The command `Image/Transparency` can be used to select the color which should be used for transparency. After selecting the transparent color, the pixel indices of the image will be recalculated, so that the selected color is on position 0 of the bitmap palette. When saving the bitmap file as C file, it will be saved with the transparency attribute.

10.6.4 Alpha blending

Alpha blending is a method of combining an image with the background to create the effect of semi transparency. The alpha value of a pixel determines its transparency. The color of a pixel after drawing the bitmap is a blend of the former color and the color value in the bitmap. In emWin, logical colors are handled as 32 bit values. The lower 24 bits are used for the color information and the upper 8 bits are used to manage the alpha value. An alpha value of 0 means the image is opaque and a value of 0xFF means completely transparent. Whereas BMP and GIF files do not support alpha blending PNG files support alpha blending. So the easiest way to create bitmap files with alpha blending is to load a PNG file. When working with BMP and/or GIF files the Bitmap Converter initially has no information about the alpha values.

Loading a PNG file

This is the most recommended way for creating bitmaps with an alpha mask:



The PNG file contains all required information.

Loading the alpha values from an alpha mask bitmap

This method loads the alpha values from a separate file. Black pixels of the alpha mask file means opaque and white means transparent. The following table shows an example:

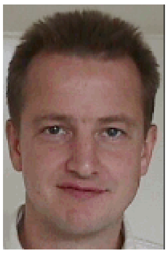
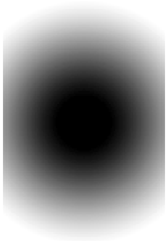
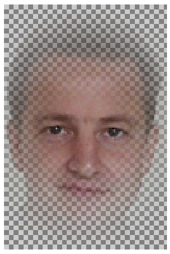
Starting point	Alpha mask	Result
		

Table 10.4: Loading alpha values from an alpha mask

The command `File/Load Alpha Mask` can be used for loading an alpha mask.

Creating the alpha values from two bitmaps

This method uses the difference between the pixels of two pictures to calculate the alpha values. The first image should show the item on a black background. The second image should show the same on a white background. The following table shows an example of how to create the alpha values using the command `File/Create Alpha`:

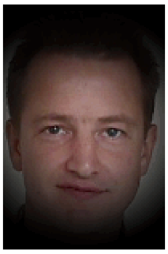
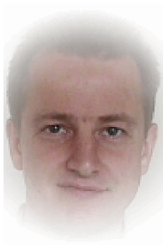
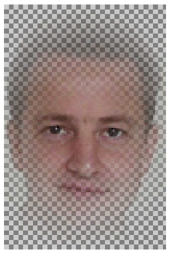
Starting point	Black background	White background	Result
			

Table 10.5: Creating alpha values from two bitmaps

The command `File/Create Alpha` can be used tor creating the alpha values.

10.6.5 Selecting the best format

emWin supports various formats for the generated C file. It depends on several conditions which will be the 'best' format and there is no general rule to be used. Color depth, compression, palette and transparency affect the drawing performance and/or ROM requirement of the bitmap.

Color depth

In general the lower the color depth the smaller the ROM requirement of the bitmap. Each display driver has been optimized for drawing 1bpp bitmaps (text) and bitmaps with the same color depth as the display.

Compression

The supported RLE compression method has the best effect on bitmaps with many horizontal sequences of equal-colored pixels. Details later in this chapter. The performance is typically slightly slower than drawing uncompressed bitmaps.

Palette

The ROM requirement of a palette is 4 bytes for each color. So a palette of 256 colors uses 1 Kbyte. Furthermore emWin needs to convert the colors of the palette before drawing the bitmap. Advantage: Bitmaps are device independent meaning they can be displayed on any display, independent of its color depth and format.

Transparency

The ROM requirement of transparent bitmaps is the same as without transparency. The performance is with transparency slightly slower than without.

High color and true color bitmaps

Special consideration is required for bitmaps in these formats. Generally the use of these formats only make sense on displays with a color depth of 15 bits and above. Further it is strongly recommended to save the C files in the exact same format used by the hardware. Note that using the right format will have a positive effect on the drawing performance. If a high color bitmap for example should be shown on a system with a color depth of 16bpp which has the red and blue components swapped, the best format is 'High color 565, red and blue swapped'. Already a slightly other format has the effect, that each pixel needs color conversion, whereas a bitmap in the right format can be rendered very fast without color conversion. The difference of drawing performance in this case can be factor 10 and more.

10.6.6 Saving the file

The basic procedure for using the Bitmap Converter is illustrated below:

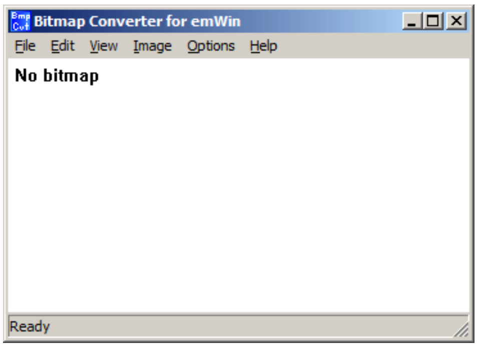
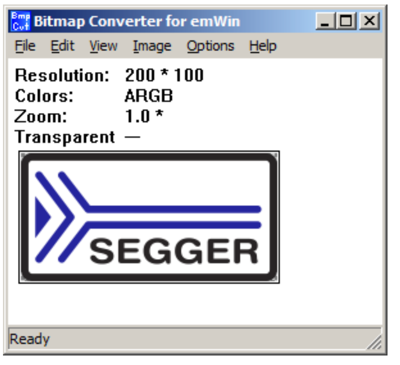
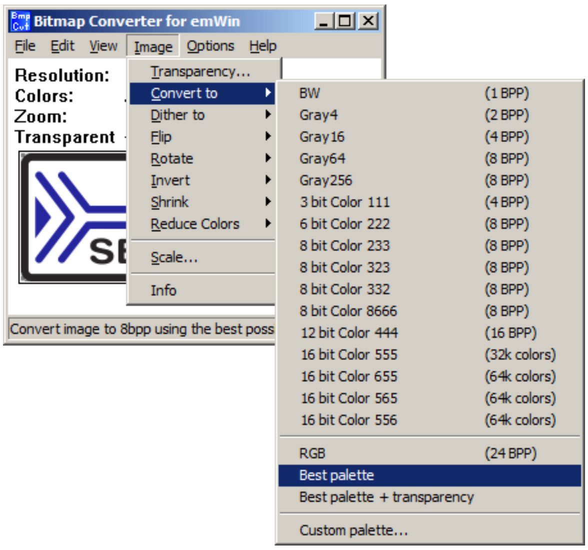
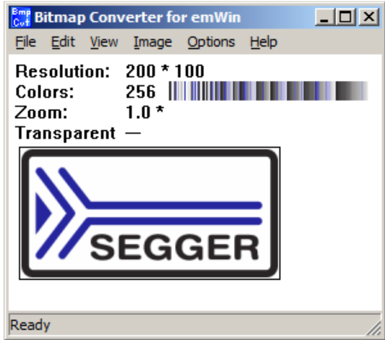
Step 1: Start the application. The Bitmap Converter is opened showing an empty window.	
Step 2: Load a bitmap into the Bitmap Converter. Choose File/Open. Locate the document you want to open and click Open. The Bitmap Converter can open images of the bmp, gif, png and sbmp format.  The Bitmap Converter displays the loaded bitmap.	

Table 10.6: Saving the file procedure



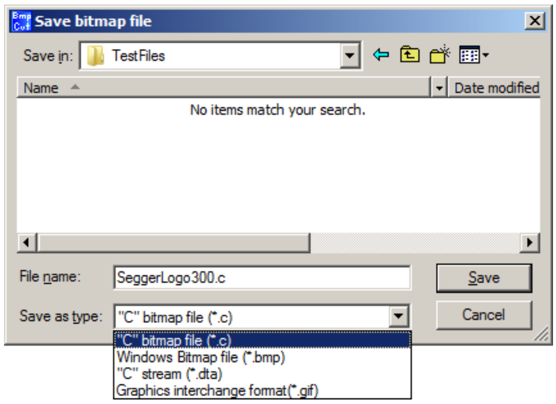
Choose Image/Convert Into.
Select the desired palette.
In this example, the option Best palette is chosen.
The Bitmap Converter displays the converted bitmap.



The image is unchanged in terms of appearance, but uses less memory since a palette of only 15 colors is used instead of the full-color mode. These 15 colors are the only ones actually required to display this particular image.

Step 4: Save the bitmap as a C file.

Choose File/Save As.
Select a destination and a name for the C file.
Select the file type. In this example, the file is saved as C bitmap file."
Click Save.



Step 5: Specify bitmap format.

If the bitmap should be saved as C file the format should now be specified. Use one of the available formats shown in the dialog. If the bitmap should be saved without palette, activate the check box "Without palette".

The Bitmap Converter will create a separate file in the specified destination, containing the C source code for the bitmap.

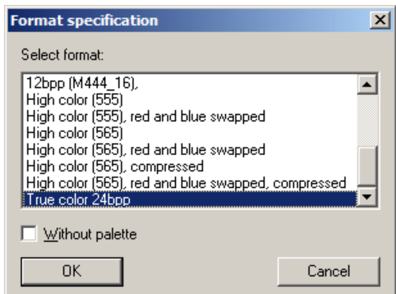


Table 10.6: Saving the file procedure

10.7 Generating C stream files

A C stream file consists of the same information as a C file. Contrary to a C file a data stream can be located anywhere and does not need to be compiled or linked with the project. All supported output formats described for C files are also available for C stream files. emWin supports creating bitmaps from data streams and drawing data streams directly. Detailed information about C stream file support can be found under “Drawing bitmaps” on page 119.

10.8 Compressed bitmaps

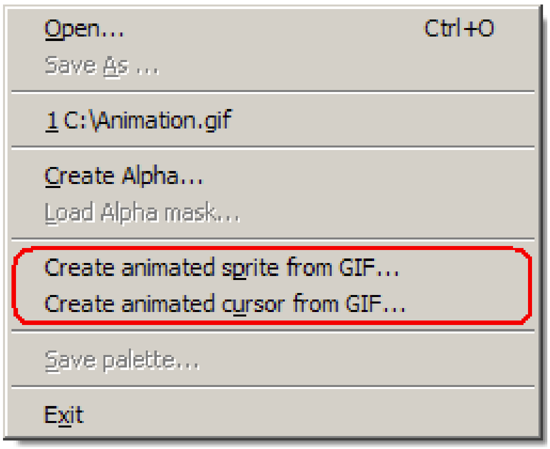
The Bitmap Converter and emWin support run-length encoding (RLE) compression of bitmaps in the resulting source code files. The RLE compression method works most efficiently if your bitmap contains many horizontal sequences of equal-colored pixels. An efficiently compressed bitmap will save a significant amount of space. However, compression is not recommended for photographic images since they do not normally have sequences of identical pixels. It should also be noted that a compressed image may take slightly longer to display.
Storing a bitmap using RLE compression can be done by selecting one of the according output formats when saving as a C file: "C with palette, compressed" or "C without palette, compressed". There are no special functions needed for displaying compressed bitmaps; they are displayed the same way uncompressed bitmaps are displayed.

Compression ratios

The ratio of compression achieved will vary depending on the bitmap used. The more horizontal uniformity in the image, the better the ratio will be. A higher number of bits per pixel will also result in a higher degree of compression.
In the bitmap used in the previous examples, the total number of pixels in the image is (200*94) = 18,800.
Since 2 pixels are stored in 1 byte, the total uncompressed size of the image is 18,800/2 = 9,400 bytes.
The total compressed size for this particular bitmap is 3,803 bytes for 18,800 pixels (see the example at the end of the chapter).
The ratio of compression can therefore be calculated as 9,400/3,803 = 2.47.

10.9 Creating animated sprites / cursors

The Bitmap Converter can be used to convert animated GIF files to animated sprites / cursors in C file format. This functionality is offered by the entries in the file menu which are shown below:



After clicking one of the according file menu entries, a file dialog appears and an animated GIF file can be chosen. Once this is done the name of the resulting C file needs to be specified. Converting animated GIF files to animated sprites / cursors does not require any further parameters. The process is performed automatically. Since the effort depends on the input GIF file, completing this task may take a moment. The Bitmap Converter can be used again as soon as the mouse cursor is changed to the simple arrow again.

Animated Sprite example

The following shows the structure of an animated sprite C file as it is generated by the Bitmap Converter. Although animations consist of several images, the palette and pixel data structures are shown only once here. Variable data is described using place holders.

File header

```
/*********************************************************************
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
**********************************************************************
*                                                                    *
*        Internet: www.segger.com    Support:  support@segger.com    *
*                                                                    *
**********************************************************************
*                                                                    *
* Source file: %_FILENAME_%.gif (Animated Sprite)                    *
* Dimensions:  %_X_SIZE_% * %_Y_SIZE_%                               *
* NumImages:   %_NUMBER_OF_IMAGES_%                                  *
* Duration:    %_OVERALL_DURATION_%                                  *
*                                                                    *
**********************************************************************
*
* %_USAGE_EXAMPLE_%
*/

#include <stdlib.h>

#include "GUI.h"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif
```

Palette and pixel data

```
static GUI_CONST_STORAGE GUI_COLOR%_FILENAME_%%_INDEX_%[] = {
  %_COLOR_DATA_%
};

static GUI_CONST_STORAGE GUI_LOGPALETTE _Pal%_FILENAME_%%_INDEX_% = {
  %_NUMBER_OF_COLORS_%,   // Number of entries
%_TRANSPARENCY_FLAG_%,  // No transparency
  &_Colors%_FILENAME_%%_INDEX_%[0]
};

static GUI_CONST_STORAGE unsigned char _ac%_FILENAME_%%_INDEX_%[] = {
  %_PIXEL_DATA_%
};
```

General data

```
static GUI_CONST_STORAGE GUI_BITMAP _abm%_FILENAME_%[] = {
  { %_X_SIZE_%,                 %_Y_SIZE_%,
    %_BYTES_PER_LINE_%,         %_BITS_PER_PIXEL_%,
    _ac%_FILENAME_%%_INDEX_%,  &_Pal%_FILENAME_%%_INDEX_%
  },
  [...]
};

const GUI_BITMAP * apbm%_FILENAME_%[] = {
  &_abm%_FILENAME_%[0],
  [...]
};

const unsigned aDelay%_FILENAME_%[] = {
  %_DELAY_DATA_%
};

/*************************** End of file ****************************/
```


Animated Cursor example

The file structure for animated cursors almost equals the structure for animated sprites. Therefor only the differences are mentioned here.
The array of bitmap pointers is defined as static:

```
static const GUI_BITMAP * _apbm%_FILENAME_%[] = {
  [...]
};
```

The array of delays is defined as static:

```
static const unsigned _aDelay%_FILENAME_%[] = {
  [...]
};
```

A non-static definition of a `GUI_CURSOR_ANIM` structure is placed at the end:

```
const GUI_CURSOR_ANIM Cursor%_FILENAME_% = {
  _apbm%_FILENAME_%,    // Pointer to an array of bitmaps
  0,                    // x coordinate of the hot spot
  0,                    // y coordinate of the hot spot
  0,                    // Period, should be 0 here
  _aDelay%_FILENAME_%,  // Pointer to an array of periods
  %_NUMBER_OF_IMAGES_%  // Number of images
};
```

Additional information

The hot spot coordinate define the position which is recognized by emWin when PID events occur. If the hot spot should not be represented by the topmost leftmost pixel, the according values in the `GUI_CURSOR_ANIM` structure may be modified.
The array of delays is always created. In case every image uses the same delay, the forth value in the `GUI_CURSOR_ANIM` structure may be set accordingly. In this case the array of delays may be deleted after the fifth value of the `GUI_CURSOR_ANIM` structure was set to `NULL`.

10.10 Command line usage

It is also possible to work with the Bitmap Converter using the command prompt. All conversion functions available in the Bitmap Converter menu are available as commands, and any number of functions may be performed on a bitmap in one command line.

10.10.1 Format for commands

Commands are entered using the following format:

```
BmpCvt <filename>.bmp <-command>
```

If more than one command is used, one space is typed between each. For example, a bitmap with the name logo.bmp is converted into Best palette format and saved as a C file named logo.bmp all at once by entering the following at the command prompt:

```
BmpCvt logo.bmp -convertintobestpalette -saveaslogo,1 -exit
```

Note that while the file to be loaded into the Bitmap Converter always includes its bmp extension, no file extension is written in the -saveas command. An integer is used instead to specify the desired file type. The number 1 in the -saveas command above designates "C with palette". The -exit command automatically closes the program upon completion. See the table below for more information.

10.10.2 Command line options

The following table lists all permitted Bitmap Converter commands. It can also be viewed at any time by entering BmpCvt -? at the command prompt.

Command	Description
-convertintobw	Convert to BW.
-convertintogray4	Convert to Gray4.
-convertintogray16	Convert to Gray16.
-convertintogray64	Convert to Gray64.
-convertintogray256	Convert to Gray256.
-convertinto111	Convert to 111.
-convertinto222	Convert to 222.
-convertinto233	Convert to 233.
-convertinto323	Convert to 323.
-convertinto332	Convert to 332.
-convertinto8666	Convert to 8666.
-convertintorgb	Convert to RGB.
-convertintobestpalette	Convert to best palette.
-convertintotranspalette	Convert to best palette with transparency.
-convertintocustompalette<filename>	Convert to a custom palette.
<filename>	User-specified filename of desired custom palette.
-exit	Terminate PC program automatically.
-fliph	Flip image horizontally.
-flipv	Flip image vertically.
-help	Display this box.
-invertindices	Invert indices.
-invertpalette	Invert palette entries.
-rotate90cw	Rotate image by 90 degrees clockwise.
-rotate90ccw	Rotate image by 90 degrees counter-clockwise.
-rotate180	Rotate image by 180 degrees.
-saveas<filename>,<type>[,<fmt>[,<noplt>]]	Save file as filename.
<filename>	User-specified file name including the file extension.
<type>	Must be an integer from 1 to 4 as follows: 1: C with palette (.c file) 2: Windows Bitmap file (bmp file) 3: C stream (.dta file) 4: GIF format (gif file)

Table 10.7: Command line options

Command	Description
<fmt>	Specifies the bitmap format (only if type == 1): 1: 1 bit per pixel* 2: 2 bits per pixel* 4: 4 bits per pixel* 5: 8 bits per pixel* 6: RLE4 compression* 7: RLE8 compression* 8: High color 565 9: High color 565, red and blue swapped 10: High color 555 11: High color 555, red and blue swapped 12: RLE16 compression 13: RLE16 compression, red and blue swapped 15: True color 32bpp, compressed 16: True color 32bpp 17: True color 24bpp 18: Alpha channel 8bpp, compressed 19: 16bpp (444_12) 20: 16bpp (444_12), RB swapped 21: 16bpp (444_12_1) 22: 16bpp (444_12_1), RB swapped 23: 16bpp (444_16) 24: 16bpp (444_16), RB swapped 25: High color (A555) with alpha channel 26: High color (AM555) with alpha channel, red and blue swapped 27: High color (A565) with alpha channel 28: High color (AM565) with alpha channel, red and blue swapped If this parameter is not given, the Bitmap Converter uses the following default formats in dependence of the number of colors of the bitmap: Number of colors <= 2: 1 bit per pixel Number of colors <= 4: 2 bits per pixel Number of colors <= 16: 4 bits per pixel Number of colors <= 256: 8 bits per pixel RGB: High color 565
<noplt>	Saves the bitmap with or without palette (only if type == 1) 0: Save bitmap with palette (default) 1: Save bitmap without palette
-transparency<RGB-Color>	Sets the transparent color.
<RGB-Color>	RGB color which should be used as transparent color.
-?	Displays the command line table.

Table 10.7: Command line options

* Images need to be converted to an according format before they can be stored in a format of 8 or less bpp.

10.11 Example of a converted bitmap

A typical example for the use of the Bitmap Converter would be the conversion of your company logo into a C bitmap. Take another look at the example bitmap pictured
below:

The bitmap is loaded into the Bitmap Converter, converted to `Best palette`, and saved as "C with palette". The resulting C source code is displayed below (some data is not shown to conserve space).

Resulting C code (generated by the Bitmap Converter)

```
/*********************************************************************
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
**********************************************************************
*                                                                    *
*        Internet: www.segger.com    Support:  support@segger.com    *
*                                                                    *
**********************************************************************
*
* Source file: SeggerLogo200
* Dimensions:  200 * 100
* NumColors:   33
*
**********************************************************************
*/

#include <stdlib.h>

#include "GUI.h"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

static GUI_CONST_STORAGE GUI_COLOR ColorsSeggerLogo200[] = {
  0xFFFFFF,0x353537,0x9C4B37,0xCDCDCD,
  [...]
};

static GUI_CONST_STORAGE GUI_LOGPALETTE PalSeggerLogo200 = {
  33,  /* number of entries */
  0,   /* No transparency */
  &ColorsSeggerLogo200[0]
};

static GUI_CONST_STORAGE unsigned char acSeggerLogo200[] = {
  0x00, 0x00,       /* Not all data is shown in this example */
  0x00, 0x92,
  [...]
  0xC6, 0x22,
  0x0A, 0x22
};

extern GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200;

GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200 = {
  200, /* XSize */
  100, /* YSize */
  200, /* BytesPerLine */
  8, /* BitsPerPixel */
  acSeggerLogo200,  /* Pointer to picture data (indices) */
  &PalSeggerLogo200  /* Pointer to palette */
};

/************************** End of file ***************************/
```


Compressing the file

We can use the same bitmap image to create a compressed C file, which is done simply by loading and converting the bitmap as before, and saving it as "C with palette, compressed". The source code is displayed below (some data is not shown to conserve space).

The compressed image size can be seen towards the end of the file as 3,730 bytes for 18,800 pixels.

Resulting compressed C code (generated by the Bitmap Converter)

```
/*********************************************************************
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
**********************************************************************
*                                                                    *
*        Internet: www.segger.com    Support:  support@segger.com    *
*                                                                    *
**********************************************************************
*
* Source file: SeggerLogo200_comp
* Dimensions:  200 * 100
* NumColors:   33
*
**********************************************************************
*/

#include <stdlib.h>

#include "GUI.h"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

static GUI_CONST_STORAGE GUI_COLOR ColorsSeggerLogo200_comp[] = {
  0xFFFFFF, 0x353537, 0x9C4B37, 0xCDCDCD,
  [...]
};

static GUI_CONST_STORAGE GUI_LOGPALETTE PalSeggerLogo200_comp = {
  33,/* number of entries */
  0, /* No transparency */
  &ColorsSeggerLogo200_comp[0]
};

static GUI_CONST_STORAGE unsigned char acSeggerLogo200_comp[] = {
  /* RLE: 006 Pixels @ 000,000*/ 6, 0x00,
  /* RLE: 188 Pixels @ 006,000*/ 188, 0x01,
  [...]
  /* RLE: 188 Pixels @ 006,099*/ 188, 0x01,
  /* RLE: 006 Pixels @ 194,099*/ 6, 0x00,
  0
};  /* 3730 for 20000 pixels */

extern GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200_comp;

GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200_comp = {
  200, /* XSize */
  100, /* YSize */
  200, /* BytesPerLine */
  GUI_COMPRESS_RLE8, /* BitsPerPixel */
  acSeggerLogo200_comp,  /* Pointer to picture data (indices) */
  &PalSeggerLogo200_comp  /* Pointer to palette */
 ,GUI_DRAW_RLE8
};

/*************************** End of file ****************************/
```


Chapter 11

Fonts

This chapter describes the various methods of font support in emWin. The most common fonts are shipped with emWin as C font files. All of them contain the ASCII character set and most of them do also contain the characters included in the ISO 8859-1 character set. In fact, you will probably find that these fonts are fully sufficient for your application. For detailed information about the individual fonts, refer to “Standard fonts” on page 230.
emWin is compiled for 8-bit characters, allowing for a maximum of 256 different character codes out of which the first 32 are reserved as control characters. The availability of characters depends on the font. In order to display certain characters selecting an according font may be required. For accessing the complete 'Basic Multilingual Plane' (BMP, plane 0) of the Unicode codespace UTF8 decoding could be enabled. Details can be found in the chapter “Language Support” on page 1035.
TrueType font files (TTF) can also be used directly. Support for those kind of fonts can be achieved by adding the FreeType library available under `www.segger.com/link/emwin_freetype.zip` which comes with its own BSD style license. More details about TTF support can be found under “TrueType Font (TTF) format” on page 213.

11.1 Introduction

The first way of font support was the possibility to use C files with font definitions containing bitmaps with 1bpp pixel information for each character. This kind of font support was limited to use only the fonts which are compiled with the application.
Over time, the font support has been improved regarding font quality, ROM requirement, performance, scalability and the ability to add further fonts at run time.
In the meantime emWin fonts cover antialiasing, drawing of compound characters like required in Thai language, fonts located on external non addressable media and TrueType support. Except the TrueType font format, which is a vector font, all other kinds of fonts are bitmap fonts.

11.2 Font types

emWin supports different internal types of fonts defined by emWin and the commonly used TrueType fonts.

Monospaced bitmap fonts

Each character of a monospaced bitmap font has the same size. In a proportional font each character has its own width, whereas in a monospaced font the width is defined only one time. The pixel information is saved with 1bpp and covers the whole character area.

Proportional bitmap fonts

Each character of a proportional bitmap font has the same height and its own width. The pixel information is saved with 1bpp and covers the whole character area.

Antialiased fonts with 2 bpp antialiasing information

Each character has the same height and its own width. The pixel information is saved with 2bpp antialiasing information and covers the whole character area.

Antialiased fonts with 4 bpp antialiasing information

Each character has the same height and its own width. The pixel information is saved with 4bpp antialiasing information and covers the whole character area.

Extended proportional bitmap fonts

Each character of an extended proportional bitmap font has its own height and its own width. The pixel information is saved with 1bpp and covers only the areas of the glyph bitmaps.

Extended proportional bitmap fonts with 2 bpp antialiasing information

Each character has the same height and its own width. The pixel information is saved with 2bpp antialiasing information and covers only the areas of the glyph bitmaps.

Extended proportional bitmap fonts with 4 bpp antialiasing information

Each character has the same height and its own width. The pixel information is saved with 4bpp antialiasing information and covers only the areas of the glyph bitmaps.

Extended proportional bitmap fonts, framed

In case the background color is unknown at compile time, it might be preferable to use a framed font. A framed font is always drawn in transparent mode regardless of the current settings. The character pixels are drawn in the currently selected foreground color and the frame is drawn in background color. A good contrast between foreground and background color makes sure, that the text can be read on any background. Framed fonts are not suitable for compound characters like in the Thai language. They are also not suitable for Arabic fonts.
The picture below shows some framed text in front of a photo:

Table of font types

The following table shows the difference between the font types. The pictures only show the pixel information saved in the font file:

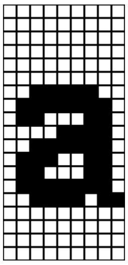
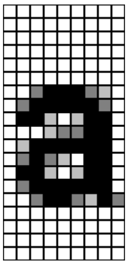
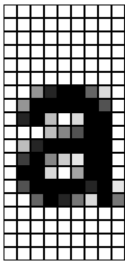
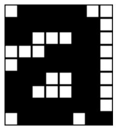
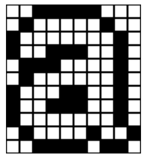
Prop. bitmap font	Prop. bitmap font, AA2	Prop. bitmap font, AA4	Ext. prop. bitmap font	Ext. prop. bitmap font, framed
				

Table 11.1: Font types

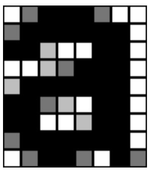
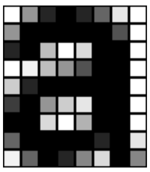
Ext. prop. bitmap font, AA2	Ext. prop. bitmap font, AA4
	

Table 11.2: Font types

TrueType vector fonts

The TrueType font support of emWin means support for the TrueType font file format described later in this chapter.

11.3 Font formats

The following explains the differences between the supported font formats, when to use them and what is required to be able to use them.

11.3.1 C file format

This is the most common way of using fonts. When using fonts in form of C files, we recommend compiling all available fonts and linking them as library modules or putting all of the font object files in a library which you can link with your application. This way you can be sure that only the fonts which are needed by your application are actually linked. The Font Converter may be used to create additional fonts.

When to use

This format should be used if the fonts are known at compile time and if there is enough addressable memory available for the font data.

Requirements

In order to be able to use a font C file in your application, the following requirements must be met:

- The font file is in a form compatible with emWin as C file, object file or library.
- The font file is linked with your application.
- The font declaration is contained in the application.

Format description

A font C file contains at first the pixel information of all characters included by the font. It is followed by a character information table with size information about each character. This table is followed by range information structures for each contiguous area of characters contained in the font file, whereas each structure points to the next one. Note that this method can enlarge a font file a lot if using many separate characters. After the range information structures a `GUI_FONT` structure follows with the main information like type, pixel size and so on of the font.

11.3.2 System Independent Font (SIF) format

System independent fonts are binary data blocks containing the font information. The Font Converter can be used to create system independent fonts. This tool is not part of the basic package. A short description follows later in this chapter.

When to use

This format should be used if the fonts are not known at compile time and if there is enough addressable memory available for the font data.

Requirements

In order to be able to use a SIF font file in your application, it is required that the whole file reside in addressable memory (ROM or RAM).

Format description

The structure of a SIF file is nearly the same as of a C file. It contains the same information in binary format. The sequence of the file components is vice versa: General font information followed by range information structures, character information table and at least pixel information of all characters.

11.3.3 External Bitmap Font (XBF) format

As well as SIF fonts XBF fonts are binary data blocks containing the font information and the Font Converter can be used to create XBF files. The Font Converter is not part of the emWin basic package. Details on how to create external bitmap fonts can be found in the chapter “Font Converter” on page 253.

Advantages

Contrary to other fonts, XBF fonts do not have to reside in memory when they are used, whereas all other kinds of emWin fonts need to reside completely in memory. The XBF font file can remain on any external media while it is used. Data access is done by a ’GetData’ callback function. The advantage of XBF fonts is that it is possible to use very large fonts on systems with little memory.
XBF fonts offer a performance advantage when using fonts including lots of characters which do not follow each other directly in sequence. This kind of character set would cause the Font Converter to create a C file font containing many `GUI_FONT_PROP` structures having a pointer to the according next one. The more `GUI_FONT_PROP` structures exist in a font the longer it might take to display a character. XBF fonts just use a memory offset so each character can be found in the same amount of time.

When to use

This format should be used if there is not enough addressable memory available for the font data and if there is any kind of external media available for storing the fonts.

Requirements

In order to be able to use a XBF font in your application, a ’GetData’ callback function is required which is responsible for getting font data.

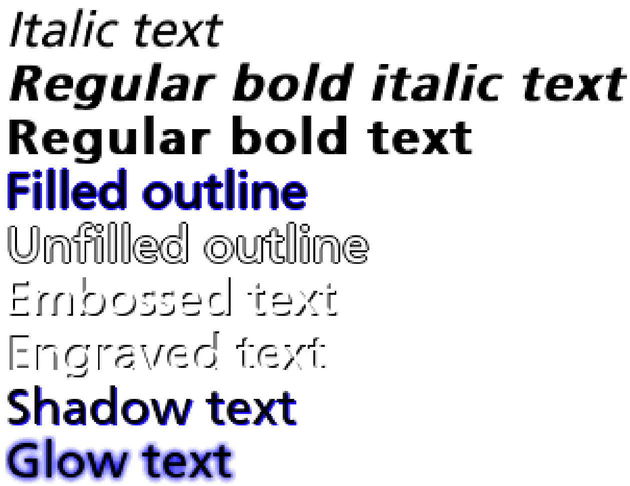
Format description

This format differs in general from SIF and C file format. At first it contains a small block of general font information including the lowest character code and the highest character code. It is followed by an access table containing offset and data size information for each character between lowest and highest character code. If a character does not exist, this information is zero for the according character. The access table is followed by the character information of all characters containing pixel data and character size information.

11.3.4 iType font engine support

Since version V5.20 emWin also supports using the iType® font engine. The iType® font engine is a font rendering subsystem developed by Monotype Imaging. It offers a host of advanced capabilities including font linking, font management and discovery, support for various industry standards and font formats in a small memory footprint. iType can be implemented into various platforms. Based on OpenType®, TrueType® and PostScript® font formats and packaged as ANSI C code for broad, flexible integration, iType meets stringent size requirements for any applications, including those that support East Asian languages requiring thousands of characters. The glue code to be able to use the iType® font engine is freely available under `www.segger.com/link/emwin_itype.zip`.

Screenshot



Licensing

The emWin library of Segger does not provide the iType® font engine itself. It provides only the glue code required to be able to use the iType library. Please contact Monotype Imaging under monotypeimaging.com for a licence request if required.

When to use

This format could be used if high quality fonts need to be scalable at run-time and/or advanced font effects are required.

Requirements

In general the requirements are similar to the requirements of the true type font support described on the next page. For detailed information about requirements and performance please also contact Monotype Imaging under monotypeimaging.com.

11.3.5 TrueType Font (TTF) format

The functionality of emWin can be enhanced by making use of TrueType fonts. TrueType is an outline font standard developed by Apple Computer. It offers font developers a high degree of control over how their fonts are displayed at various font heights. Contrary to bitmap fonts which are based on bitmaps for each character, TrueType fonts are based on vector graphics. The advantage of the vector representation is the loss-free scalability.
This implies that each character first needs to be rasterized into a bitmap before it is drawn. To avoid rasterization each time a character is drawn the bitmap data normally is cached by the font engine. This requires a fast CPU and enough RAM.
TTF support for emWin can be achieved by using the FreeType font library from David Turner, Robert Wilhelm and Werner Lemberg which is not part of emWin. An adapted version of this library ready to use with emWin is available under `www.segger.com/link/emwin_freetype.zip`. That library can be added to emWin in order to use the TTF-API as explained later in this chapter.

Licensing

The use of FreeType font library is subject to a BSD style license with credit clause (http://www.freetype.org/license.html) also included in GUI\TrueType\FTL.txt of the zip file. The original version of the library is available for free under `www.freetype.org`.

When to use

This format should be used if fonts need to be scalable at run-time.

Requirements

- CPU: TTF support works only on 32 bit CPUs. Our definition of a 32bit CPU: `sizeof(int)` = 4.
- ROM: The ROM requirement of the TTF engine is app. 250K. The exact size depends on the CPU, the compiler and the optimization level of the compiler.
- RAM: The RAM requirement of the library depends a lot on the used fonts. The basic RAM requirement of the TTF engine is app. 50K. When creating a GUI font with `GUI_TTF_CreateFont()` the font engine loads all font tables defined in the TTF file required to generate the characters. The table sizes varies a lot between the fonts. The additional required amount of RAM for creating a font can be between a few KB up to more than 1MB. For typical fonts 80-300 Kbytes are required. It depends on the used font file how much RAM is required. At least the TTF engine requires a bitmap cache. Per default the engine uses 200K for the cache. This should be enough for most applications.
  The TTF engine allocates its memory via the non emWin functions `malloc()` and `free()`. It must be made sure that these functions work before using the TTF engine.

Format description

For details about the TTF format, refer to the information available under www.apple.com.

11.4 Converting a TTF file to C source

Under some circumstances it can be useful to add a TTF file as 'C' file to the project, for example if no file system is available. This can be done by using the tool `Bin2C.exe` shipped with emWin. It can be found in the Tools subfolder. It converts the given binary file (in this case the TTF file) to a 'C' file.

11.5 Declaring custom fonts

The most recommended way of declaring the prototypes of custom fonts is to put them into an application defined header file. This should be included from each application source file which uses these fonts. It could look like the following example:

```
#include "GUI.h"

extern GUI_CONST_STORAGE GUI_FONT GUI_FontApp1;
extern GUI_CONST_STORAGE GUI_FONT GUI_FontApp2;
```

Note that this kind of declaring prototypes does not work if the fonts should be used with emWin configuration macros like `BUTTON_FONT_DEFAULT` or similar. In this case the fonts need to be declared in the configuration file `GUIConf.h`. The declaration in this case can look like the following example:

```
typedef struct GUI_FONT GUI_FONT;

extern const GUI_FONT GUI_FontApp1;

#define BUTTON_FONT_DEFAULT &GUI_FontApp1
#define EDIT_FONT_DEFAULT   &GUI_FontApp1
```

The `typedef` is required because the structure `GUI_FONT` has not been defined at the early point where `GUIConf.h` is included by emWin.

11.6 Selecting a font

emWin offers different fonts, one of which is always selected. This selection can be changed by calling the function `GUI_SetFont()` or one of the `GUI_XXX_CreateFont()` functions, which select the font to use for all text output to follow for the current task.

If no font has been selected by your application, the default font is used. This default is configured in `GUIConf.h` and can be changed. You should make sure that the default font is one that you are actually using in your application because the default font will be linked with your application and will therefore use up ROM memory.

11.7 Font API

The table below lists the available font-related routines in alphabetical order within their respective categories. Detailed descriptions can be found in the following sections.

Routine	Description
C file related font functions	
GUI_SetDefaultFont()	Sets the default font
GUI_SetFont()	Sets the current font
'SIF' file related font functions	
GUI_SIF_CreateFont()	Creates and selects a font by passing a pointer to system independent font data.
GUI_SIF_DeleteFont()	Deletes a font created by GUI_SIF_CreateFont()
'TTF' file related font functions	
GUI_TTF_CreateFont()	Creates a GUI font from a TTF font file.
GUI_TTF_CreateFontAA()	Creates a GUI font from a TTF font file with antialiasing.
GUI_TTF_DestroyCache()	Destroys the cache of the TTF engine.
GUI_TTF_Done()	Frees all dynamically allocated memory of the TTF engine.
GUI_TTF_GetFamilyName()	Returns the family name of the font.
GUI_TTF_GetStyleName()	Returns the style name of the font.
GUI_TTF_SetCacheSize()	Can be used to set the default size of the TTF cache.
'XBF' file related font functions	
GUI_XBF_CreateFont()	Creates and selects a font by passing a pointer to a callback function, which is responsible for getting data from the XBF font file.
GUI_XBF_DeleteFont()	Deletes a font created by GUI_XBF_CreateFont()
Common font-related functions	
GUI_GetCharDistX()	Returns the width in pixels (X-size) of a specified character in the current font.
GUI_GetFont()	Returns a pointer to the currently selected font.
GUI_GetFontDistY()	Returns the Y-spacing of the current font.
GUI_GetFontInfo()	Returns a structure containing font information.
GUI_GetFontSizeY()	Returns the height in pixels (Y-size) of the current font.
GUI_GetLeadingBlankCols()	Returns the number of leading blank pixel columns of the given character.
GUI_GetStringDistX()	Returns the X-size of a text using the current font.
GUI_GetTextExtend()	Evaluates the size of a text using the current font
GUI_GetTrailingBlankCols()	Returns the number of trailing blank pixel columns of the given character.
GUI_GetYDistOfFont()	Returns the Y-spacing of a particular font.
GUI_GetYSizeOfFont()	Returns the Y-size of a particular font.
GUI_IsInFont()	Evaluates whether a specified character is in a particular font.
GUI_SetDefaultFont()	Sets the default font to be used after GUI_Init().

Table 11.3: Font API list

11.7.1 C file related font functions

GUI_SetDefaultFont()

Description

Sets the font to be used by default for text output.

Prototype

```
void GUI_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font to be selected as default

Table 11.4: GUI_SetDefaultFont() parameter list

Additional information

This function is intended to be used in GUI_X_Config(). Defining GUI_DEFAULT_FONT is not mandatory anymore. If there is neither defined GUI_DEFAULT_FONT nor GUI_SetDefaultFont is called, GUI_Font6x8 will be set as the default Font. If none of the emWin fonts shall be used, GUI_DEFAULT_FONT has to be defined by NULL and a custom font needs to be set as default with this function.

GUI_SetFont()

Description

Sets the font to be used for text output.

Prototype

```
const GUI_FONT * GUI_SetFont(const GUI_FONT * pNewFont);
```

Parameter	Description
pFont	Pointer to the font to be selected and used.

Table 11.5: GUI_SetFont() parameter list

Return value

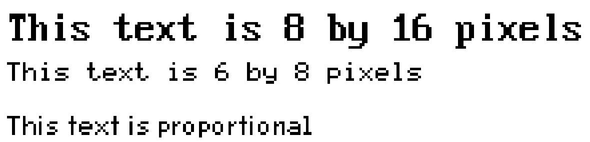
Returns a pointer to the previously selected font so that it may be buffered.

Examples

Displays example text in 3 different sizes, restoring the former font afterwards:

```
const GUI_FONT GUI_FLASH * OldFont;

OldFont = GUI_SetFont(&GUI_Font8x16);                    // Buffer old font
GUI_DispStringAt("This text is 8 by 16 pixels", 0, 0);
GUI_SetFont(&GUI_Font6x8);
GUI_DispStringAt("This text is 6 by 8 pixels", 0, 20);
GUI_SetFont(&GUI_Font8);
GUI_DispStringAt("This text is proportional", 0, 40);
GUI_SetFont(OldFont);                                    // Restore old font
```

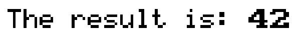
Screenshot of above example:



Displays text and value in different fonts:

```
GUI_SetFont(&GUI_Font6x8);
GUI_DispString("The result is: ");  // Disp text
GUI_SetFont(&GUI_Font8x8);
GUI_DispDec(42,2);                  // Disp value
```


Screenshot of above example:



11.7.2 'SIF' file related font functions

GUI_SIF_CreateFont()

Description

Sets the font to be used by passing a pointer to system independent font data.

Prototype

```
void GUI_SIF_CreateFont(void                * pFontData,
                        GUI_FONT            * pFont,
                        const GUI_SIF_TYPE  * pFontType);
```

Parameter	Description
pFontData	Pointer to the system independent font data.
pFont	Pointer to a GUI_FONT structure in RAM filled by the function.
pFontType	See table below.

Table 11.6: GUI_SIF_CreateFont() parameter list

Permitted values for element pFontType	
GUI_SIF_TYPE_PROP	Should be used if the parameter pFont points to a proportional font.
GUI_SIF_TYPE_PROP_EXT	Should be used if the parameter pFont points to an extended proportional font.
GUI_SIF_TYPE_PROP_FRM	Should be used if the parameter pFont points to an extended proportional framed font.
GUI_SIF_TYPE_PROP_AA2	Should be used if the parameter pFont points to a proportional font, which uses 2bpp antialiasing.
GUI_SIF_TYPE_PROP_AA4	Should be used if the parameter pFont points to a proportional font, which uses 4bpp antialiasing.
GUI_SIF_TYPE_PROP_AA2_EXT	Should be used if the parameter pFont points to an extended proportional font, which uses 2bpp antialiasing.
GUI_SIF_TYPE_PROP_AA4_EXT	Should be used if the parameter pFont points to an extended proportional font, which uses 4bpp antialiasing.

Additional information

Contrary to the emWin standard fonts which must be compiled and linked with the application program, system independent fonts (SIF) are binary data blocks containing the font information. The Font Converter can be used to create system independent fonts. This tool is not part of the basic package. A short description follows later in this chapter. For details about how to create system independent fonts, refer to the chapter "Font Converter" on page 253.
When using this function emWin needs to fill a GUI_FONT structure with the font information. The user needs to pass a pointer to this structure in the parameter pFont. The contents of this structure must remain valid during the use of the font.
The function does not know what kind of font should be created. To tell the function the type of the font to be created it must be passed in the parameter pFontType. This has been done to avoid linkage of code which is not required.

Example

```
static GUI_FONT _Font;  // Font structure in RAM

GUI_SIF_CreateFont(_DownloadedFont, &_Font, GUI_SIF_TYPE_PROP);
GUI_DispString("Hello World!");
```

GUI_SIF_DeleteFont()

Description

Deletes a font pointed by the parameter `pFont`.

Prototype

```
void GUI_SIF_DeleteFont(GUI_FONT * pFont);
```

Parameter	Description
`pFont`	Pointer to the font to be deleted.

Table 11.7: GUI_SIF_DeleteFont() parameter list

Additional information

After using a font created with `GUI_SIF_CreateFont()` the font should be deleted if not used anymore.

Example

```
static GUI_FONT _Font;  // Font structure in RAM

GUI_SIF_CreateFont(_DownloadedFont, &_Font, GUI_SIF_TYPE_PROP);
//
// Use the font
//
GUI_SIF_DeleteFont(&_Font);
```

11.7.3 'TTF' file related font functions

The emWin implementation of TTF file support is based on the FreeType font library from David Turner, Robert Wilhelm and Werner Lemberg. For details, refer to "True-Type Font (TTF) format" on page 213.

GUI_TTF_CreateFont()

Description

Creates and selects an emWin font by using a TTF font file.

Prototype

```
int GUI_TTF_CreateFont(GUI_FONT * pFont, GUI_TTF_CS * pCS);
```

Parameter	Description
`pFont`	Pointer to a `GUI_FONT` structure in RAM filled by the function.
`pCS`	Pointer to a `GUI_TTF_CS` structure containing the creation parameters.

Table 11.8: GUI_TTF_CreateFont() parameter list

Return value

0 on success, 1 on error.

Elements of structure GUI_TTF_CS

Data type	Element	Description
GUI_TTF_DATA *	pTTF	Pointer to GUI_TTF_DATA structure which contains location and size of the font file to be used.
PixelHeight	PixelHeight	Pixel height of new font. It means the height of the surrounding rectangle between the glyphs 'g' and 'f'. Note that it is not the distance between two lines of text. With other words the value returned by GUI_GetFontSizeY() is not identical with this value.
FaceIndex	FaceIndex	Some font files can contain more than one font face. In case of more than one face this index specifies the zero based face index to be used to create the font. Usually 0.

Table 11.9: GUI_TTF_CS element list

Elements of structure GUI_TTF_DATA

Data type	Element	Description
const void *	pData	Pointer to TTF font file in addressable memory area.
NumBytes	NumBytes	Size of file in bytes.

Table 11.10: GUI_TTF_DATA element list

Additional information

When using the function the first time it initializes the TTF engine and the internal cache system. If the cache should use other values as defined per default it needs to be configured before the first call of this function. For details how to configure the cache, refer to “GUI_TTF_SetCacheSize()” on page 221.
The internal data cache manages the complete mechanism of creating fonts and caching bitmap data. Font faces are uniquely identified from the cache by the address given in parameter pTTF and the parameter FaceIndex, which normally is 0. If the same font file for example should be used for creating fonts of different sizes the parameter pTTF should point to the same location of a GUI_TTF_DATA structure.
The parameter PixelHeight specifies the height of the surrounding rectangle between the glyphs ’g’ and ’f’. The value PixelHeight does not represent the offset between lines.

Example

```
GUI_TTF_CS   Cs0, Cs1;
GUI_TTF_DATA Data;
GUI_FONT     Font0, Font1;

//
// Set parameters for accessing the font file
//
Data.pData      = aTTF;         // Address
Data.NumBytes   = sizeof(aTTF); // Size
//
// Set creation parameters of first font
//
Cs0.pTTF        = &Data;        // Use address of GUI_TTF_DATA
Cs0.PixelHeight = 24;           // Pixel height
Cs0.FaceIndex   = 0;            // Initialize to 0
//
// Set creation parameters of second font
//
Cs1.pTTF        = &Data;        // Use address of GUI_TTF_DATA
Cs1.PixelHeight = 48;           // Pixel height
Cs1.FaceIndex   = 0;            // Initialize to 0
//
// Create 2 fonts
//
GUI_TTF_CreateFont(&Font0, &Cs0);
GUI_TTF_CreateFont(&Font1, &Cs1);
//
// Draw something using the fonts
//
GUI_SetFont(&Font0);
GUI_DispString("Hello world\n");
GUI_SetFont(&Font1);
```



```
GUI_DispString("Hello world");
```

GUI_TTF_CreateFontAA()

Description

Creates and selects an antialised emWin font by using a TTF font file.

Prototype

```
int GUI_TTF_CreateFontAA(GUI_FONT * pFont, GUI_TTF_CS * pCS);
```

Parameter	Description
pFont	Pointer to a GUI_FONT structure in RAM filled by the function.
pCS	Pointer to a GUI_TTF_CS structure containing the creation parameters.

Table 11.11: GUI_TTF_CreateFontAA() parameter list

Return value

0 on success, 1 on error.

GUI_TTF_DestroyCache()

Description

This function frees all memory allocated by the TTF cache system and destroys the cache.

Prototype

```
void GUI_TTF_DestroyCache(void);
```

Additional information

The next time GUI_TTF_CreateFont() is used emWin automatically creates and initializes a new cache.

GUI_TTF_Done()

Description

This function frees all memory allocated by the TTF engine and its internal cache system.

Prototype

```
void GUI_TTF_Done(void);
```

Additional information

The next time GUI_TTF_CreateFont() is used emWin automatically initializes the TTF engine and creates and initializes a new cache.

GUI_TTF_GetFamilyName()

Description

The function returns the font family name defined in the font file.

Prototype

```
int GUI_TTF_GetFamilyName(GUI_FONT * pFont, char * pBuffer, int NumBytes);
```

Parameter	Description
pFont	Pointer to a GUI_FONT structure which has been created using GUI_TTF_CreateFont().
pBuffer	Buffer to be filled with the family name.
NumBytes	Size of buffer in bytes.

Table 11.12: GUI_TTF_GetFamilyName() parameter list

Return value

0 on success, 1 on error.

GUI_TTF_GetStyleName()

Description

The function returns the style name (bold, regular, ...) defined in the font file.

Prototype

```
int GUI_TTF_GetStyleName(GUI_FONT * pFont, char * pBuffer, int NumBytes);
```

Parameter	Description
pFont	Pointer to a GUI_FONT structure which has been created using `GUI_TTF_CreateFont()`.
pBuffer	Buffer to be filled with the style name.
NumBytes	Size of buffer in bytes.

Table 11.13: GUI_TTF_GetStyleName() parameter list

Return value

0 on success, 1 on error.

GUI_TTF_SetCacheSize()

Description

Sets the size parameters used to create the cache on the first call of `GUI_TTF_CreateFont()`.

Prototype

```
void GUI_TTF_SetCacheSize(unsigned MaxFaces,
                          unsigned MaxSizes, U32 MaxBytes);
```

Parameter	Description
MaxFaces	Maximum number of font faces the cache should be able to handle simultaneously. 0 selects default value.
MaxSizes	Maximum number of size objects the cache should be able to handle simultaneously. 0 selects default value.
MaxBytes	Maximum number of bytes used for the bitmap cache. 0 selects default value.

Table 11.14: GUI_TTF_SetCacheSize() parameter list

Additional information

If for example 3 font faces should be used, each with 2 sizes, the cache should be able to manage 6 size objects.
The default values used by the TTF engine are: 2 faces, 4 size objects and 200K of bitmap data cache.

11.7.4 'XBF' file related font functions

GUI_XBF_CreateFont()

Description

Creates and selects a font by passing a pointer to a callback function, which is responsible for getting data from the XBF font file.

Prototype

```
int GUI_XBF_CreateFont(GUI_FONT              * pFont,
                       GUI_XBF_DATA          * pXBF_Data,
                       const GUI_XBF_TYPE    * pFontType,
                       GUI_XBF_GET_DATA_FUNC * pfGetData,
                       void                  * pVoid);
```

Parameter	Description
pFont	Pointer to a GUI_FONT structure in RAM filled by the function.
pXBF_Data	Pointer to a GUI_XBF_DATA structure in RAM filled by the function.
pFontType	See table below.
pfGetData	Pointer to a callback function which is responsible for getting data from the font file. See prototype below.
pVoid	Application defined pointer passed to the 'GetData' callback function.

Table 11.15: GUI_XBF_CreateFont() parameter list

Permitted values for element pFontType	
GUI_XBF_TYPE_PROP	Should be used if the parameter pFont points to a proportional font.
GUI_XBF_TYPE_PROP_EXT	Should be used if the parameter pFont points to an extended proportional font.
GUI_XBF_TYPE_PROP_FRM	Should be used if the parameter pFont points to an extended framed proportional font.
GUI_XBF_TYPE_PROP_AA2_EXT	Should be used if the parameter pFont points to an extended proportional font, which uses 2bpp antialiasing.
GUI_XBF_TYPE_PROP_AA4_EXT	Should be used if the parameter pFont points to an extended proportional font, which uses 4bpp antialiasing.

GUI_XBF_GET_DATA_FUNC

```
int GUI_XBF_GET_DATA_FUNC(U32     Off,    U16    NumBytes,
                          void * pVoid, void * pBuffer);
```

The function has to set pBuffer to point to the location the requested data resides in.

Additional information

The parameter `pfGetData` should point to an application defined callback routine, which is responsible for getting data from the font. Parameter `pVoid` is passed to the callback function when requesting font data. It can be used for example to pass a file handle to the callback function.
The function requires pointers to a `GUI_FONT` structure and a `GUI_XBF_DATA` structure. The function will fill these structures with font information. It is required, that the contents of these structures remain valid during the usage of the font. The function does not know what kind of XBF font has to be created, so the parameter `pFontType` has to be used to tell the function the type of the font to be created. This has been done to avoid unnecessary linkage of code.

The maximum number of data bytes per character is limited to 200 per default. This should cover the most requirements. If loading a character with more bytes a warning will be generated in the debug version. The default value can be increased by adding the following define to the file `GUIConf.h`:

```
#define GUI_MAX_XBF_BYTES 500 // Sets the maximum number of bytes/chars to 500
```

Example

```
static  GUI_FONT     Font;     // GUI_FONT structure in RAM
static  GUI_XBF_DATA XBF_Data; // GUI_XBF_DATA structure in RAM

static int _cbGetData(U32 Off, U16 NumBytes, void * pVoid, void * pBuffer) {
  //
  // The pVoid pointer may be used to get a file handle
  //
  ...// TBD
  //
  // Set file pointer to the given position
  //
  ...// TBD
  //
  // Read the required number of bytes into the given buffer
  //
  ...// TBD
  //
  // Return 0 on success. Return 1 if the function fails.
  //
}

GUI_XBF_CreateFont(&Font, &XBF_Data, GUI_XBF_TYPE_PROP, _cbGetData, pVoid);
```

GUI_XBF_DeleteFont()

Description

Deletes an XBF font pointed by the parameter `pFont`.

Prototype

```
void GUI_XBF_DeleteFont(GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font to be deleted.

Table 11.16: GUI_XBF_DeleteFont() parameter list

Additional information

After using a font created with `GUI_XBF_CreateFont()` the font should be deleted if not used anymore.

11.7.5 Common font-related functions

GUI_GetFont()

Description

Returns a pointer to the currently selected font.

Prototype

```
const GUI_FONT * GUI_GetFont(void)
```

GUI_GetCharDistX()

Description

Returns the width in pixels (X-size) used to display a specified character in the currently selected font.

Prototype

```
int GUI_GetCharDistX(U16 c);
```

Parameter	Description
c	Character to calculate width from.

Table 11.17: GUI_GetCharDistX() parameter list

GUI_GetFontDistY()

Description

Returns the Y-spacing of the currently selected font.

Prototype

```
int GUI_GetFontDistY(void);
```

Additional information

The Y-spacing is the vertical distance in pixels between two adjacent lines of text. The returned value is the YDist value of the entry for the currently selected font. The returned value is valid for both proportional and monospaced fonts.

GUI_GetFontInfo()

Description

Calculates a pointer to a GUI_FONTINFO structure of a particular font.

Prototype

```
void GUI_GetFontInfo(const GUI_FONT * pFont, GUI_FONTINFO * pfi);
```

Parameter	Description
pFont	Pointer to the font.
pfi	Pointer to a GUI_FONTINFO structure.

Table 11.18: GUI_GetFontInfo() parameter list

Additional information

The definition of the GUI_FONTINFO structure is as follows:

```
typedef struct {
  U16 Flags;
} GUI_FONTINFO;
```

The member variable flags can take the following values:


```
GUI_FONTINFO_FLAG_PROP
GUI_FONTINFO_FLAG_MONO
GUI_FONTINFO_FLAG_AA
GUI_FONTINFO_FLAG_AA2
GUI_FONTINFO_FLAG_AA4
```

Example

Gets the info of GUI_Font6x8. After the calculation `FontInfo.Flags` contains the flag `GUI_FONTINFO_FLAG_MONO`.

```
GUI_FONTINFO FontInfo;
GUI_GetFontInfo(&GUI_Font6x8, &FontInfo);
```

GUI_GetFontSizeY()

Description

Returns the height in pixels (Y-size) of the currently selected font.

Prototype

```
int GUI_GetFontSizeY(void);
```

Additional information

The returned value is the `YSize` value of the entry for the currently selected font. This value is less than or equal to the Y-spacing returned by the function `GUI_GetFontDistY()`.
The returned value is valid for both proportional and monospaced fonts.

GUI_GetLeadingBlankCols()

Description

Returns the number of leading blank pixel columns in the currently selected font for the given character.

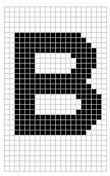
Prototype

```
int GUI_GetLeadingBlankCols(U16 c);
```

Parameter	Description
`c`	Character to be used.

Table 11.19: GUI_GetLeadingBlankCols() parameter list

Example



The result for the character ’B’ shown in the screenshot above should be 2.

GUI_GetStringDistX()

Description

Returns the X-size used to display a specified string in the currently selected font.

Prototype

```
int  GUI_GetStringDistX(const char * s);
```

Parameter	Description
`s`	Pointer to the string.

Table 11.20: GUI_GetStringDistX() parameter list

GUI_GetTextExtend()

Description

Calculates the pixel size in X required to draw the given string using the current font.

Prototype

```
void GUI_GetTextExtend(GUI_RECT * pRect, const char * s, int Len);
```

Parameter	Description
pRect	Pointer to GUI_RECT-structure to store result.
s	Pointer to the string.
Len	Number of characters of the string.

Table 11.21: GUI_GetTextExtend() parameter list

GUI_GetTrailingBlankCols()

Description

Returns the number of trailing blank pixel columns in the currently selected font for the given character.

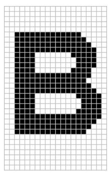
Prototype

```
int GUI_GetTrailingBlankCols(U16 c);
```

Parameter	Description
c	Character to be used.

Table 11.22: GUI_GetTrailingBlankCols() parameter list

Example



The result for the character ’B’ shown in the screenshot above should be 1.

GUI_GetYDistOfFont()

Description

Returns the Y-spacing of a particular font.

Prototype

```
int  GUI_GetYDistOfFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font.

Table 11.23: GUI_GetYDistOfFont() parameter list

Additional information

(see `GUI_GetFontDistY()`)

GUI_GetYSizeOfFont()

Description

Returns the Y-size of a particular font.

Prototype

```
int GUI_GetYSizeOfFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font.

Table 11.24: GUI_GetYSizeOfFont() parameter list

Additional information

Additional information can be found in the description of `GUI_GetFontSizeY()`.

GUI_IsInFont()

Description

Evaluates whether a particular font contains a specified character or not.

Prototype

```
char GUI_IsInFont(const GUI_FONT * pFont, U16 c);
```

Parameter	Description
pFont	Pointer to the font.
c	Character to be searched for.

Table 11.25: GUI_IsInFont() parameter list

Additional information

If the pointer `pFont` is set to 0, the currently selected font is used.

Example

Evaluates whether the font GUI_FontD32 contains an "X":

```
if (GUI_IsInFont(&GUI_FontD32, 'X') == 0) {
  GUI_DispString("GUI_FontD32 does not contain 'X'");
}
```

Return value

1, if the character was found.
0, if the character was not found.

GUI_SetDefaultFont()

Description

Sets the default font to be used after `GUI_Init()`.

Prototype

```
void GUI_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font to be used.

Table 11.26: GUI_SetDefaultFont() parameter list

11.8 Character sets

11.8.1 ASCII

emWin supports the full set of ASCII characters. These are the following 96 characters from 32 to 127:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2x		!		"#	$	%	&		'(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x		`a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Table 11.27: ASCII character set

Unfortunately, as ASCII stands for American Standard Code for Information Interchange, it is designed for American needs. It does not include any of the special characters used in European languages, such as Ä, Ö, Ü, á, à, and others. There is no single standard for these "European extensions" of the ASCII set of characters; several different ones exist. The one used on the Internet and by most Windows programs is ISO 8859-1, a superset of the ASCII set of characters.

11.8.2 ISO 8859-1 Western Latin character set

emWin supports the ISO 8859-1, which defines characters as listed below:

Code	Description	Char
160	non-breaking space	
161	inverted exclamation	¡
162	cent sign	¢
163	pound sterling	£
164	general currency sign	¤
165	yen sign	¥
166	broken vertical bar	¦
167	section sign	§
168	umlaut (dieresis)	¨
169	copyright	©
170	feminine ordinal	ª
171	left angle quote, guillemotleft	«
172	not sign	¬
173	soft hyphen	
174	registered trademark	®
175	macron accent	¯
176	degree sign	°
177	plus or minus	±
178	superscript two	²
179	superscript three	³
180	acute accent	´
181	micro sign	µ
182	paragraph sign	¶
183	middle dot	·
184	cedilla	¸
185	superscript one	¹
186	masculine ordinal	º
187	right angle quote, guillemot right	»
188	fraction one-fourth	¼
189	fraction one-half	½

Table 11.28: Western Latin character set

Code	Description	Char
190	fraction three-fourth	¾
191	inverted question mark	¿
192	capital A, grave accent	À
193	capital A, acute accent	Á
194	capital A, circumflex accent	Â
195	capital A, tilde	Ã
196	capital A, dieresis or umlaut mark	Ä
197	capital A, ring	Å
198	capital A, diphthong (ligature)	Æ
199	capital C, cedilla	Ç
200	capital E, grave accent	È
201	capital E, acute accent	É
202	capital E, circumflex accent	Ê
203	capital E, dieresis or umlaut mark	Ë
204	capital I, grave accent	Ì
205	capital I, acute accent	Í
206	capital I, circumflex accent	Î
207	capital I, dieresis or umlaut mark	Ï
208	Eth, Icelandic	Ð
209	N, tilde	Ñ
210	capital O, grave accent	Ò
211	capital O, acute accent	Ó
212	capital O, circumflex accent	Ô
213	capital O, tilde	Õ
214	capital O, dieresis or umlaut mark	Ö
215	multiply sign	×
216	capital O, slash	Ø
217	capital U, grave accent	Ù
218	capital U, acute accent	Ú
219	capital U, circumflex accent	Û
220	capital U, dieresis or umlaut mark	Ü
221	capital Y, acute accent	Ý
222	THORN, Icelandic	Þ
223	sharp s, German (s-z ligature)	ß
224	small a, grave accent	à
225	small a, acute accent	á
226	small a, circumflex accent	â
227	small a, tilde	ã
228	small a, dieresis or umlaut mark	ä
229	small a, ring	å
230	small ae diphthong (ligature)	æ
231	cedilla	ç
232	small e, grave accent	è
233	small e, acute accent	é
234	small e, circumflex accent	ê
235	small e, dieresis or umlaut mark	ë
236	small i, grave accent	ì
237	small i, acute accent	í
238	small i, circumflex accent	î
239	small i, dieresis or umlaut mark	ï
240	small eth, Icelandic	ð
241	small n, tilde	ñ
242	small o, grave accent	ò
243	small o, acute accent	ó
244	small o, circumflex accent	õ
245	small o, tilde	õ
246	small o, dieresis or umlaut mark	ö
247	division sign	÷
248	small o, slash	ø

Table 11.28: Western Latin character set

Code	Description	Char
249	small u, grave accent	ù
250	small u, acute accent	ú
251	small u, circumflex accent	û
252	small u, dieresis or umlaut mark	ü
253	small y, acute accent	ý
254	small thorn, Icelandic	þ
255	small y, dieresis or umlaut mark	ÿ

Table 11.28: Western Latin character set

11.8.3 Unicode

Unicode is the ultimate in character coding. It is an international standard based on ASCII and ISO 8859-1. Contrary to ASCII, UNICODE requires 16-bit characters because all characters have their own code. Currently, more than 30,000 different characters are defined. However, not all of the character images are defined in emWin. It is the responsibility of the user to define these additional characters.

11.9 Standard fonts

emWin is shipped with a selection of fonts which should cover most of your needs. The standard font package contains monospaced and proportional fonts in different sizes and styles. **Monospaced fonts** are fonts with a fixed character width, in which all characters have the same width in pixels. **Proportional fonts** are fonts in which each character has its own individual pixel-width.
The following sections provide an overview of emWin standard fonts.

11.9.1 Font identifier naming convention

All standard fonts are named as follows. The components of the naming convention are explained in the table below:

```
GUI_Font[<style>][<width>x]<height>[x<MagX>x<MagY>][H][B][_<characterset>]
```

Element	Description
GUI_Font	Standard prefix for all fonts shipped with emWin.
<style>	Specifies a non-standard font style. Example: Comic style in GUI_FontComic18B_ASCII.
<width>	Width of characters, contained only in monospaced fonts.
<height>	Height of the font in pixels.
<MagX>	Factor of magnification in X, contained only in magnified fonts.
<MagY>	Factor of magnification in Y, contained only in magnified fonts.
H	Abbreviation for "high". Only used if there is more than one font with the same height. It means that the font appears "higher" than other fonts.
B	Abbreviation for "bold". Used in bold fonts.
<characterset>	Specifies the contents of characters: ASCII: Only ASCII characters 0x20-0x7E (0x7F). 1: ASCII characters and European extensions 0xA0 - 0xFF. HK: Hiragana and Katakana. 1HK: ASCII, European extensions, Hiragana and Katakana. D: Digit fonts, character set: +-.0123456789.

Table 11.29: Font identifier naming convention

Example 1

GUI_Font16_ASCII

Element	Description
GUI_Font	Standard font prefix.
16	Height in pixels.
ASCII	Font contains ASCII characters only.

Table 11.30: GUI_Font16_ASCII name component list

Example 2

GUI_Font8x15B_ASCII

Element	Description
GUI_Font	Standard font prefix.
8	Width of characters.
x15	Height in pixels.
B	Bold font.
ASCII	Font contains ASCII characters only.

Table 11.31: GUI_Font8x15B_ASCII name component list

Example 3

GUI_Font8x16x1x2

Element	Description
GUI_Font	Standard font prefix.
8	Width of characters.
x16	Height in pixels.
x1	Magnification factor in X.
x2	Magnification factor in Y.

Table 11.32: GUI_Font8x16x1x2 name component list

11.9.2 Font file naming convention

The names for the font files are similar to the names of the fonts themselves. The files are named as follows:
F[<width>]<height>[H][B][<characterset>]

Element	Description
F	Standard prefix for all fonts files shipped with emWin.
<width>	Width of characters, contained only in monospaced fonts.
<height>	Height of the font in pixels.
H	Abbreviation for "high". Only used if there is more than one font with the same height. It means that the font appears "higher" than other fonts.
B	Abbreviation for "bold". Used in bold fonts.
<characterset>	Specifies the contents of characters: ASCII: Only ASCII characters 0x20-0x7E (0x7F). 1: ASCII characters and European extensions 0xA0 - 0xFF. HK: Hiragana and Katakana. 1HK: ASCII, European extensions, Hiragana and Katakana. D: Digit fonts.

Table 11.33: Font file naming convention

11.9.3 Measurement, ROM-size and character set of fonts

The following sections describe the standard fonts shipped with emWin. For each font there is a measurement diagram, an overview of all characters included and a table containing the ROM size in bytes and the font files required for use.

The following parameters are used in the measurement diagrams:

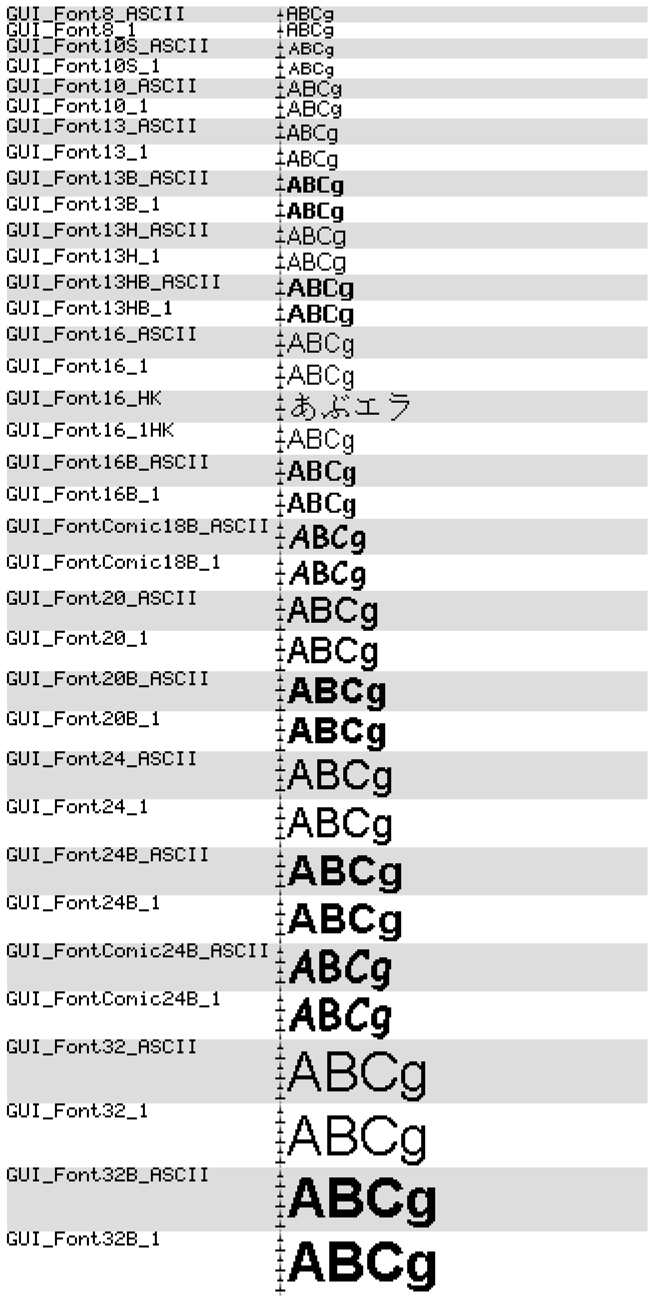
Element	Description
F	Size of font in Y.
B	Distance of base line from the top of the font.
C	Height of capital characters.
L	Height of lowercase characters.
U	Size of underlength used by letters such as "g", "j" or "y".

Table 11.34: Font measurement

11.9.4 Proportional fonts

11.9.4.1 Overview

The following screenshot gives an overview of all available proportional fonts:



11.9.4.2 Font details

The following table shows the measurement, ROM size and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_Font8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1562	F08_ASCII.c
GUI_Font8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1562+ 1586	F08_ASCII.c F08_1.c
GUI_Font10_ASCII	F: 10, B: 9, C: 8, L: 6, U: 1	1800	F10_ASCII.c
GUI_Font10_1	F: 10, B: 9, C: 8, L: 6, U: 1	1800+ 2456	F10_ASCII.c F10_1.c
GUI_Font10S_ASCII	F: 10, B: 8, C: 6, L: 4, U: 2	1760	F10S_ASCII.c
GUI_Font10S_1	F: 10, B: 8, C: 6, L: 4, U: 2	1760+ 1770	F10S_ASCII.c F10S_1.c
GUI_Font13_ASCII	F: 13, B: 11, C: 8, L: 6, U: 2	2076	F13_ASCII.c
GUI_Font13_1	F: 13, B: 11, C: 8, L: 6, U: 2	2076+ 2149	F13_ASCII.c F13_1.c
GUI_Font13B_ASCII	F: 13, B: 11, C: 8, L: 6, U: 2	2222	F13B_ASCII.c
GUI_Font13B_1	F: 13, B: 11, C: 8, L: 6, U: 2	2222+ 2216	F13B_ASCII.c F13B_1.c
GUI_Font13H_ASCII	F: 13, B: 11, C: 9, L: 7, U: 2	2232	F13H_ASCII.c
GUI_Font13H_1	F: 13, B: 11, C: 9, L: 7, U: 2	2232+ 2291	F13H_ASCII.c F13H_1.c
GUI_Font13HB_ASCII	F: 13, B: 11, C: 9, L: 7, U: 2	2690	F13HB_ASCII.c
GUI_Font13HB_1	F: 13, B: 11, C: 9, L: 7, U: 2	2690+ 2806	F13HB_ASCII.c F13HB_1.c
GUI_Font16_ASCII	F: 16, B: 13, C: 10, L: 7, U: 3	2714	F16_ASCII.c
GUI_Font16_1	F: 16, B: 13, C: 10, L: 7, U: 3	2714+ 3850	F16_ASCII.c F16_1.c
GUI_Font16_HK	F: 16, B: 13, C: 10, L: 7, U: 3	6950	F16_HK.c
GUI_Font16_1HK	F: 16, B: 13, C: 10, L: 7, U: 3	120+ 6950+ 2714+ 3850	F16_1HK.c F16_HK.c F16_ASCII.c F16_1.c
GUI_Font16B_ASCII	F: 16, B: 13, C: 10, L: 7, U: 3	2690	F16B_ASCII.c
GUI_Font16B_1	F: 16, B: 13, C: 10, L: 7, U: 3	2690+ 2790	F16B_ASCII.c F16B_1.c
GUI_FontComic18B_ASCII	F: 18, B: 15, C: 12, L: 9, U: 3	3572	FComic18B_ASCII.c
GUI_FontComic18B_1	F: 18, B: 15, C: 12, L: 9, U: 3	3572+ 4334	FComic18B_ASCII.c FComic18B_1.c
GUI_Font20_ASCII	F: 20, B: 16, C: 13, L: 10, U: 4	4044	F20_ASCII.c
GUI_Font20_1	F: 20, B: 16, C: 13, L: 10, U: 4	4044+ 4244	F20_ASCII.c F20_1.c
GUI_Font20B_ASCII	F: 20, B: 16, C: 13, L: 10, U: 4	4164	F20B_ASCII.c
GUI_Font20B_1	F: 20, B: 16, C: 13, L: 10, U: 4	4164+ 4244	F20B_ASCII.c F20B_1.c
GUI_Font24_ASCII	F: 24, B: 20, C: 17, L: 13, U: 4	4786	F24_ASCII.c

Table 11.35: Font details

Font name	Measurement	ROM size in bytes	Used files
GUI_Font24_1	F: 24, B: 20, C: 17, L: 13, U: 4	4786+ 5022	F24_ASCII.c F24_1.c
GUI_Font24B_ASCII	F: 24, B: 19, C: 15, L: 11, U: 5	4858	F24B_ASCII.c
GUI_Font24B_1	F: 24, B: 19, C: 15, L: 11, U: 5	4858+ 5022	F24B_ASCII.c F24B_1.c
GUI_FontComic24B_ASCII	F: 24, B: 20, C: 17, L: 13, U: 4	6146	FComic24B_ASCII.c
GUI_FontComic24B_1	F: 24, B: 20, C: 17, L: 13, U: 4	6146+ 5598	FComic24B_ASCII.c FComic24B_1.c
GUI_Font32_ASCII	F: 32, B: 26, C: 20, L: 15, U: 6	7234	F32_ASCII.c
GUI_Font32_1	F: 32, B: 26, C: 20, L: 15, U: 6	7234+ 7734	F32_ASCII.c F32_1.c
GUI_Font32B_ASCII	F: 32, B: 25, C: 20, L: 15, U: 7	7842	F32B_ASCII.c
GUI_Font32B_1	F: 32, B: 25, C: 20, L: 15, U: 7	7842+ 8118	F32B_ASCII.c F32B_1.c

Table 11.35: Font details

11.9.4.3 Characters

The following shows all characters of all proportional standard fonts:

GUI_Font8_ASCII

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

GUI_Font8_1

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹
º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæç
èéêëìíîïðñòóôõö÷øùúûüýþÿ
```

GUI_Font10_ASCII

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUV
WXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

GUI_Font10_1

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUV
WXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³
´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâã
äåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ
```

GUI_Font10S_ASCII

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdef
ghijklmnopqrstuvwxyz{|}~
```

GUI_Font10S_1

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdef
ghijklmnopqrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓ
ÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ
```

GUI_Font13_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

GUI_Font13_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-
®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜ
ÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font13B_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
~

GUI_Font13B_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊ
ËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷ø
ùúûüýþÿ

GUI_Font13H_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
}~

GUI_Font13H_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇ
ÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõ
ö÷øùúûüýþÿ

GUI_Font13HB_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGH
IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopq
rstuvwxyz{|}~

GUI_Font13HB_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGH
IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopq
rstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»
¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞ
ßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font16_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
~

GUI_Font16_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊ
ËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö
÷øùúûüýþÿ

GUI_Font16_HK

ぁあぃいぅうぇえぉおかがきぎくぐけげこご
さざしじすずせぜそぞただちぢっつづてでと
どなにぬねのはばぱひびぴふぶぷへべぺほぼ
ぽまみむめもゃやゅゆょよらりるれろゎわゐ
ゑをんァアィイゥウェエォオカガキギクグケ
ゲコゴサザシジスズセゼソゾタダチヂッツヅ
テデトドナニヌネノハバパヒビピフブプヘベ
ペホボポマミムメモャヤュユョヨラリルレロ
ヮワヰヱヲンヴヵヶ

GUI_Font16_1HK

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}
~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊ
ËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö
÷øùúûüýþÿぁあぃいぅうぇえぉおかがきぎくぐ
けげこごさざしじすずせぜそぞただちぢっつ
づてでとどなにぬねのはばぱひびぴふぶぷへ
べぺほぼぽまみむめもゃやゅゆょよらりるれ
ろゎわゐゑをんァアィイゥウェエォオカガキ
ギクグケゲコゴサザシジスズセゼソゾタダチ
ヂッツヅテデトドナニヌネノハバパヒビピフ
ブプヘベペホボポマミムメモャヤュユョヨラ
リルレロヮワヰヱヲンヴヵヶ

GUI_Font16B_ASCII

**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
xyz{|}~**

GUI_Font16B_1

**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
xyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆ
ÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîï
ðñòóôõö÷øùúûüýþÿ**

GUI_FontComic18B_ASCII

**!"#$%&'()*+,-./0123456789:;<=>?@ABCD
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`ab
cdefghijklmnopqrstuvwxyz{|}~□€**

GUI_FontComic18B_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCD
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`ab
cdefghijklmnopqrstuvwxyz{|}~ ¡¢£¤¥¦§¨
©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇ
ÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäå
æçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font20_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@AB
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^
_`abcdefghijklmnopqrstuvwxyz{|}~

GUI_Font20_1

!"#$%&'()*+,-./0123456789:;<=>?@AB
CDEFGHIJKLMNOPQRSTUVWXYZ[\]^
_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¤
¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅ
ÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßà
áâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font20B_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@A
BCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz{|}
~

GUI_Font20B_1

!"#$%&'()*+,-./0123456789:;<=>?@A
BCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz{|}
~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿À
ÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚ
ÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øù
úûüýþÿ

GUI_Font24_ASCII

!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTU
VWXYZ[\]^_`abcdefghijklmnopqrst
uvwxyz{|}~

GUI_Font24_1

!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTU
VWXYZ[\]^_`abcdefghijklmnopqrst
uvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ
¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäå
æçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font24B_ASCII

!"#$%&'()*+,-./0123456789:;<=>
?@ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_`abcdefghijklmnop
qrstuvwxyz{|}~

GUI_Font24B_1

!"#$%&'()*+,-./0123456789:;<=>
?@ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_`abcdefghijklmnop
qrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯
°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈ
ÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàá
âãäåæçèéêëìíîïðñòóôõö÷øùúûü
ýþÿ

GUI_FontComic24B_ASCII

!"#$%&'()*+,-./0123456789:
;<=>?@ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`abcdefghi
jklmnopqrstuvwxyz{|}~□

GUI_FontComic24B_1

!"#$%&'()*+,-./0123456789:
;<=>?@ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`abcdefghi
jklmnopqrstuvwxyz{|}~ ¡¢£¤¥¦
§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾
¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔ
ÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëì
íîïðñòóôõö÷øùúûüýþÿ

GUI_Font32_ASCII

!"#$%&'()*+,-./012345678
9:;<=>?@ABCDEFGHIJK
LMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuv
wxyz{|}~

GUI_Font32_1

!"#$%&'()*+,-./012345678
9:;<=>?@ABCDEFGHIJK
LMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuv
wxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°
±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅ
ÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×
ØÙÚÛÜÝÞßàáâãäåæçèé
êëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font32B_ASCII

 !"#$%&'()*+,-./01234567
89:;<=>?@ABCDEFGHIJ
KLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopq
rstuvwxyz{|}~

GUI_Font32B_1

 !"#$%&'()*+,-./01234567
89:;<=>?@ABCDEFGHIJ
KLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopq
rstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª
«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿À
ÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒ
ÓÔÕÖ×ØÙÚÛÜÝÞßàáâã
äåæçèéêëìíîïðñòóôõö÷ø
ùúûüýþÿ

11.9.5 Proportional fonts, framed

11.9.5.1 Overview

The following screenshot shows the currently available framed proportional fonts:

11.9.5.2 Font details

The following table shows the measurement, ROM size and used file of the font:

Font name	Measurement	ROM size in bytes	Used files
GUI_Font20F_ASCII	F: 20, B: 19, C: 19, L: 19, U: 1	5248	F20F_ASCII.c

Table 11.36: Font details

11.9.5.3 Characters

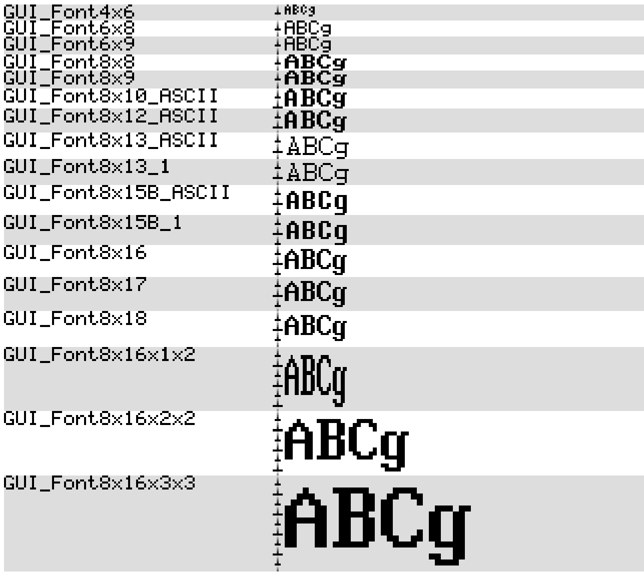
The following shows all characters of the font:

GUI_Font20F_ASCII

11.9.6 Monospaced fonts

11.9.6.1 Overview

The following screenshot gives an overview of all available monospaced fonts:



11.9.6.2 Font details

The following table shows the measurement, ROM size and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_Font4x6	F: 6, B: 5, C: 5, L: 4, U: 1	620	F4x6.c
GUI_Font6x8	F: 8, B: 7, C: 7, L: 5, U: 1	1840	F6x8.c
GUI_Font6x8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1568	F6x8_ASCII.c
GUI_Font6x8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1568+ 1584	F6x8_ASCII.c F6x8_1.c
GUI_Font6x9	F: 9, B: 7, C: 7, L: 5, U: 2	1840 (same ROM location as GUI_Font6x8)	F6x8.c
GUI_Font8x8	F: 8, B: 7, C: 7, L: 5, U: 1	1840	F8x8.c
GUI_Font8x8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1568	F8x8_ASCII.c
GUI_Font8x8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1568+ 1584	F8x8_ASCII.c F8x8_1.c
GUI_Font8x9	F: 9, B: 7, C: 7, L: 5, U: 2	1840 (same ROM location as GUI_Font8x8)	F8x8.c
GUI_Font8x10_ASCII	F: 10, B: 9, C: 9, L: 7, U: 1	1770	F8x10_ASCII.c
GUI_Font8x12_ASCII	F: 12, B: 10, C: 9, L: 6, U: 2	1962	F8x12_ASCII.c
GUI_Font8x13_ASCII	F: 13, B: 11, C: 9, L: 6, U: 2	2058	F8x13_ASCII.c
GUI_Font8x13_1	F: 13, B: 11, C: 9, L: 6, U: 2	2058+ 2070	F8x13_ASCII.c F8x13_1.c
GUI_Font8x15B_ASCII	F: 15, B: 12, C: 9, L: 7, U: 3	2250	F8x15_ASCII.c
GUI_Font8x15B_1	F: 15, B: 12, C: 9, L: 7, U: 3	2250+ 2262	F8x15B_ASCII.c F8x15B_1.c

Table 11.37: Font details

Font name	Measurement	ROM size in bytes	Used files
GUI_Font8x16	F: 16, B: 12, C: 10, L: 7, U: 4	3304	F8x16.c
GUI_Font8x17	F: 17, B: 12, C: 10, L: 7, U: 5	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x18	F: 18, B: 12, C: 10, L: 7, U: 6	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16x1x2	F: 32, B: 24, C: 20, L: 14, U: 8	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16x2x2	F: 32, B: 24, C: 20, L: 14, U: 8	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16x3x3	F: 48, B: 36, C: 30, L: 21, U: 12	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16_ASCII	F: 16, B: 12, C: 10, L: 7, U: 4	2328	F8x16_ASCII.c
GUI_Font8x16_1	F: 16, B: 12, C: 10, L: 7, U: 4	2328+ 2352	F8x16_ASCII.c F8x16_1.c

Table 11.37: Font details

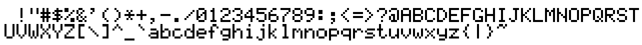
11.9.6.3 Characters

The following shows all characters of all monospaced standard fonts:

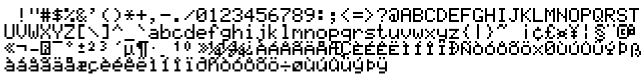
GUI_Font4x6

GUI_Font6x8

GUI_Font6x8_ASCII

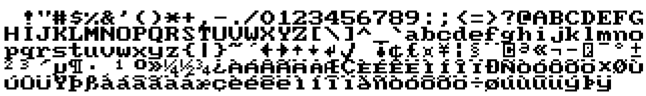


GUI_Font6x8_1



GUI_Font6x9

GUI_Font8x8



GUI_Font8x8_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~

GUI_Font8x8_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·
¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßà
áâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x9

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~´←→↑↓↵√ ¡¢£¤¥¦§¨©ª«¬-®¯°±
²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙ
ÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x10_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~⌂

GUI_Font8x12_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~⌂

GUI_Font8x13_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~▮

GUI_Font8x13_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~▮ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·
¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x15B_ASCII

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~█

GUI_Font8x15B_1

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~█ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·
¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x16

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~´←→↑↓↲✓ ¡¢£¤¥¦§¨©ª«¬-®¯°±
²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙ
ÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x17

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~´←→↑↓↲✓ ¡¢£¤¥¦§¨©ª«¬-®¯°±
²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙ
ÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

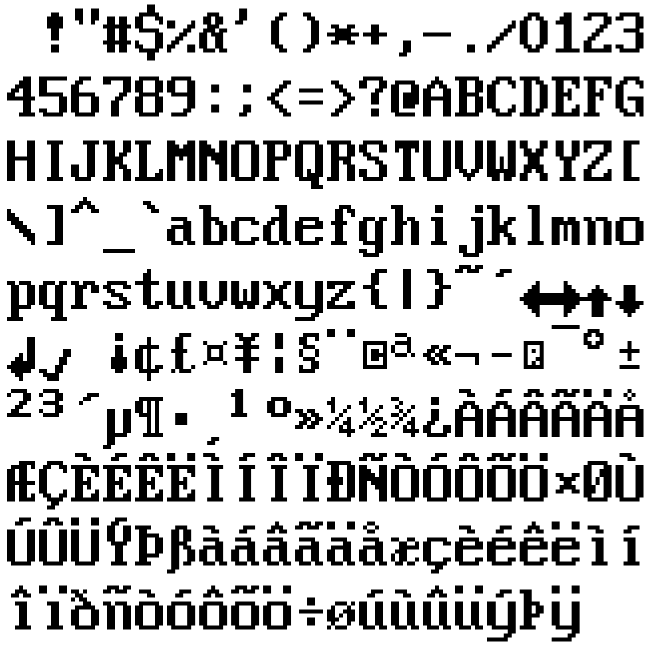
GUI_Font8x18

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~´←→↑↓↲✓ ¡¢£¤¥¦§¨©ª«¬-®¯°±
²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙ
ÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x16x1x2

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~´←→↑↓↲✓ ¡¢£¤¥¦§¨©ª«¬-®¯°±
²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙ
ÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

GUI_Font8x16x2x2



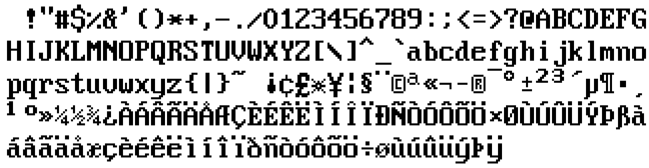
GUI_Font8x16x3x3

 !"#$%&'()*+,
-./0123456789
:;<=>?@ABCDEF
GHIJKLMNOPQRS
TUVWXYZ[\]^_`
abcdefghijklm
nopqrstuvwxyz
{|}~´←→↑↓↲✓ ¡
¢£¤¥¦§¨©ª«¬-®
¯°±²³´µ¶·¸¹º»
¼½¾¿ÀÁÂÃÄÅÆÇÈ
ÉÊËÌÍÎÏÐÑÒÓÔÕ
Ö×ØÙÚÛÜÝÞßàáâ
ãäåæçèéêëìíîï
ðñòóôõö÷øùúûü
ýþÿ

GUI_Font8x16_ASCII

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~

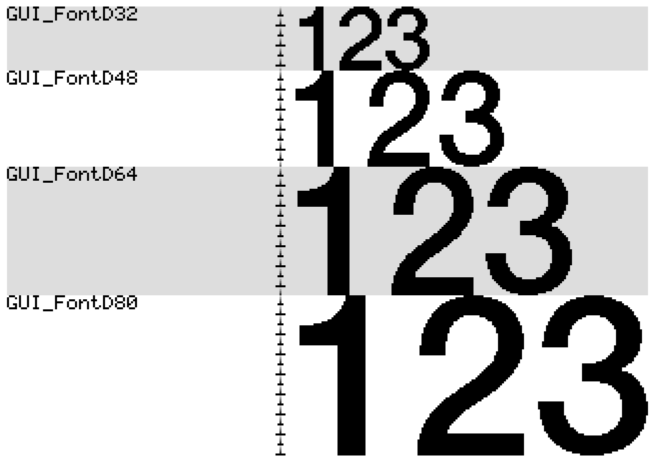
GUI_Font8x16_1



11.9.7 Digit fonts (proportional)

11.9.7.1 Overview

The following screenshot gives an overview of all available proportional digit fonts:



11.9.7.2 Font details

The following table shows the measurement, ROM size and used files of the fonts:

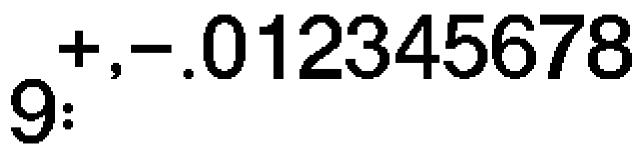
Font name	Measurement	ROM size in bytes	Used files
GUI_FontD32	F: 32, C: 31	1574	FD32.c
GUI_FontD48	F: 48, C: 47	3512	FD48.c
GUI_FontD64	F: 64, C: 63	5384	FD64.c
GUI_FontD80	F: 80, C: 79	8840	FD80.c

Table 11.38: Font details

11.9.7.3 Characters

The following shows all characters of all proportional digit fonts:

GUI_FontD32



GUI_FontD48

+,-.01234
56789:

GUI_FontD64

+,-.012
345678
9:

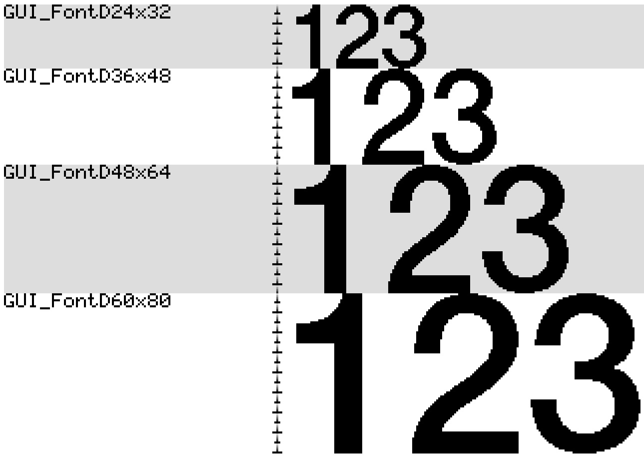
GUI_FontD80

+,-.0
12345
6789:

11.9.8 Digit fonts (monospaced)

11.9.8.1 Overview

The following screenshot gives an overview of all available monospaced digit fonts:



11.9.8.2 Font details

The following table shows the measurement, ROM size and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_FontD24x32	F: 32, C: 31	1606	FD24x32.c
GUI_FontD36x48	F: 48, C: 47	3800	FD36x48.c
GUI_FontD48x64	F: 64, C: 63	5960	FD48x60.c
GUI_FontD60x80	F: 80, C: 79	9800	FD60x80.c

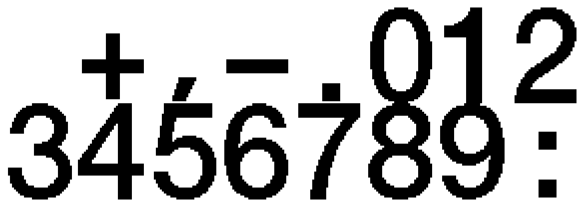
Table 11.39: Font details

11.9.8.3 Characters

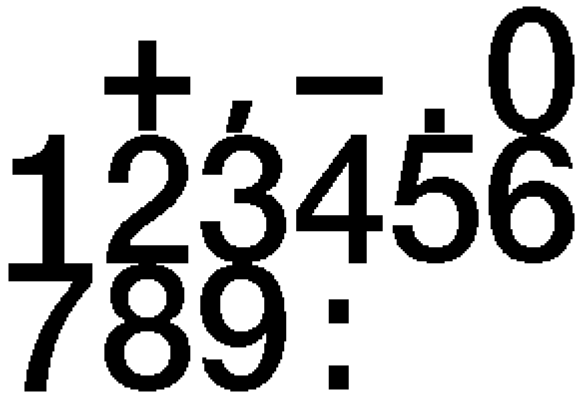
The following shows all characters of all monospaced digit fonts:

GUI_FontD24x32

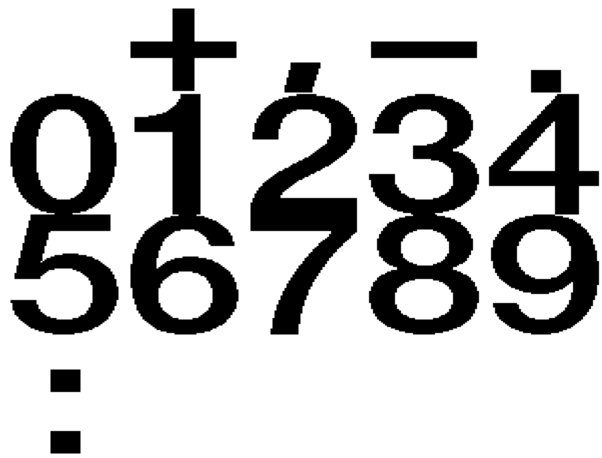
GUI_FontD36x48



GUI_FontD48x64



GUI_FontD60x80



Chapter 12

Font Converter

The Font Converter is a Windows program which allows convenient converting of any PC installed font into an emWin (bitmap) font which can be easily integrated into emWin based applications. PC installed fonts may be protected by copyright or any other intellectual property right of their legal owner. emWin fonts should be defined either as GUI_FONT structures in C files or should exist as binary files containing System Independent Fonts (SIF) or External Bitmap Font (XBF). Manual creation of those fonts is possible, but since this would be very time-consuming and inefficient, it is recommended to use the Font Converter instead.

The Font Converter is not part of the emWin Basic package. The full version has to be purchased separately. The emWin Basic package comes with the demo version of the Font Converter which provides full functionality but accurate storage of pixel data. Nevertheless the structure of C file fonts is stored well, so one might have a look at it in order to estimate the possibly saved effort by using the Font Converter.

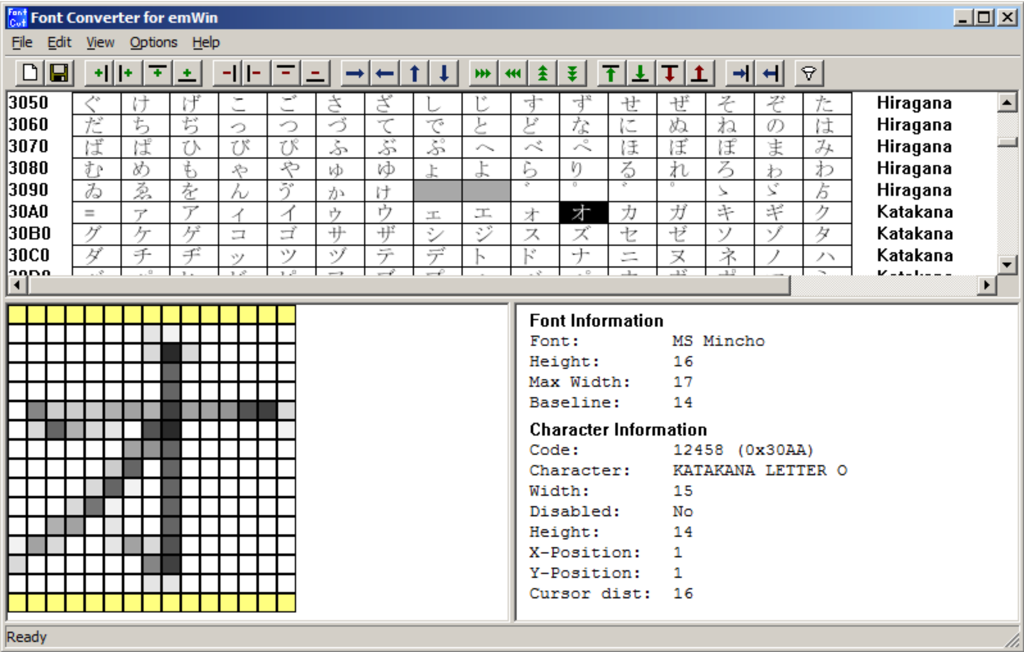
The Font Converter does not come with any fonts or a permission or license to use any PC installed font for converting purposes. It is users sole responsibility to not infringe upon any third party intellectual property right by making use of the fonts in its application and obtain a license if required by the legal owner of the font.

12.1 Requirements

The Font Converter is a Windows program, so it can be used only within a windows environment. The source fonts need to meet the following requirements:

- The font is installed in Windows.
- The font is usable in Windows. (e.g. in MS Word)
- The 'Character To Glyph Index Mapping Table' (cmap) of a font file needs at least a legal subtable with a platform ID == 3 (Windows) and an encoding ID == 1 (Unicode).

Screenshot of the Font Converter with a loaded font

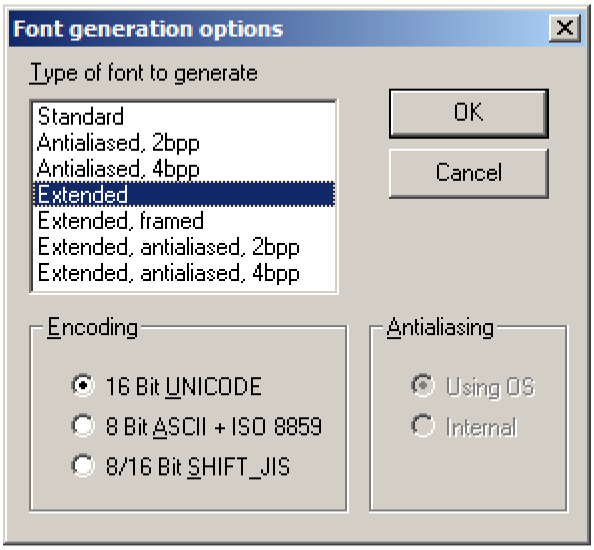


12.2 Creating an emWin font file from a Windows font

The basic procedure for using the Font Converter for creating an emWin font file from an installed Windows font is illustrated below. The steps are explained in detail in the following sections.

Step 1: Application start

As the Font Converter is started it immediately shows the "Font generation options" dialog box. The same dialog box appears if `File/New` (CRTL+N) is chosen from the Font Converter menu bar.



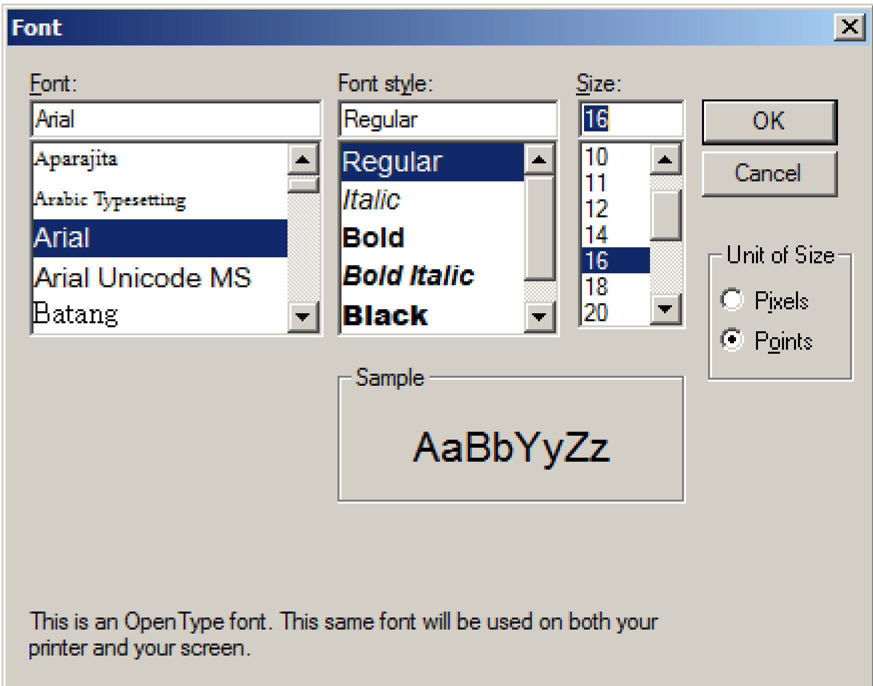
Step 2: Specifying the type of font

In the above screenshot, a font is to be generated in extended mode using Unicode 16 Bit encoding. The antialiasing option can be modified only in case a font type is selected which supports antialiasing.

Step 3: Specifying the actual font.

After confirming the "Font genration options", the "Font" dialog is opened which allows chosing a font, its style and size.
The fonts shown in the selection list depend on MS Windows. In case an installed font is not shown, it might be required to change Windows font settings. Detailed information can be found in the section "Troubleshooting" on page 271.

In this example, a regular-style, 16 pixel Arial font is chosen.



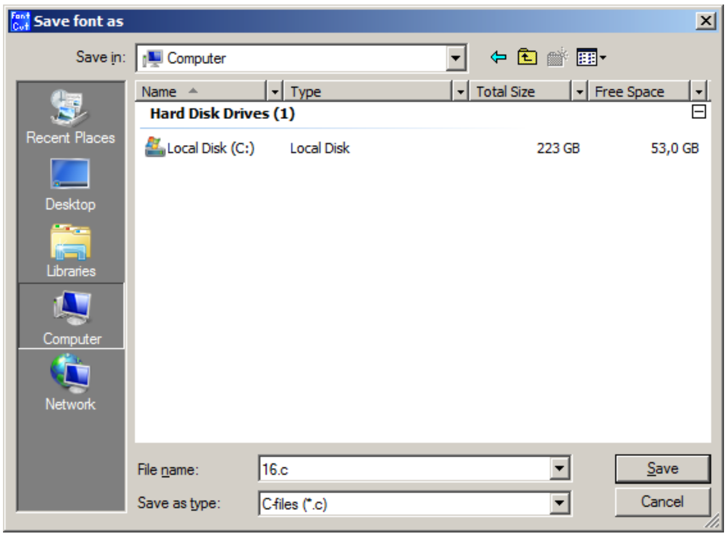
Step 4: Modifying the font

Detail information on the possibilities to modify the currently loaded font can be found in the section "User Interface" on page 259.

Step 5: Saving the font file

Chosing `File/Save As...` from the menu bar opens the "Save font as" dialog which is shown below. In this dialog the desired format of the font file can be selected:

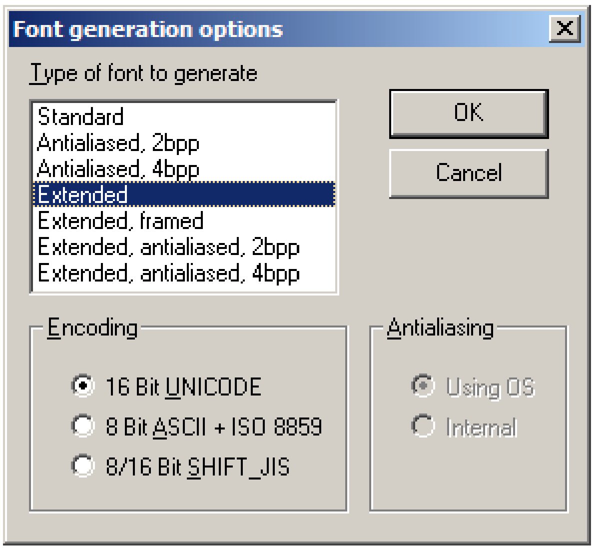
- C file (*.c)
- System independent font (*.sif)
- External binary font (*.xbf)



Clicking the "Save" button creates a font file of the selected format at the current location using the specified file name. Detailed information about the font types and how to use them can be found in the chapter “Fonts” on page 207.

12.3 Font generation options dialog

After starting the program or when choosing the menu point `File/New`, the following dialog automatically occurs:



The selections made here will determine the output mode of the generated font, how it is to be encoded, and how it will be antialiased in case an antialiased output mode is selected.

12.3.1 Type of font to generate

Type	Description
Standard	Creates a 1 bit per pixel font without antialiasing.
Antialiased 2bpp	Creates an antialiased font using 2 bits per pixel.
Antialiased 4bpp	Creates an antialiased font using 4 bits per pixel.
Extended	Creates a non antialiased 1 bit per pixel font with extended character information. This type supports compound characters like they are used in Thai language.
Extended framed	Creates a non antialiased 1 bit per pixel font with extended character information with a surrounding frame. A framed font is always drawn in transparent mode regardless of the current settings. The character pixels are drawn in the currently selected foreground color and the frame is drawn in background color.
Extended antialiased 2bpp	Creates an antialiased 2 bit per pixel font with extended character information. Each character has the same height and its own width. The pixel information is saved with 2bpp antialiasing information and covers only the areas of the glyph bitmaps.
Extended antialiased 4bpp	Creates an antialiased 4 bit per pixel font with extended character information. Each character has the same height and its own width. The pixel information is saved with 4bpp antialiasing information and covers only the areas of the glyph bitmaps.

Table 12.1: Font types

12.3.2 Encoding

Encoding	Description
Unicode 16 Bit	With Unicode encoding, you have access to all characters of a font. Windows font files contain a maximum of 65536 characters. All character codes of the C file are the same as those in the Windows font file.
ASCII 8 Bit + ISO 8859	This encoding mode includes the ASCII codes (0x20 - 0x7F) and the ISO 8859 characters (0xA0 - 0xFF).
SHIFT JIS 8/16 Bit	Shift JIS (Japanese Industry Standard) enables mapping from Unicode to Shift JIS in accordance with the Unicode standard 2. For example, the Katakana letter “KU” is shifted from its Unicode value of 0x30AF to the Shift JIS value of 0x834E, the Kanji character 0x786F is shifted to 0x8CA5 and so on.

Table 12.2: Font encoding

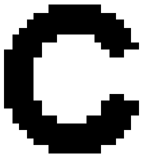
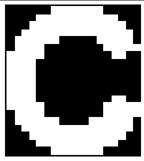
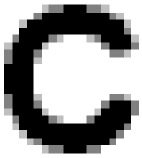
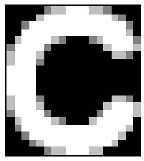
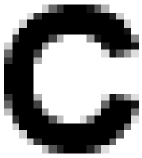
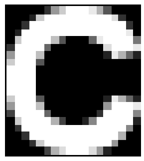
12.3.3 Antialiasing

You can choose between two ways of antialiasing. This choice only applies when an antialiased font type has been selected.

Antialiasing	Description
Using OS	The operating system is used to do the antialiasing. The resulting characters appear exactly the same as in any other windows application where antialiased characters are displayed.
Internal	The internal antialiasing routines of the Font Converter are used to do the antialiasing. The resulting characters are more exact with regard to proportions.

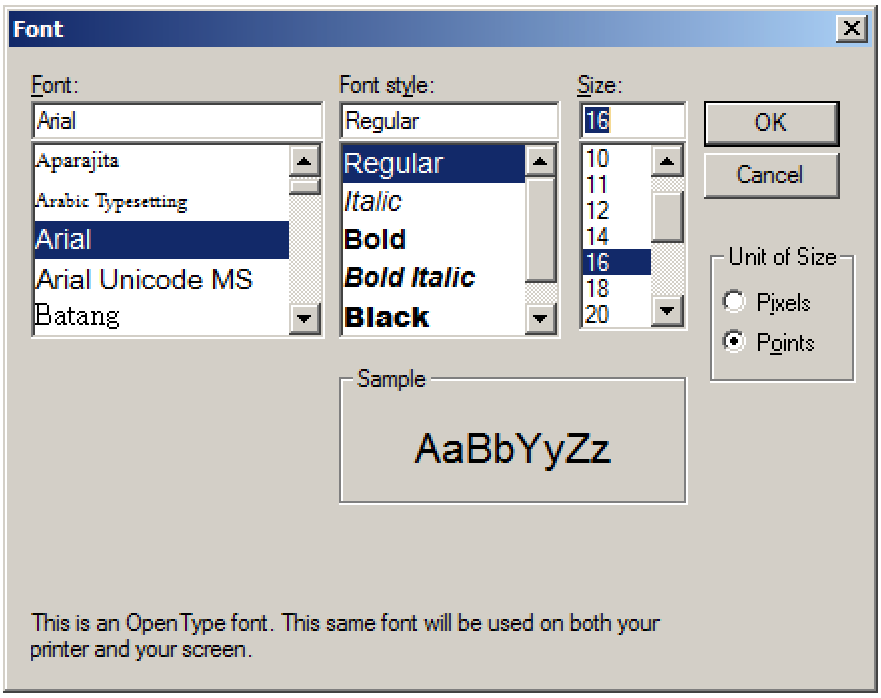
Table 12.3: Font antialiasing

Showcase

Font Type	Black On White	White On Black
Standard (no antialiasing) 1 bpp 2 shades		
Low-quality (antialiased) 2 bpp 4 shades		
High-quality (antialiased) 4 bpp 16 shades		

12.4 Font Dialog

The Font Dialog appears after the font generation options have been confirmed:



This dialog allows selecting a font which has to be used in the target application. Confirming this dialog using the 'OK' button makes the Font Converter load the included characters and display it in the main user interface.

Warning: Fonts which are legally owned by third parties may require a valid license in order to use them in a target system. emWin does not come with licenses for third party fonts.

All fonts included in emWin can be used according to the license under which emWin was provided. Detailed listings of those fonts and the included character sets can be found under "Standard fonts" on page 230.

12.4.1 Font, Font Style, and Size

These menus are used to select the particular font to be converted. The size of the font is specified in pixels.

12.4.2 Script

The Script box is used to select the character set which should be mapped down from Unicode into the first 256 characters in accordance with ISO 8859. It only applies when using the 8 Bit ASCII + ISO 8859 encoding mode.

12.4.3 Unit of Size

This option button can be used to set 'Points' or 'Pixels' as measuring unit. Please note that emWin does not know something about the unit 'Points' whereas most of other PC applications use the point size for specifying the font size. The Font Converter uses the operating system for getting the desired font resource. Please note that the font mapper of the operating system is not able to create each font in each desired pixel height. In these cases the font mapper of the operating system creates the nearest possible pixel height. This is not a bug of the Font Converter.

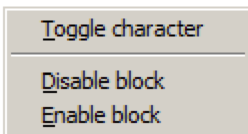
12.5 User Interface

After clicking OK in the Font dialog box, the main user interface of the Font Converter appears, loaded with the previously selected font. You may convert the font into a C file immediately if you wish or edit its appearance first.
The Font Converter is divided into two areas. In the upper area, all font characters appear scaled 1:1 as they will be displayed on your target device. Disabled characters are shown with a gray background. Per default all character codes which are not included in the chosen font are disabled. For example, many fonts do not include character codes from 0x00 to 0x1F and 0x7F to 0x9F, so these codes are grayed.
The current character is displayed in a magnified scale on the left side of the lower area. Additional information about the font and the current character can be seen on the right side. If you want to modify the character data, you must first activate the lower area, either by pressing the `<TAB>` key or by simply clicking in the area.

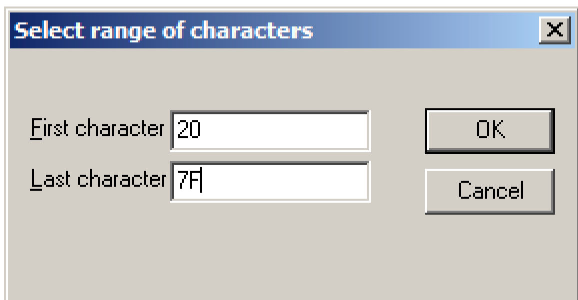
12.5.1 Selecting the current character

Characters may be selected:
- by using the keys `<UP>`, `<DOWN>`, `<LEFT>`, `<RIGHT>`, `<PGUP>`, `<PGDOWN>`, `<POS1>` or `<END>`
- by using the scroll bars
- by clicking a character with the left mouse button

12.5.2 Toggling character status



Use the right mouse button to toggle the status of a specific character or to enable/disable an entire row of characters. The menu point `Edit/Toggle` activation as well as the `<SPACE>` key will toggle the status of the current character.

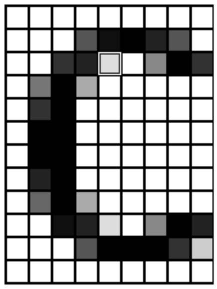


If you need to change the status of a particular range of characters, choose `Edit/Enable range of characters` or `Edit/Disable range of characters` from the menu. The range to be enabled or disabled is then specified in a dialog box using hexadecimal character values. To disable all characters, select `Edit/Disable all characters` from the menu.

12.5.3 Selecting pixels

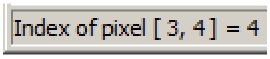
When the lower area of the user interface is activated, you can move through the pixels with the cursor, either by using the `<UP>`, `<DOWN>`, `<LEFT>` and `<RIGHT>` keys or by clicking on the pixels with the left mouse button.

12.5.4 Modifying character bits



In the lower area you can use the `<SPACE>` key to invert the currently selected bit. In antialiased mode, you can increase and decrease the intensity of a pixel with the keys `<+>` and `<->`.

The status bar displays the intensity of the current pixel as follows:



12.5.5 Operations

The following size / shift / move operations are available:

Size operations

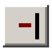
The size of a character (the font) may be modified by selecting `Edit/Insert/Right`, `Left`, `Top`, `Bottom` or `Edit/Delete/Right`, `Left`, `Top`, `Bottom` from the menu, or by using the toolbar:

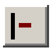
 Add one pixel to the right

 Add one pixel to the left

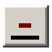
 Add one pixel at the top

 Add one pixel at the bottom

 Delete one pixel from the right

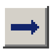
 Delete one pixel from the left

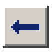
 Delete one pixel at the top

 Delete one pixel at the bottom

Shift operations

Choose `Edit/Shift/Right`, `Left`, `Up`, `Down` from the menu to shift the bits of the current character in the respective direction, or use the toolbar:

 Shift all pixels right

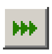
 Shift all pixels left

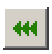
 Shift all pixels up

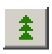
 Shift all pixels down

Move operations (extended font format only)

Choose `Edit/Move/Right`, `Left`, `Up`, `Down` from the menu to move the character position in the respective direction, or use the toolbar:

 Move image to the right

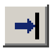
 Move image to the left

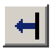
 Move image up

 Move image down

Change cursor distance (extended font format only)

Choose `Edit/Cursor distance/Increase`, Decrease from the menu to move the character position in the respective direction, or use the toolbar:

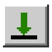
 Increase cursor distance

 Decrease cursor distance

Change font height (extended font format only)

Choose `Edit/Font height/[Insert, Delete] [top, bottom]` from the menu to add or remove a row to or from the font, or use the toolbar:

 Insert a row at the top of the font

 Insert a row at the bottom of the font

 Delete a row from the top of the font

 Delete a row from the bottom of the font

12.5.5.1 Modifying the viewing mode

The view mode may be changed by selecting the following options from the menu:

View/All Characters

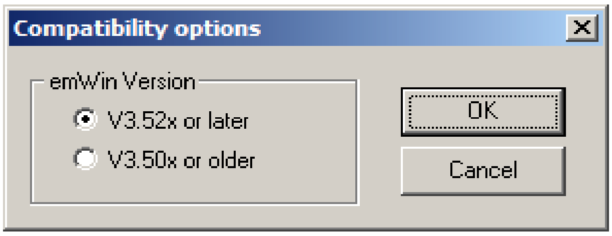
If enabled (standard), all characters are shown. If disabled, only the rows with at least one enabled character are shown.

 Toggles viewing mode

12.6 Options

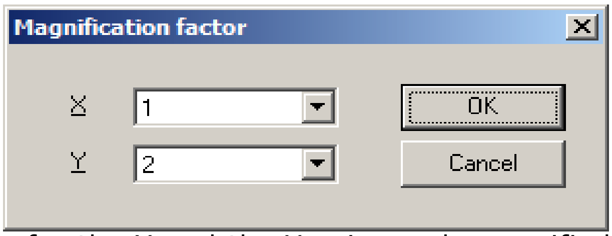
Compatibility options

The Font Converter is able to create font files for all versions of emWin. Because there have been a few small changes of the font format from the emWin version 3.50 to the version 3.52, the C font files for these versions should be slightly different to avoid compiler warnings or compiler errors.
Use the command `Options/Compatibility` to get into the following dialog:

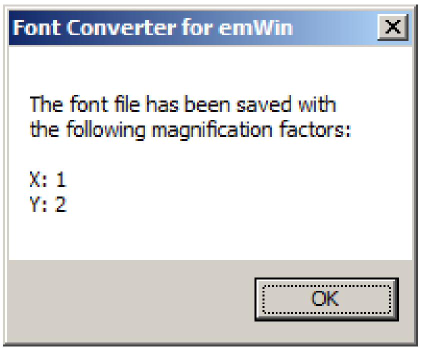


Magnification options

The Font Converter is able to save the font data in a magnified format.
Use the command `Options/Magnification` to get into the following dialog:

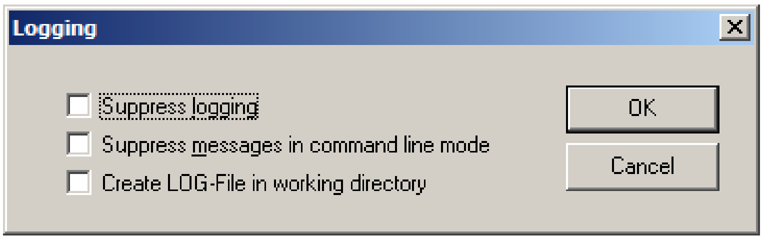


A magnification factor for the X and the Y axis can be specified here. If for example the magnification factor for the Y axis is 2 and the height of the current font data is 18, the font height in the font file will be 36. The magnification in X works similar. After saving the font in a magnified format a short message is shown to inform the user, that the saved font is magnified:



Logging

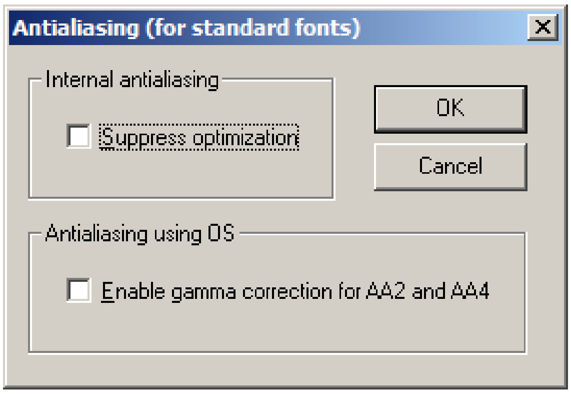
Logging of commands can be enabled or disabled using the command `Options/Logging`:



When logging is enabled the C files contain a history of the commands which has been used to modify the font file.

Antialiasing

When using 'Internal antialiasing' it is recommended to enable `Suppress optimization`. This makes sure, that the horizontal and vertical alignment of the characters fit to each other:



The option `Enable gamma correction for AA2 and AA4` should be disabled. When the option is enabled the antialiased pixels of the characters will appear a little more darker.

12.7 Saving the font

The Font Converter can create C font files or system independent font data files. Details about the SIF format can be found under “System Independent Font (SIF) format” on page 210.

12.7.1 Creating a C file

When you are ready to generate a C file, simply select `File/Save As` from the Font Converter menu, specify a destination and name for the file, choose the C file format and click `Save`. A C file will automatically be created.
The default setting for the filename is built by the name of the source font and the current height in pixels. For example, if the name of the source font is "Example" and the pixel height is 10, the default filename would be Example10.c. If you keep this default name when generating a C file, the resulting name of the font will be GUI_FontExample10.c.
Examples of C files generated from fonts can be found in the sub chapter “Font Examples” on page 267.

12.7.2 Creating a System Independent Font (SIF)

When you are ready to generate the file, simply select `File/Save As` from the Font Converter menu, specify a destination and name for the file, choose the System independent font format and click `Save`. A system independent font file will automatically be created.
This file does not contain C structures which can be compiled with emWin but binary font data, which can be used as described in “System Independent Font (SIF) format” on page 210.

12.7.3 Creating an External Binary Font (XBF)

When you are ready to generate the file, simply select `File/Save As` from the Font Converter menu, specify a destination and name for the file, choose the External binary font format and click `Save`. An external binary font file will automatically be created.
This file does not contain C structures which can be compiled with emWin but binary font data, which can be used as described in “External Bitmap Font (XBF) format” on page 211.

12.8 Loading and modifying a C font file

The Font Converter is able to open existing font files and to modify their font data. The tool can only open C font files generated by the Font Converter. If the C font files have been modified manually, it can not be guaranteed, that they can be opened by the Font Converter.

Step 1: Starting the application

The Font Converter is opened and automatically displays the Font generation options dialog box. In order to load an existing C font file, the "Font generation options" dialog should be closed.

Step 2: Loading an existing C file.

The desired C font file can be opened by selecting `File/Load ’C’ file...` entry in the menu bar. The "Select ’C’ file" dialog can be used to navigate through the file system in order to chose a ’C’ font file which was previously created using the Font Converter. Not all of the font files which are shipped with emWin were created using the Font Converter. According to that not all of those can be reused.

12.9 Loading Glyph (B)itmap (D)istribution (F)ormat

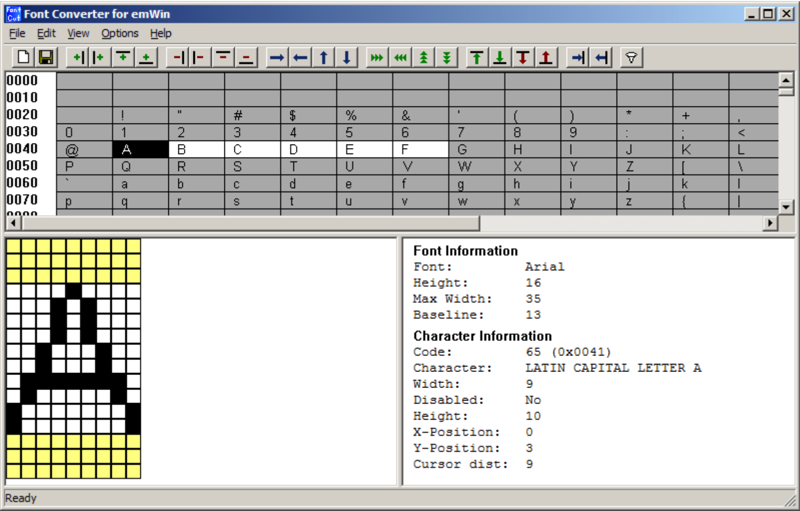
That standard has been defined by Adobe 1993. Because there exist a large number of fonts that format, mainly in the Unix-world, the FontConverter has an option for loading BDF font files. That can be done by selecting `File/Load 'BDF' file...`

12.10 Merging fonts with existing C font files

The Font Converter is able to add the content of an existing C font file to the current font data. Once a font is loaded via `File/Load 'C' file...` or created by `File/New` a C font file can be merged to it using `File/Merge 'C' file...`. The Font Converter requires the fonts to be of the same size, so the merging can be processed properly.

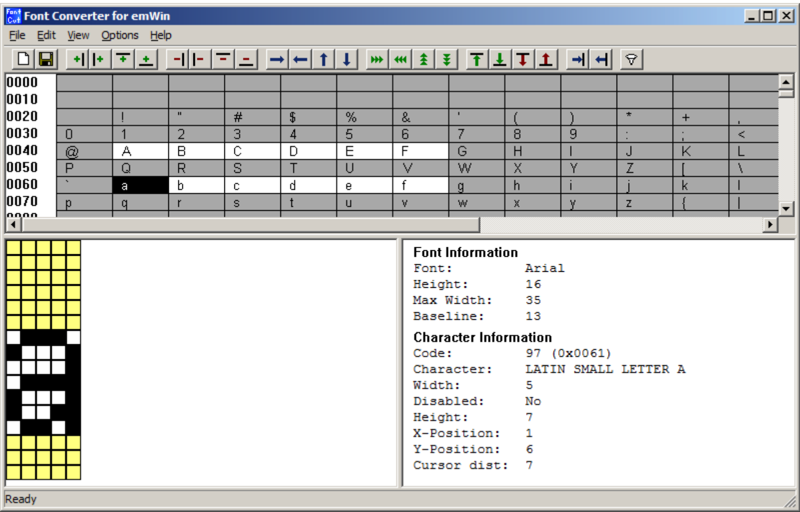
Step 1: Loading an existing font or creating a new one

This step was already described in previous sections. In this example the existing font contains the characters A-F (0x41 - 0x46).



Step 2: Merging a C file

The desired C font file can be merged by selecting `File/Merge 'C' file...` entry in the menu bar. The "Select 'C' file" dialog can be used to navigate through the file system in order to chose a 'C' font file to merge which was previously created using the Font Converter. Not all of the font files which are shipped with emWin were created using the Font Converter. According to that not all of those can be reused.
The characters included in the merged font will be shown and enabled. In this example the characters a-f (0x61 - 0x66) are merged to the existing font. The resulting font can be edited and saved as a new font file.



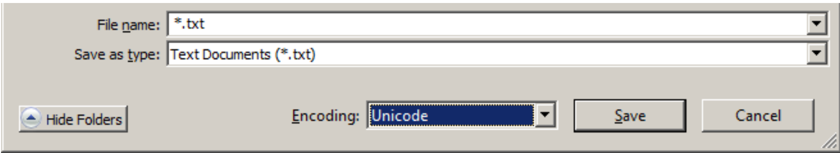
12.11 Pattern files

In order not having to enable every single required character manually, pattern files can be used. Pattern files are simple text files, including the text which has to be displayed in the application.

12.11.1 Creating pattern files using Notepad

One option for creating a pattern file is to use `Notepad`, which is part of the `Windows accessories`:

- Copy the text you want to display into the clipboard.
- Open `Notepad.exe`.
- Insert the contents of the clipboard into the Notepad document.
- Use Format/Font to choose a font which contains all characters of the text. You can skip this step if you do not want to see the characters.
- Use File/Save As to save the pattern file. It is very important that you save the file in text format:



12.11.2 Creating pattern files using the Font Converter

A pattern file may also be created directly in the Font Converter. Select `Edit/Save pattern file` from the menu to create a text file which includes all currently enabled characters.

12.11.3 Enabling characters using a pattern file

It is usually helpful to begin by disabling all characters. Select `Edit/Disable all characters` from the menu if you need to do so.
Now choose `Edit/Read pattern file`. After opening the appropriate pattern file, all characters included in the file are enabled. If the pattern file contains characters which are not included in the currently loaded font, a message box will appear.

12.12 Command line options

12.12.1 Table of commands

The following table shows the available command line options:

Command	Description
create<FONTNAME>,<STYLE>, <HEIGHT>,<TYPE>, <ENCODING>[,<METHOD>]	Create font: <FONTNAME> Name of the font to be used <STYLE> REGULAR - Creates a normal font BOLD - Creates a bold font REGULAR_ITALIC - Creates an italic font BOLD_ITALIC - Creates an italic bold font <HEIGHT> Height in pixels of the font to be created <TYPE> STD - Standard 1 bpp font AA2 - Antialiased font (2bpp) AA4 - Antialiased font (4bpp) EXT - Extended font EXT_FRM - Extended framed font EXT_AA2 - Extended font using 2bpp antialiasing EXT_AA4 - Extended font using 4bpp antialiasing <ENCODING> UC16 - 16 bit Unicode encoding ISO8859 - 8 bit ASCII + ISO8859 JIS - Shift JIS <METHOD> OS - Antialiasing of operating system (default) INTERNAL - Internal antialiasing method
edit<ACTION>, <DETAIL>[,<CNT>]	Equivalent to the 'Edit' menu: <ACTION> DEL - Deletes pixels INS - Inserts pixels <DETAIL> TOP - Delete/insert from top BOTTOM - Delete/insert from bottom <CNT> - Number of operations, default is 1
enable[FIRST-LAST>, <STATE>	Enables or disables the given range of characters: <FIRST-LAST> Hexadecimal values separated by a '-' defining the range of characters <STATE> 1 - Enables the given range 0 - Disables the given range
exit	Exits the application after the job was done.
merge<FILENAME>	Merges the given 'C' file to the current content.
readpattern<FILENAME>	Reads a pattern file: <FILENAME> Name of the pattern file to be read
saveas<FILENAME>,<TYPE>	Saves the font data in a specific format: <FILENAME> File name including extension <TYPE> C - Saves as 'C' file SIF - Saves as System independent font file XBF - Saves as external binary font file
?	Shows all available commands.

Table 12.4: Command line options table

- All commands are processed from left to right.
- If using -exit Font Converter will stop execution if an error occurs. The return code in this case is != 0.

12.12.2 Execution examples

```
FontCvt -create"Cordia New",BOLD,32,EXT,UC16
```

Creates an extended bold font of 32 pixels height with Unicode encoding using the font "Cordia New".

```
FontCvt FontFile.c -enable0-ffff,0 –readpattern"data.txt"
```

Reads the C font file "FontFile.c", disables all characters and reads a pattern file.

12.13 Font Examples

This section provide examples of C files generated by the Font Converter in standard, 2bpp antialiased and 4bpp antialiased mode, respectively.

12.13.1 Resulting C code, standard mode

The following is an example of a C file in standard mode:

```
/*
  C-file generated by Font Converter for emWin version 3.04
  Compiled:       Dec 13 2005 at 12:51:50
  C-file created: Dec 21 2005 at 12:42:57
  Copyright (C) 1998-2005
  Segger Microcontroller Systeme GmbH
  www.segger.com
  Solutions for real time microcontroller applications
  Source file: Sample10.c
  Font:        Arial
  Height:      10
*/

#include "GUI.H"
#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
   font. A good place would be GUIConf.H
*/
extern GUI_CONST_STORAGE GUI_FONT GUI_FontSample10;

/* Start of unicode area <Basic Latin> */
GUI_CONST_STORAGE unsigned char acFontSample10_0041[10] = { /* code 0041 */
  ________,
  ___X____,
  __X_X___,
  __X_X___,
  __X_X___,
  _X___X__,
  _XXXXX__,
  X_____X_,
  X_____X_,
  ________};

GUI_CONST_STORAGE unsigned char acFontSample10_0061[10] = { /* code 0061 */
  ________,
  ________,
  ________,
  _XXX____,
  X___X___,
  _XXXX___,
  X___X___,
  X__XX___,
  _XX_X___,
  ________};

GUI_CONST_STORAGE GUI_CHARINFO GUI_FontSample10_CharInfo[2] = {
   {   8,   8,  1, acFontSample10_0041 } /* code 0041 */
  ,{   6,   6,  1, acFontSample10_0061 } /* code 0061 */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop2 = {
   97                          /* first character               */
  ,97                          /* last character                */
  ,&GUI_FontSample10_CharInfo[1] /* address of first character    */
```

```
  ,(GUI_CONST_STORAGE GUI_FONT_PROP*)0  /* pointer to next GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop1 = {
   65                         /* first character               */
  ,65                         /* last character                */
  ,&GUI_FontSample10_CharInfo[0] /* address of first character    */
  ,&GUI_FontSample10_Prop2      /* pointer to next GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT GUI_FontSample10 = {
   GUI_FONTTYPE_PROP /* type of font    */
  ,10                /* height of font  */
  ,10                /* space of font y */
  ,1                /* magnification x */
  ,1                /* magnification y */
  ,&GUI_FontSample10_Prop1
};
```

12.13.2 Resulting C code, 2 bpp antialiased mode

The following is an example of a C file in 2 bpp antialiased mode:

```
/*
  C-file generated by Font Converter for emWin version 3.04
  Compiled:       Dec 13 2005 at 12:51:50
  C-file created: Dec 21 2005 at 12:42:57
  Copyright (C) 1998-2005
  Segger Microcontroller Systeme GmbH
  www.segger.com
  Solutions for real time microcontroller applications
  Source file: Sample10.c
  Font:        Arial
  Height:      14
*/

#include "GUI.H"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
   font. A good place would be GUIConf.H
*/

extern GUI_CONST_STORAGE GUI_FONT GUI_FontSample10;

/* Start of unicode area <Basic Latin> */
GUI_CONST_STORAGE unsigned char acFontSample10_0041[ 28] = { /* code 0041 */
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x0B, 0xC0,
  0x1F, 0xD0,
  0x2E, 0xE0,
  0x3C, 0xF0,
  0x78, 0xB4,
  0xBF, 0xF8,
  0xE0, 0x78,
  0xE0, 0x3C,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00
};

GUI_CONST_STORAGE unsigned char acFontSample10_0061[ 28] = { /* code 0061 */
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00,
  0x6F, 0x40,
  0x93, 0xC0,
  0x2B, 0xC0,
  0xB7, 0xC0,
  0xF7, 0xC0,
  0x7B, 0xC0,
  0x00, 0x00,
  0x00, 0x00,
  0x00, 0x00
```

```
};

GUI_CONST_STORAGE GUI_CHARINFO GUI_FontSample10_CharInfo[2] = {
   {   8,   8,  2, acFontSample10_0041 } /* code 0041 */
  ,{   6,   6,  2, acFontSample10_0061 } /* code 0061 */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop2 = {
   0x0061 /* first character */
  ,0x0061 /* last character  */
  ,&GUI_FontSample10_CharInfo[  1] /* address of first character */
  ,(GUI_CONST_STORAGE GUI_FONT_PROP*)0 /* pointer to next GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop1 = {
   0x0041 /* first character */
  ,0x0041 /* last character  */
  ,&GUI_FontSample10_CharInfo[  0] /* address of first character */
  ,&GUI_FontSample10_Prop2 /* pointer to next GUI_FONT_PROP */
};

GUI_CONST_STORAGE GUI_FONT GUI_FontSample10 = {
   GUI_FONTTYPE_PROP_AA2 /* type of font    */
  ,14 /* height of font  */
  ,14 /* space of font y */
  ,1 /* magnification x */
  ,1 /* magnification y */
  ,&GUI_FontSample10_Prop1
};
```

12.13.3 Resulting C code, 4 bpp antialiased mode

The following is an example of a C file in 4 bpp antialiased mode:

```
/*
  C-file generated by Font Converter for emWin version 3.04
  Compiled:       Dec 13 2005 at 12:51:50
  C-file created: Dec 21 2005 at 12:42:57
  Copyright (C) 1998-2005
  Segger Microcontroller Systeme GmbH
  www.segger.com
  Solutions for real time microcontroller applications
  Source file: Sample10.c
  Font:        Arial
  Height:      10
*/

#include "GUI.H"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
   font. A good place would be GUIConf.H
*/
extern GUI_CONST_STORAGE GUI_FONT GUI_FontSample10;

/* Start of unicode area <Basic Latin> */
GUI_CONST_STORAGE unsigned char acFontSample10_0041[ 40] = { /* code 0041 */
  0x00, 0x00, 0x00, 0x00,
  0x00, 0xCF, 0xF2, 0x00,
  0x03, 0xFF, 0xF6, 0x00,
  0x09, 0xFB, 0xFB, 0x00,
  0x0E, 0xE2, 0xFE, 0x00,
  0x5F, 0x90, 0xCF, 0x40,
  0xBF, 0xFF, 0xFF, 0x90,
  0xFC, 0x00, 0x6F, 0xC0,
  0xF8, 0x00, 0x2F, 0xF2,
  0x00, 0x00, 0x00, 0x00
};

GUI_CONST_STORAGE unsigned char acFontSample10_0061[ 30] = { /* code 0061 */
  0x00, 0x00, 0x00,
  0x00, 0x00, 0x00,
  0x00, 0x00, 0x00,
  0x3D, 0xFE, 0x60,
  0xD3, 0x0F, 0xE0,
  0x29, 0xCF, 0xF0,
  0xDF, 0x4F, 0xF0,
  0xFF, 0x3F, 0xF0,
```



```
    0x6F, 0xAF, 0xF0,
    0x00, 0x00, 0x00
  };

  GUI_CONST_STORAGE GUI_CHARINFO GUI_FontSample10_CharInfo[2] = {
     {   8,   8,  4, acFontSample10_0041 } /* code 0041 */
    ,{   6,   6,  3, acFontSample10_0061 } /* code 0061 */
  };

  GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop2 = {
     0x0061 /* first character */
    ,0x0061 /* last character  */
    ,&GUI_FontSample10_CharInfo[  1] /* address of first character */
    ,(GUI_CONST_STORAGE GUI_FONT_PROP*)0 /* pointer to next GUI_FONT_PROP */
  };

  GUI_CONST_STORAGE GUI_FONT_PROP GUI_FontSample10_Prop1 = {
     0x0041 /* first character */
    ,0x0041 /* last character  */
    ,&GUI_FontSample10_CharInfo[  0] /* address of first character */
    ,&GUI_FontSample10_Prop2 /* pointer to next GUI_FONT_PROP */
  };

  GUI_CONST_STORAGE GUI_FONT GUI_FontSample10 = {
     GUI_FONTTYPE_PROP_AA4 /* type of font    */
    ,10 /* height of font  */
    ,10 /* space of font y */
    ,1 /* magnification x */
    ,1 /* magnification y */
    ,&GUI_FontSample10_Prop1
  };
```

12.13.4 Resulting C code, extended mode

```
/*
  C-file generated by Font Converter for emWin version 3.04
  Compiled:       Dec 13 2005 at 12:51:50
  C-file created: Dec 21 2005 at 12:45:52
  Copyright (C) 1998-2005
  Segger Microcontroller Systeme GmbH
  www.segger.com
  Solutions for real time microcontroller applications
  Source file: Arial16.c
  Font:        Arial
  Height:      16
*/

#include "GUI.H"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

/* The following line needs to be included in any file selecting the
   font. A good place would be GUIConf.H
*/

extern GUI_CONST_STORAGE GUI_FONT GUI_Font16;

/* Start of unicode area <Basic Latin> */
GUI_CONST_STORAGE unsigned char acGUI_Font16_0041[ 20] = { /* code 0041 */
  ____X___,________,
  ___X_X__,________,
  ___X_X__,________,
  ___X_X__,________,
  __X___X_,________,
  __X___X_,________,
  _XXXXXXX,________,
  _X_____X,________,
  X_______,X_______,
  X_______,X_______};

GUI_CONST_STORAGE unsigned char acGUI_Font16_0061[  7] = { /* code 0061 */
  _XXX____,
  X___X___,
  ____X___,
  _XXXX___,
  X___X___,
  X__XX___,
  _XX_X___};
```

```
GUI_CONST_STORAGE GUI_CHARINFO_EXT GUI_Font16_CharInfo[2] = {
   {   9,  10,   0,   3,   9, acGUI_Font16_0041 } /* code 0041 */
  ,{   5,   7,   1,   6,   7, acGUI_Font16_0061 } /* code 0061 */
};

GUI_CONST_STORAGE GUI_FONT_PROP_EXT GUI_Font16_Prop2 = {
   0x0061 /* first character */
  ,0x0061 /* last character  */
  ,&GUI_Font16_CharInfo[  1] /* address of first character */
  ,(GUI_CONST_STORAGE GUI_FONT_PROP_EXT *)0
};

GUI_CONST_STORAGE GUI_FONT_PROP_EXT GUI_Font16_Prop1 = {
   0x0041 /* first character */
  ,0x0041 /* last character  */
  ,&GUI_Font16_CharInfo[  0] /* address of first character */
  ,&GUI_Font16_Prop2 /* pointer to next GUI_FONT_PROP_EXT */
};

GUI_CONST_STORAGE GUI_FONT GUI_Font16 = {
   GUI_FONTTYPE_PROP_EXT /* type of font    */
  ,16 /* height of font  */
  ,16 /* space of font y */
  ,1 /* magnification x */
  ,1 /* magnification y */
  ,{&GUI_Font16_Prop1}
  ,13 /* Baseline */
  ,7 /* Height of lowercase characters */
  ,10 /* Height of capital characters */
};
```

12.14 Troubleshooting

MS Windows 7 by default shows only fonts which match the computers language settings. If it is required to create a font for other languages, the required Windows fonts might not be shown in the ’Font’ dialog. This is caused by the MS Windows font settings. In order to change those settings, the following ’Control Panel’ path should be opened:

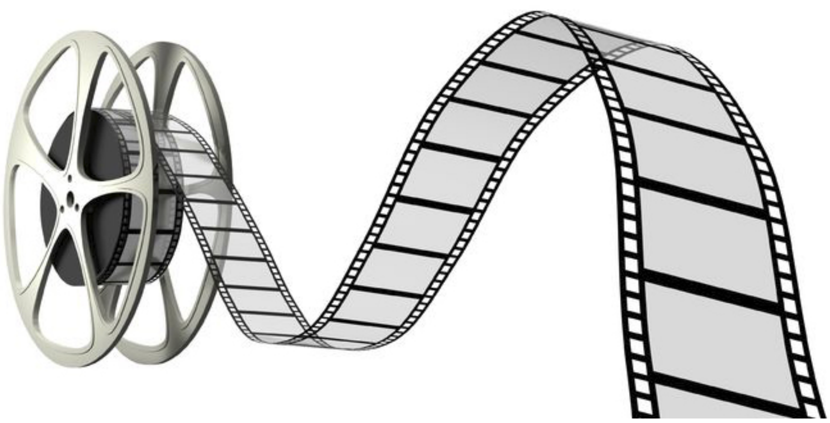
```
Control Panel\All Control Panel Items\Fonts\Font settings
```

Once the checkbox "`Hide fonts based on language settings`" is checked, all Windows fonts installed on the computer should be shown in the ’Font’ dialog.

Chapter 13

Movies

With the new movie file support of emWin images now learned to move. Instead of supporting standard movie file formats, we decided to define our own simple movie file format which is based on single JPEG files. One advantage of this format is that no new decompression algorithms are required. The one and only requirement is the already available JPEG file support. Further only one frame needs to be available in RAM which makes it easy to walk through a movie.



13.1 Introduction

To be able to play movies with the emWin API functions it is required to create files of the emWin specific (**E**)mWin (**M**)ovie (**F**)ile format. These EMF files are containers for single JPEG files. To be able to create such movie files each emWin-shipment contains the tool JPEG2Movie in the \Tools-folder. This converter requires a folder containing JPEG images for each frame to be used.
Usually there are already existing movie files which should be shown with emWin. But the format of these files do not match the EMF file format. Because of that at first a tool is required which is able to create a folder with single JPEG files for each frame of the movie.
All that sounds quite complicated but can be done by a single drag-and-drop operation to one of our helper files. The following will explain in detail what needs to be done to be able to do that.

13.2 Requirements

In opposite to movie file rendering using different frame methods the EMF file format contains complete JPEG files for each frame. The advantage of this format is that not more than one frame is required in memory.

RAM requirement

For the rendering process of an EMF file it is required to have enough dynamic RAM as required for rendering a single JPEG file plus the file size of a single JPEG file. The RAM requirement for rendering a JPEG file can be found in the chapter “JPEG file support” on page 168.

```
Requirement = JPEG requirement + File size of a single JPEG file
```

Please note that ’File size’ does not mean the whole movie file. It means the size of the biggest JPEG file of the movie only.

ROM requirement

Apart from the ROM requirement of the movie file itself app. 22 KByte of additional ROM for the binary code for rendering JPEG based movie files are required.

Performance

To achieve a fluently rendering of the movie a frame rate of 25 frames/seconds is recommended.

13.3 Creating JPEG files with FFmpeg.exe

As mentioned above at first a tool is required which is able to convert files of any movie file format into a folder of single JPEG files for each frame. Currently a plenty of movie file formats exist. emWin does not support all these movie file formats directly.
We recommend the open source tool FFmpeg which is available under www.ffmpeg.org. It is free software licensed under the LGPL or GPL. It is able to convert files of nearly any movie file source format into any desired destination format, also into single JPEG files.
Because the tool is licensed under the LGPL we do not ship this tool directly. It can be loaded from www.ffmpeg.org. Version N-49757-g969039e or newer should be used.

13.4 Creating an EMF

To make the conversion process as easy as possible there are batch files available in the folder `Sample\JPEG2Movie`. These files are:

File	Description
`Prep.bat`	Sets some defaults to be used. Needs to be adapted as explained in the following.
`MakeMovie.bat`	Main conversion file. Does not to be adapted normally.
`<X_SIZE>x<Y_SIZE>.bat`	Some helper files for different resolutions. Detailed description follows.

Table 13.1: Creating an EMF

Please note that all these files need to be located in the same folder. Otherwise they won't work right.

Prep.bat

The `Prep.bat` is required to prepare the environment for the actual process. Calling it directly will not have any effect. It is called by the MakeMovie.bat. To be able to use the batch files it is required to adapt this file at first. This file sets variables used by the file `MakeMovie.bat`. The following table shows these variables:

Variable	Description
`%OUTPUT%`	Destination folder for the JPEG files. Will be cleared automatically when starting the conversion with `MakeMovie.bat`.
`%FFMPEG%`	Access variable for the FFmpeg tool. Should contain the complete path required to call `FFmpeg.exe`.
`%JPEG2MOVIE%`	Access variable for the JPEG2MOVIE tool. Should contain the complete path required to call `JPEG2Movie.exe`.
`%DEFAULT_SIZE%`	Default movie resolution to be used. Can be ignored if one of the `<X-SIZE>x<Y-SIZE>.bat` files are used.
`%DEFAULT_QUALITY%`	Default quality to be used by `FFmpeg.exe` for creating the JPEG files. The less the number the better the quality. The value 1 indicates that a very good quality should be achieved. The value 31 indicates the worst quality. Details can be found in the FFmpeg documentation.
`%DEFAULT_FRAMERATE%`	Frame rate in frames/second to be used by FFmpeg. It defines the number of JPEG files to be generated by `FFmpeg.exe` for each second of the movie. Details can be found in the FFmpeg documentation.

Table 13.2: Prep.bat variables

MakeMovie.bat

This is the main batch file used for the conversion process. Normally it is not required to be change this file, but it is required to adapt Prep.bat first. It could be called with the following parameters:

Parameter	Description
`%1`	Movie file to be converted.
`%2 (optional)`	Size to be used. If not given %DEFAULT_SIZE% of Prep.bat is used.
`%3 (optional)`	Quality to be used. If not given %DEFAULT_QUALITY% of Prep.bat is used.
`%4 (optional)`	Frame rate to be used. If not given %DEFAULT_FRAMERATE% of Prep.bat is used.

Table 13.3: MakeMovie.bat parameters

Since the `FFmpeg` output can differ strongly from the output of previous actions, the `MakeMovie.bat` deletes all output files in the first place. The output folder is defined by in the environmental variable `%OUTPUT%` in `Prep.bat`. After that it uses `FFmpeg.exe` to create the required JPEG files for each frame. Afterwards it calls JPEG2Movie to create a single EMF file which can be used by emWin directly. After

the conversion operation the result can be found in the conversion folder under `FFmpeg.emf`. It also creates a copy of that file into the source file folder. It will have the same name as the source file with a size-postfix and `.emf` extension.
If for example the source file is: `C:\Temp\Movie.mp4` and the size to be used is 480x272 the folder `C:\Temp\` will contain the file `Movie_480x272.emf` after the conversion.

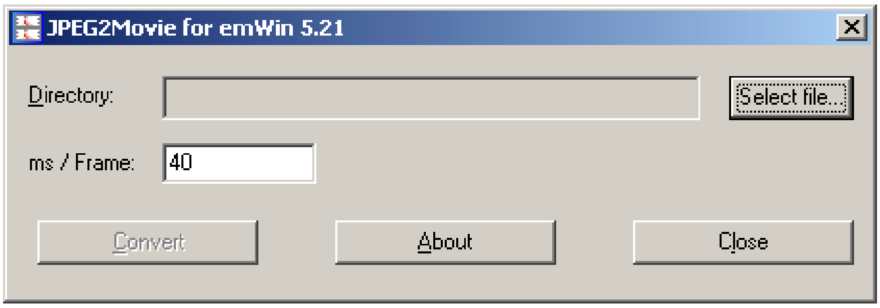
<X_SIZE>x<Y_SIZE.bat

These files are small but useful helpers if several movie resolutions are required. The filenames of the batch files itself are used as parameter '-s' for `FFmpeg.exe`. You can simply drag-and-drop the file to be converted to one of these helper files. After that an .emf file with the corresponding size-postfix can be found in the source file folder.

13.5 Modifying the conversion result

The process of conversion explained above describes how to convert a given movie automatically. But sometimes it could be required to remove or edit JPEGs after generating the images by FFmpeg and before creating the EMF file with JPEG2Movie.
The most simple method for doing that is first creating a complete movie automatically as described above. After that the conversion folder defined by the `%FOLDER%` variable in `Prep.bat` contains all images. Now please feel free to remove, change or add images to the folder. After that JPEG2Movie can be used to convert the new compilation of files to an EMF file.

13.6 Using JPEG2Movie



If there is an already existing compilation of JPEG files to be used the tool JPEG2Movie can be used directly. It is available in the `\Tool` folder of each shipment:

- Start `JPEG2Movie.exe`.
- Select one of the existing JPEG files from the source folder with 'Select file'.
- Define the frame duration to be used (default is 40ms).
- Click the 'Convert' button for creating the EMF file.

After that the folder of the selected file should contain an EMF file. Please note that all JPEGs should have exactly the same resolution.

13.7 Movies API

The table below lists the available movie-related routines in alphabetical order. Detailed descriptions follow:

Routine	Description
GUI_MOVIE_Create()	Creates a movie handle of a file which is completely available in RAM or ROM.
GUI_MOVIE_CreateEx()	Creates a movie handle of a file which is not available in RAM or ROM and needs to be loaded at runtime.
GUI_MOVIE_Delete()	Deletes a movie handle.
GUI_MOVIE_GetFrameIndex()	Returns the current frame number of the given movie.
GUI_MOVIE_GetInfo()	Fills a GUI_MOVIE_INFO structure from a file completely available in RAM or ROM.
GUI_MOVIE_GetInfoEx()	Fills a GUI_MOVIE_INFO structure from a file not available in RAM or ROM which needs to be loaded at runtime.
GUI_MOVIE_GetPos()	Returns the drawing position of the given movie.
GUI_MOVIE_GotoFrame()	Specifies the frame to be drawn at next.
GUI_MOVIE_Pause()	Stops the given movie.
GUI_MOVIE_Play()	Starts playing the given movie.
GUI_MOVIE_SetPeriod()	Sets the period to be used for each frame.
GUI_MOVIE_SetPos()	Sets the drawing position of the given movie.
GUI_MOVIE_Show()	Shows the given movie at the given position.

Table 13.4: Movies API list

GUI_MOVIE_Create()

Description

Creates a movie using a file which is completely available in addressable RAM or ROM.

Prototype

```
GUI_MOVIE_HANDLE GUI_MOVIE_Create(const void      * pFileData, U32 FileSize,
                                  GUI_MOVIE_FUNC * pfNotify);
```

Parameter	Description
pFileData	Pointer to the memory location of the file.
FileSize	Size of file in bytes.
pfNotify	Optional pointer to a notification function of type GUI_MOVIE_FUNC. If set this function would be called for each frame.

Table 13.5: GUI_MOVIE_Create() parameter list

Prototype of GUI_MOVIE_FUNC

```
void GUI_MOVIE_FUNC(GUI_MOVIE_HANDLE hMovie,        int Notification,
                    U32              CurrentFrame);
```

Permitted values for element Notification	
GUI_MOVIE_NOTIFICATION_PREDRAW	This notification is send immediately before a single frame is drawn.
GUI_MOVIE_NOTIFICATION_POSTDRAW	This notification is send immediately after a single frame is drawn.
GUI_MOVIE_NOTIFICATION_START	This notification is send immediately after rendering of a movie file has been started.
GUI_MOVIE_NOTIFICATION_STOP	This notification is send immediately after stopping a movie file.
GUI_MOVIE_NOTIFICATION_DELETE	This notification is send immediately after a movie file is deleted.

Return value

Movie handle on success, 0 on error.

Additional information

The callback function can be used to achieve overlays for specific frames or for using multiple buffers for the drawing process. The sample folder contains the sample `BASIC_ShowMovies.c` which shows how to use that feature in detail.

GUI_MOVIE_CreateEx()

Description

Creates a movie using a file which is not available in addressable RAM or ROM. A user defined `GetData()` function is used to fetch the file data.

Prototype

```
GUI_MOVIE_HANDLE GUI_MOVIE_CreateEx(GUI_GET_DATA_FUNC * pfGetData,
                                    void              * pParam,
                                    GUI_MOVIE_FUNC    * pfNotify);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. Details about the GetData() function can be found in the section "Getting data with the ...Ex() functions" on page 187.
pParam	Void pointer passed to the function pointed by pfGetData.
pfNotify	Optional pointer to a notification function as described under GUI_MOVIE_Create().

Table 13.6: GUI_MOVIE_CreateEx() parameter list

Return value

Movie handle on success, 0 on error.

Additional information

When playing a movie not from an addressable memory location, the movie function of emWin reads the file frame by frame. That means that only one file access is required for each frame. But that also means that enough RAM needs to be available for buffering a complete JPEG file.
For more information please also refer to `GUI_MOVIE_Create()`.

GUI_MOVIE_Delete()

Description

Deletes the given movie from memory.

Prototype

```
int GUI_MOVIE_Delete(GUI_MOVIE_HANDLE hMovie);
```

Parameter	Description
hMovie	Handle to the movie to be deleted.

Table 13.7: GUI_MOVIE_Delete() parameter list

Return value

0 on success, 1 on error.

Additional information

If the movie is currently playing, the function stops it. It is not required to call `GUI_MOVIE_Stop()` explicitly.

GUI_MOVIE_GetFrameIndex()

Description

If the movie is already playing the function returns the index of the next frame to be shown. If the movie is currently stopped/paused, it returns the frame index of the last shown image.

Prototype

```
U32 GUI_MOVIE_GetFrameIndex(GUI_MOVIE_HANDLE hMovie);
```

Parameter	Description
hMovie	Handle of the movie.

Table 13.8: GUI_MOVIE_GetFrameIndex() parameter list

Return value

Frame index as described above.

GUI_MOVIE_GetInfo()

Description

Fills a `GUI_MOVIE_INFO` structure with information about the given movie. The movie needs to be available in an addressable memory location.

Prototype

```
int GUI_MOVIE_GetInfo(const void * pFileData,
                      U32 FileSize, GUI_MOVIE_INFO * pInfo);
```

Parameter	Description
pFileData	Pointer to the memory location of the file.
FileSize	Size of file in bytes.
pInfo	Pointer to a structure of type GUI_MOVIE_INFO to be filled by the function.

Table 13.9: GUI_MOVIE_GetInfo() parameter list

Elements of structure GUI_MOVIE_INFO

Data type	Element	Description
int	xSize	Horizontal resolution of the movie in pixels.
int	ySize	Vertical resolution of the movie in pixels.
int	msPerFrame	Period of one frame in ms.
U32	NumFrames	Number of frames of the movie file.

Table 13.10: GUI_MOVIE_INFO element list

Return value

0 on success, 1 on error.

GUI_MOVIE_GetInfoEx()

Description

Fills a `GUI_MOVIE_INFO` structure with information about the given movie. The movie does not need to be available in an addressable memory location.

Prototype

```
int GUI_MOVIE_GetInfoEx(GUI_GET_DATA_FUNC * pfGetData,
```

```
void * pParam, GUI_MOVIE_INFO * pInfo);
```

Parameter	Description
pfGetData	Pointer to a function which is called for getting data. Details about the GetData() function can be found in the section "Getting data with the ...Ex() functions" on page 187.
pParam	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a structure of type GUI_MOVIE_INFO to be filled by the function.

Table 13.11: GUI_MOVIE_GetInfoEx() parameter list

Return value

0 on success, 1 on error.

GUI_MOVIE_GetPos()

Description

Returns the drawing position and resolution of the given movie.

Prototype

```
int GUI_MOVIE_GetPos(GUI_MOVIE_HANDLE hMovie, int * pxPos, int * pyPos,
                     int * pxSize, int * pySize);
```

Parameter	Description
hMovie	Handle of the movie.
pxPos	Pointer to an integer to be filled with the drawing position in x. Could be NULL.
pyPos	Pointer to an integer to be filled with the drawing position in y. Could be NULL.
pxSize	Pointer to an integer to be filled with the horizontal resolution in x. Could be NULL.
pySize	Pointer to an integer to be filled with the horizontal resolution in y. Could be NULL.

Table 13.12: GUI_MOVIE_GetPos() parameter list

Return value

0 on success, 1 on error.

GUI_MOVIE_GotoFrame()

Description

Sets the frame index to be shown at next.

Prototype

```
int GUI_MOVIE_GotoFrame(GUI_MOVIE_HANDLE hMovie, U32 Frame);
```

Parameter	Description
hMovie	Handle of the movie.
Frame	Number of the desired frame.

Table 13.13: GUI_MOVIE_GotoFrame() parameter list

Return value

0 on success, 1 on error.

Additional information

If the given frame index is not in the range of the given file, the function stops the movie and returns with an error.

GUI_MOVIE_Pause()

Description

Stops playing the given movie immediately. Can be continued later with `GUI_MOVIE_Play()`.

Prototype

```
int GUI_MOVIE_Pause(GUI_MOVIE_HANDLE hMovie);
```

Parameter	Description
hMovie	Handle of the movie to be stopped.

Table 13.14: GUI_MOVIE_Pause() parameter list

Return value

0 on success, 1 on error.

GUI_MOVIE_Play()

Description

Continues playing of the given movie.

Prototype

```
int GUI_MOVIE_Play(GUI_MOVIE_HANDLE hMovie);
```

Parameter	Description
hMovie	Handle of the movie to be stopped.

Table 13.15: GUI_MOVIE_Play() parameter list

Return value

0 on success, 1 on error.

GUI_MOVIE_SetPeriod()

Description

Sets the period to be used for one single frame in ms.

Prototype

```
int GUI_MOVIE_SetPeriod(GUI_MOVIE_HANDLE hMovie, unsigned Period);
```

Parameter	Description
hMovie	Handle of the movie to be stopped.
Period	Period to be used in ms.

Table 13.16: GUI_MOVIE_SetPeriod() parameter list

Return value

0 on success, 1 on error.

Additional information

This function can be used to vary the speed of a movie. If the period is too short to be achieved by the hardware emWin skips the next image(s).

GUI_MOVIE_SetPos()

Description

Sets the drawing position to be used for the given movie.

Prototype

```
int GUI_MOVIE_SetPos(GUI_MOVIE_HANDLE hMovie, int xPos, int yPos);
```

Parameter	Description
hMovie	Handle of the movie to be stopped.
xPos	X-position is screen coordinates to be used.
yPos	Y-position is screen coordinates to be used.

Table 13.17: GUI_MOVIE_SetPos() parameter list

Return value

0 on success, 1 on error.

Additional information

It is not required that the given position makes the movie completely visible. It can be partly or completely outside of the visible screen.

GUI_MOVIE_Show()

Description

Starts playing the given movie at the given position.

Prototype

```
int GUI_MOVIE_Show(GUI_MOVIE_HANDLE hMovie, int xPos, int yPos, int DoLoop);
```

Parameter	Description
hMovie	Handle of the movie to be stopped.
xPos	X-position is screen coordinates to be used.
yPos	Y-position is screen coordinates to be used.
DoLoop	1 if the movie should be shown in an endless loop, 0 if not.

Table 13.18: GUI_MOVIE_Show() parameter list

Return value

0 on success, 1 on error.

Additional information

If the given movie is already playing the function returns an error. But the movie remains playing.

Chapter 14

Animations

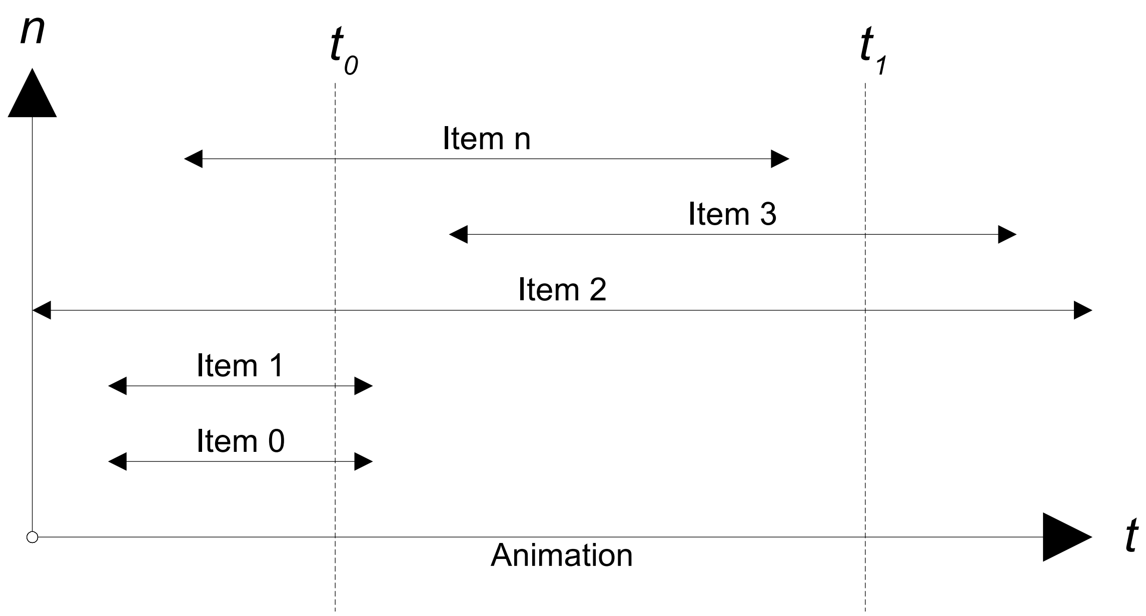
Animations can be achieved by playing GIF animations, movies or showing animated graphic objects. Whereas GIF animations and movies can be used only for playing a fixed sequence of pictures, the animation object of emWin is able to draw runtime generated scenes of one or multiple independent animated objects, which could be used to reflect dynamic application data. The following chapter explains how the animation object could be used to achieve user defined animations.

14.1 Introduction

Bringing up motion into an application could be done by several ways. If a fixed sequence of pictures does not work and one or more objects should change their properties like position, color, shape, size or whatever within a given period of time, emWin offers an animation object for exactly that purpose. It is able to animate multiple animation items during the timeline of an animation. Animation item means a custom defined routine which receives a position value from the animation object. That value is calculated in dependence of the current point in time. The calculation could be done by predefined or custom defined methods.
Each item has its own start and end time which determine if it is called or not during the execution.

14.2 Creating an animation object

Creating an animation object is done by passing the duration and the minimum time per 'slice' to the creation routine. Optionally a custom defined void pointer and a pointer to a 'slice callback' routine could be passed. The term 'slice' will be explained later. After the handle of an animation object is available one or more animation items could be added to the animation.



The diagram shows an animation with several items within the animation period where each item has its own start and end time. emWin does not limit the number of items. Start time and end time of each item have to be within the timeline of the animation.

Example:

```
//
// Creating an animation of 4 seconds and a min time/slice of 50ms
//
hAnim = GUI_ANIM_Create(4000, 50, NULL, NULL);
```

14.3 Adding items to an animation

An animation item is a routine which receives information from the animation object during the period of execution. When adding an item its start time, end time, 'method of position calculation' and animation routine to be called should be passed. Further an optional void pointer could be used which is passed to the animation routine during the animation is executed.

Example:

```
static void _AnimDrawSomething(GUI_ANIM_INFO * pInfo, void * pVoid) {
  ...
}

void Application(void) {
  ...
  GUI_ANIM_AddItem(hAnim,  500, 2500, ANIM_LINEAR, pVoid, _AnimDrawSomething);
 ...
 }
```

14.4 Position calculation

Each item has its own start and end time within the timeline of the animation. If the current point of time is within the period of an item, the animation object calculates a position value. The following table shows the available methods for calculating the position value:

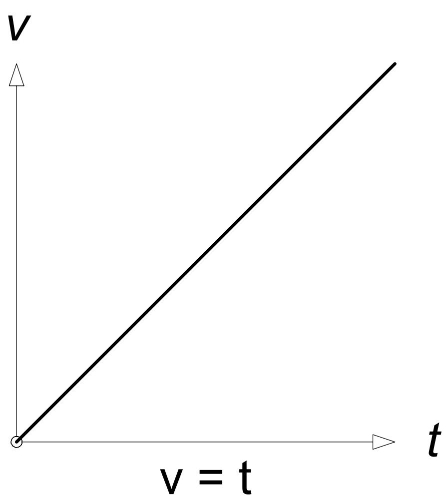
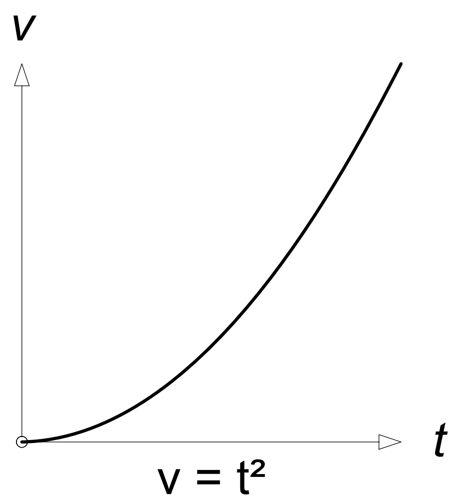
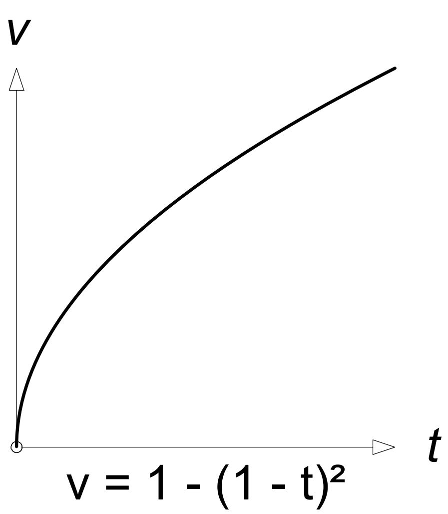
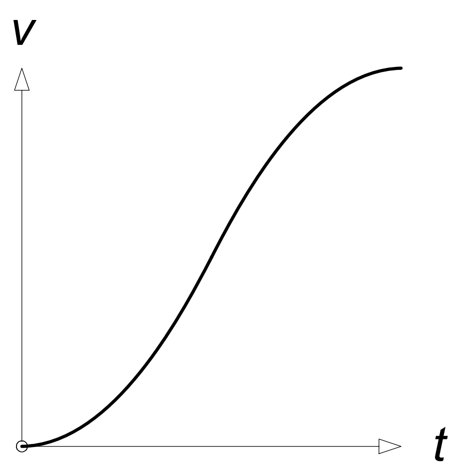
ANIM_LINEAR	ANIM_ACCEL
	
ANIM_DECEL	**ANIM_ACCELDECEL**
	

Table 14.1: Available methods of position calculation

The value passed to the application is between 0 and GUI_ANIM_RANGE. If for example ANIM_ACCEL is used and the animation item is currently exactly in the middle the value passed to the application is 0.5 * 0.5 * GUI_ANIM_RANGE.

GUI_ANIM_RANGE

That value is defined by emWin as follows:

Type	Macro	Default	Description
V	GUI_ANIM_RANGE	32767	Define the default range of an animation.

Table 14.2: Animation configuration

The value could be changed on demand.

14.4.1 Custom defined position calculation

If the above predefined methods do not meet the requirements a custom defined routine could be used. The below sample shows how that works. The routine is called by the animation object each time a single position value is required. The routine gets start time, end time and current time of the animation item. Its job is to calculate a value in dependence of the given parameters. The predefined methods of emWin generate values between 0 and GUI_ANIM_RANGE. If a custom defined routine is used the range could be different.

Example:

```
I32 _CalcPosition(GUI_TIMER_TIME ts, GUI_TIMER_TIME te, GUI_TIMER_TIME tNow) {
  ...
}

void Application(void) {
  ...
  GUI_ANIM_AddItem(hAnim,  ..., ..., _CalcPosition, ..., ...);
  ...
}
```

14.5 Executing an animation

To keep the animation running the routine GUI_ANIM_Exec() has to be called periodically with the handle of the animation to be executed. Each time it is called it checks which items of the given animation are within the current point of time. That collection of items represents one slice of animation items which are called at the given point of time. The animation diagram at the beginning for example shows the points of time t_0 and t_1. The t_0 slice consists of items 0, 1, 2 and n and the t_1 slice of item 2 and 3. The following example shows how an animation could be executed:

Example:

```
void Application(void) {
  ...
  while (GUI_ANIM_Exec(hAnim) == 0) {
    GUI_X_Delay(5); // Idle time for other tasks
  }
}
```

14.5.1 Using a slice callback function

Before calling the first item of an animation slice and after calling the last item an optional callback function could be called. A slice callback function could be used for example to avoid flickering. Before drawing the first item the application could switch to the back buffer or lock the cache and after drawing the last item the back buffer could be made visible or the cache could be unlocked. To use a slice callback function a pointer to that function should be passed to the creation routine.

Example:

```
static void _cbSliceInfo(int State, void * pVoid) {
  switch (State) {
```



```
    case GUI_ANIM_START:
      GUI_MULTIBUF_Begin();
      break;
    case GUI_ANIM_END:
      GUI_MULTIBUF_End();
      break;
    }
}

void Application(void) {
  ...
  hAnim = GUI_ANIM_Create(..., ..., pVoid, _cbSliceInfo);
}
```

14.5.2 Animation item

An animation item is simply a routine which is executed by the animation object. It is called by passing a pointer to a GUI_ANIM_INFO structure and the application defined void pointer passed to GUI_ANIM_AddItem().

Prototype

```
static void _AnimDrawRect(GUI_ANIM_INFO * pInfo, void * pVoid);
```

Elements of structure GUI_ANIM_INFO

Data type	Element	Description
int	Pos	Position value calculated by the selected position calculation routine.
int	State	(see table below)
GUI_ANIM_HANDLE	hAnim	Handle of the animation object.
GUI_TIMER_TIME	Period	Period of the animation object.

Table 14.3: GUI_ANIM_INFO element list

Permitted values for element State	
GUI_ANIM_START	First execution.
GUI_ANIM_RUNNING	Passed to all items which are not the first and not the last.
GUI_ANIM_END	Last execution.

14.6 Animations API

The following table lists the animation related API functions.

Routine	Description
GUI_ANIM_AddItem()	Adds an animation item to an animation object.
GUI_ANIM_Create()	Creates an animation object.
GUI_ANIM_Delete()	Removes an animation object from memory.
GUI_ANIM_Exec()	Used to keep an animation alive by calling it periodically.
GUI_ANIM_Start()	Sets the current time as starting time of the animation.

Table 14.4: Animation API list

GUI_ANIM_AddItem()

Description

Adds an item to the given animation.

Prototype

```
int GUI_ANIM_AddItem(GUI_ANIM_HANDLE hAnim,
                     GUI_TIMER_TIME ts, GUI_TIMER_TIME te,
                     GUI_ANIM_GETPOS_FUNC pfGetPos,
```

```
void * pVoid, GUI_ANIMATION_FUNC * pfAnim);
```

Parameter	Description
hAnim	Handle of animation object to be used.
ts	Relative start time in ms.
te	Relative end time in ms. Needs to be > ts.
pfGetPos	Pointer to a routine for calculating the position value.
pVoid	Optional void pointer passed to the animation function during the execution.
pfAnim	Pointer to the animation function (item) to be executed by the obect.

Table 14.5: GUI_ANIM_AddItem() parameter list

Additional information

Start and end time are relative to the time value set when calling `GUI_ANIM_Start()`.

Return value

0 on success, 1 on error.

GUI_ANIM_Create()

Description

Creates an animation object.

Prototype

```
GUI_ANIM_HANDLE GUI_ANIM_Create(GUI_TIMER_TIME Period,
                                unsigned MinTimePerSlice,
                                void * pVoid,
                                void (* pfSlice)(int State, void * pVoid));
```

Parameter	Description
Period	Duration in ms of the whole animation.
MinTimePerSlice	Minimum time of one animation slice in ms.
pVoid	Optional void pointer passed to the slice callback function.
pfSlice	Optional slice callback function.

Table 14.6: GUI_ANIM_Create() parameter list

Additional information

The value MinTimePerSlice determines the Framerate when executing the animation with `GUI_ANIM_Exec()`. The given time determines the minimum period between the execution of 2 slices of animation items.

Return value

Handle of the animation object or 0 if no memory is available.

GUI_ANIM_Delete()

Description

Deletes an animation object and all of its data.

Prototype

```
void GUI_ANIM_Delete(GUI_ANIM_HANDLE hAnim);
```

Parameter	Description
hAnim	Animation object to be used.

Table 14.7: GUI_ANIM_Delete() parameter list

GUI_ANIM_Exec()

Description

Should be used to keep the given animation alive. The function needs to be called periodically.

Prototype

```
int GUI_ANIM_Exec(GUI_ANIM_HANDLE hAnim);
```

Parameter	Description
hAnim	Animation object to be used.

Table 14.8: GUI_ANIM_Exec() parameter list

Additional information

Within a typical execution loop the application should give other tasks a chance to do something. That could be done by simply calling `GUI_Delay()`.

Return value

0 if the given animation is within its timeline, 1 if the animation period is expired.

Example

```
while (GUI_ANIM_Exec(hAnim) == 0) {
  GUI_X_Delay(5); // Idle time for other tasks
}
```

GUI_ANIM_Start()

Description

Sets the start time of the given animation.

Prototype

```
void GUI_ANIM_Start(GUI_ANIM_HANDLE hAnim);
```

Parameter	Description
hAnim	Animation object to be used.

Table 14.9: GUI_ANIM_Start() parameter list

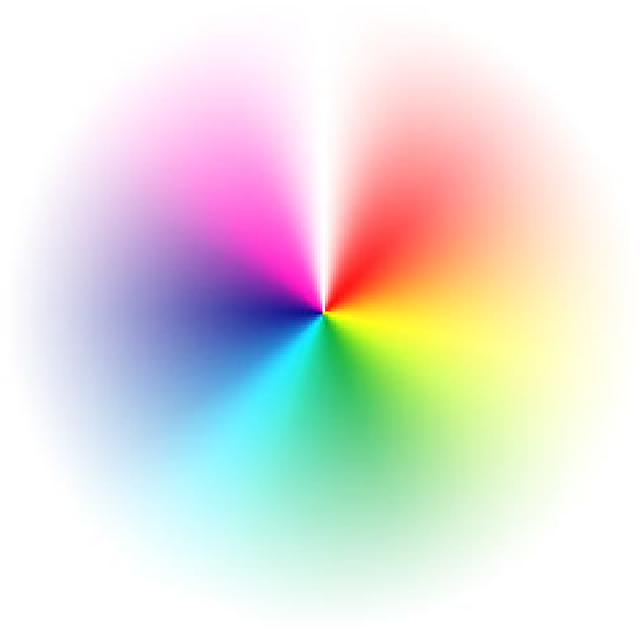
Additional information

The function does nothing but setting the current time as start time. From that moment `GUI_ANIM_Exec()` should return 0 for the given animation period.

Chapter 15

Colors

emWin supports color displays, grayscale (monochrome with different intensities) and black/white displays. An existing emWin application can be used with different kinds of displays. If an existing application should be used with a new display only the display configuration (normally located in LCDConf.c) needs to be changed. To achieve this the application uses 'logical colors'. That logical color format is independent of the color (or better pixel-) format required for the display controller.



15.1 Color management

If an application uses a color for a drawing operation that color normally should be a 'logical color' containing 8 bits for each color channel and 8 bits for the alpha channel. The display controller normally requires a different format, called 'index value' in this document. 'Logical colors' are independent of the used hardware. The format of the 'index values' depend on the requirements of the display controller. Wherever emWin draws anything on the display it converts the 'logical color' used by the application into an 'index value' for the controller. That conversion is done automatically by the color conversion routines configured in the display configuration routine `LCD_X_Config()` which should set up the routines to be used. That could be done separately for each layer.
emWin supports different ways of color conversion:

Fixed palette mode

Using a fixed palette mode is the most recommended way of color conversion. It sets up conversion routines for converting a color into an index value and vice versa. emWin provides a large set of predefined fixed palette modes explained later in this chapter.

Application defined color conversion

If none of the predefined fixed palette modes match the requirements a custom color conversion could be used. That simply means custom defined routines for converting a color into an index value and vice versa. Details explained later in this chapter.

Custom palette mode

If a display controller with a palette based color management is used, either one of the fixed palette modes could be used or a custom palette could be defined.
In case of using a custom palette emWin converts the logical colors by using an optimized version of the "least-square deviation search". It compares the color to display (the logical color) with all the available palette colors and uses the one that the LCD-metric considers closest. Please note that using a custom palette mode could degrade the performance. Details about how to use a custom palette explained later in this chapter.

15.2 Logical colors

A logical color contains 8 bits for each color component and 8 bits for the alpha channel. Since emWin V5.30 emWin supports 2 logical color formats:

Logical color format ABGR (default)

Alpha	Blue	Green	Red
0x00 - opaque 0xFF - transparent			
DB31 - DB24	DB23 - DB16	DB15 - DB08	DB07 - DB00

For a long period of time the above format was the only supported logical color format. That implies that the used logical color format of all applications using emWin written within that period is also the same.

Logical color format ARGB

Alpha	Red	Green	Blue
0x00 - transparent 0xFF - opaque			
DB31 - DB24	DB23 - DB16	DB15 - DB08	DB07 - DB00

Because of more and more hardware platforms using a slightly different pixel format we decided to add the option of using ARGB as logical color format to be able to improve performance significantly under certain circumstances. Please note that the alpha definition of the ARGB format is also different to the ABGR format.

15.3 Switching to ARGB

Using that logical color format could make sense if a display controller is used which supports exactly that color format as index value. In that case the performance could be improved significantly, for example when using an STM32F4 with ChromeART accelerator.

15.3.1 Configuration

When starting with emWin or with a new project only the configuration file needs to be changed and the following line needs to be added to GUIConf.h:

```
#define GUI_USE_ARGB (1)
```

15.3.2 Required changes in existing applications

When switching from ABGR to ARGB or vice versa some things have to be considered.

Colors

Wherever colors are defined as hexadecimal values in the application the values have to be changed or even better a conversion macro has to be used. The following table shows the use of the same color with ARGB, ABGR and conversion macro:

File	Description
ARGB	GUI_SetColor(**0xA02010**);
ABGR	GUI_SetColor(**0xFF1020A0**);
Conversion macro	GUI_SetColor(**GUI_MAKE_COLOR**(0xFF1020A0));

Table 15.1: Creating an EMF

Of course all predefined color values will be changed automatically.

32 bpp Memory devices

emWin contains a couple of functions to be used with 32 bpp memory devices only. Those functions expect a determined memory device format. When working in ABGR mode that format is GUICC_8888. When switching to ARGB that format is GUICC_M8888I.

DIB bitmaps

Palette based bitmaps created by the bitmap converter contains an array of palette colors. Example:

```
static GUI_CONST_STORAGE GUI_COLOR _Colors8x1[] = {
  0x000000, 0xC04040, 0x40C020, 0xC0A000,
  0x4020E0, 0xC040A0, 0x00FFFF, 0xFFFFFF
};
```

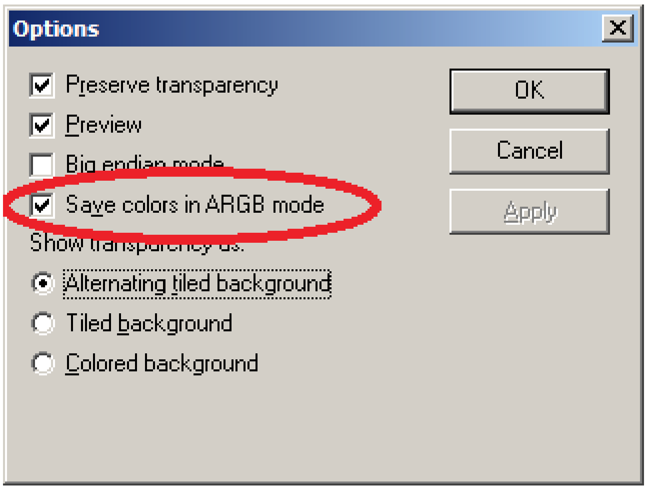

All existing bitmaps need to be changed:

```
static GUI_CONST_STORAGE GUI_COLOR _Colors8x1[] = {
  0xFF000000, 0xFF4040C0, 0xFF20C040, 0xFF00A0C0,
  0xFFE02040, 0xFFA040C0, 0xFFFFFF00, 0xFFFFFFFF
};
```

If an application conains a large number of bitmaps a better way would be to convert the bitmaps again with new settings for the bitmap converter.

15.3.3 Configuring the BitmapConverter

In order to configure the bitmap converter to save colors directly in ARGB instead of ABGR the option available under 'Options\Save colors in ARGB mode' should be activated:



15.4 Predefined colors

In addition to self-defined colors, some standard colors are predefined in emWin, as shown in the following table:

GUI_BLUE		0xFF0000
GUI_GREEN		0x00FF00
GUI_RED		0x0000FF
GUI_CYAN		0xFFFF00
GUI_MAGENTA		0xFF00FF
GUI_YELLOW		0x00FFFF
GUI_LIGHTBLUE		0xFF8080
GUI_LIGHTGREEN		0x80FF80
GUI_LIGHTRED		0x8080FF
GUI_LIGHTCYAN		0xFFFF80
GUI_LIGHTMAGENTA		0xFF80FF
GUI_LIGHTYELLOW		0x80FFFF
GUI_DARKBLUE		0x800000
GUI_DARKGREEN		0x008000
GUI_DARKRED		0x000080
GUI_DARKCYAN		0x808000
GUI_DARKMAGENTA		0x800080
GUI_DARKYELLOW		0x008080
GUI_WHITE		0xFFFFFF
GUI_LIGHTGRAY		0xD3D3D3
GUI_GRAY		0x808080
GUI_DARKGRAY		0x404040
GUI_BLACK		0x000000
GUI_BROWN		0x2A2AA5
GUI_ORANGE		0x00A5FF

Example

```
/* Set background color to magenta */
GUI_SetBkColor(GUI_MAGENTA);
GUI_Clear();
```


15.5 The color bar test routine

The color bar example program is used to show 13 color bars as follows:

Black -> Red, White -> Red, Black -> Green, White -> Green, Black -> Blue, White -> Blue, Black -> White, Black -> Yellow, White -> Yellow, Black -> Cyan, White -> Cyan, Black -> Magenta and White -> Magenta.

This little routine may be used on all displays in any color format. Of course, the results vary depending on the colors that can be displayed; the routine requires a display size of 320*240 in order to show all colors. The routine is used to demonstrate the effect of the different color settings for displays. It may also be used by a test program to verify the functionality of the display, to check available colors and grayscales, as well as to correct color conversion. The screenshots are taken from the windows simulation and will look exactly like the actual output on your display if your settings and hardware are working properly. The routine is available as `COLOR_ShowColorBar.c` in the examples shipped with emWin.

15.6 Fixed palette modes

The following table lists the available fixed palette color modes and the necessary identifiers which need to be used when creating a driver- or a Memory Device. Detailed descriptions follow.

Identifier	No. available colors	Mask
GUICC_1	black and white	0x01 -> 00000001
GUICC_2	4 grayscales	0x03 -> 00000011
GUICC_4	16 grayscales	0x0F -> 00001111
GUICC_5	32 grayscales	0x1F -> 00011111
GUICC_16	16	0x0F -> 00001111
GUICC_1616I	16 + 4 bit alpha blending	0xFF -> 11111111
GUICC_111	8	0x07 -> 00000BGR
GUICC_M111	8	0x07 -> 00000RGB
GUICC_222	64	0x3F -> 00BBGGRR
GUICC_M222	64	0x3F -> 00RRGGBB
GUICC_8	256 grayscales	0xFF -> 11111111
GUICC_233	256	0xFF -> BBGGGRRR
GUICC_M233	256	0xFF -> RRGGGBBB
GUICC_323	256	0xFF -> BBBGGRRR
GUICC_M323	256	0xFF -> RRRGGBBB
GUICC_332	256	0xFF -> BBBGGGRR
GUICC_M332	256	0xFF -> RRRGGGBB
GUICC_444_12	4096	0x0FFF -> 0000BBBBGGGGRRRR
GUICC_M444_12	4096	0x0FFF -> 0000RRRRGGGGBBBB
GUICC_444_12_1	4096	0xFFF0 -> BBBBGGGGRRRR0000
GUICC_444_16	4096	0x7BDE -> 0BBBB0GGGG0RRRR0
GUICC_M444_16	4096	0x7BDE -> 0RRRR0GGGG0BBBB0
GUICC_M4444I	4096 + 4 bit alpha blending	0xFFFF -> AAAARRRRGGGGBBBB
GUICC_555	32768	0x7FFF -> 0BBBBBGGGGGRRRRR
GUICC_M555	32768	0x7FFF -> 0RRRRRGGGGGBBBBB
GUICC_M1555I	32768 + 1 bit transparency	0xFFFF -> TRRRRRGGGGGBBBBB
GUICC_556	65536	0xFFFF -> BBBBBGGGGGRRRRRR
GUICC_M556	65536	0xFFFF -> RRRRRGGGGGBBBBBB
GUICC_565	65536	0xFFFF -> BBBBBGGGGGGRRRRR
GUICC_M565	65536	0xFFFF -> RRRRRGGGGGGBBBBB
GUICC_655	65536	0xFFFF -> BBBBBBGGGGGRRRRR

Table 15.2: Fixed palette mode details

Identifier	No. available colors	Mask
GUICC_M655	65536	0xFFFF -> RRRRRRGGGGGBBBBB
GUICC_666	262144	0x0003FFFF -> BBBBBBGGGGGGRRRRRR
GUICC_M666	262144	0x0003FFFF -> RRRRRRGGGGGGBBBBBB
GUICC_666_9	262144	0x01FF01FF -> 0000000BBBBBBGGG0000000GGGRRRRRR
GUICC_M666_9	262144	0x01FF01FF -> 0000000RRRRRRGGG0000000GGGBBBBBB
GUICC_822216	256	0xFF - Bits are not explicitly assigned to a color.
GUICC_84444	240	0xFF - Bits are not explicitly assigned to a color.
GUICC_8666	232	0xFF - Bits are not explicitly assigned to a color.
GUICC_8666_1	233 (232 + transparency)	0xFF - Bits are not explicitly assigned to a color.
GUICC_88666I	232 + 8 bits alpha blending	0xFFFF -> AAAAAAAACCCCCCCC
GUICC_888	16M	0x00FFFFFF -> BBBBBBBBGGGGGGGGRRRRRRRR
GUICC_M888	16M	0x00FFFFFF -> RRRRRRRRGGGGGGGGBBBBBBBB
GUICC_8888	16M + 8 bit alpha blending	0xFFFFFFFF -> AAAAAAAABBBBBBBBGGGGGGGGRRRRRRRR
GUICC_M8888	16M + 8 bit alpha blending	0xFFFFFFFF -> AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB
GUICC_M8888I	16M + 8 bit alpha blending	0xFFFFFFFF -> AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB
GUICC_0	-	CUSTOM DEFINED FIXED PALETTE MODE
GUICC_1_2 GUICC_1_4 GUICC_1_5 GUICC_1_8 GUICC_1_16 GUICC_1_24	2 (black and white)	0x00000001 0x00000003 0x0000001F 0x000000FF 0x0000FFFF 0x00FFFFFF

Table 15.2: Fixed palette mode details

R - Red
G - Green
B - Blue
C - Color (in case of no explicit bit assignment to colors)
T - Transparency bit
A - Alpha mask

15.7 Detailed fixed palette mode description

The following gives a detailed description of the available colors in each predefined fixed palette mode.

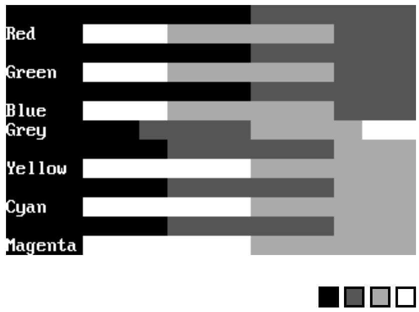
GUICC_1: 1 bpp (black and white)

Use of this mode is necessary for monochrome displays with 1 bit per pixel.

Available colors: 2:

GUICC_2: 2 bpp (4 grayscales)

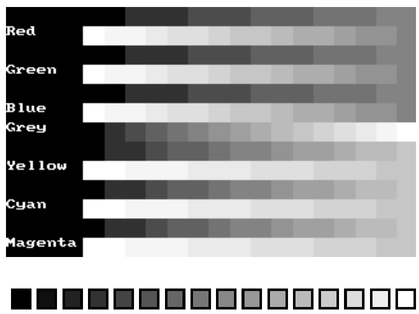
Use of this mode is necessary for monochrome displays with 2 bits per pixel.



Available colors: 2 x 2 = 4:

GUICC_4: 4 bpp (16 grayscales)

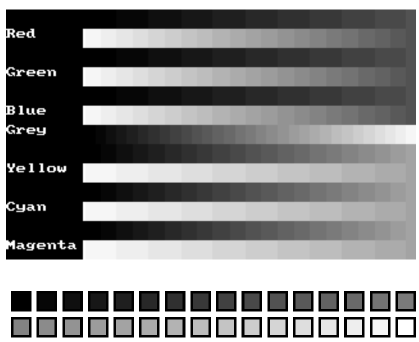
Use of this mode is necessary for monochrome displays with 4 bits per pixel.



Available colors: 2 x 2 x 2 x 2 = 16:

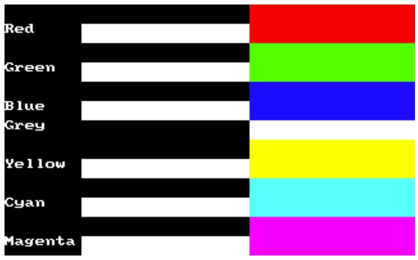
GUICC_5: 5 bpp (32 grayscales)

Use of this mode is necessary for monochrome displays with 5 bits per pixel.



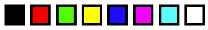
Available colors: 2 x 2 x 2 x 2 x 2 = 32:

GUICC_111: 3 bpp (2 levels per color)



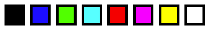
Use this mode if the basic 8 colors are enough, if your hardware supports only one bit per pixel and color or if you do not have sufficient video memory for a higher color depth.
Color mask: BGR

Available colors: 2 x 2 x 2 = 8:



GUICC_M111: 3 bpp (2 levels per color), red and blue swapped

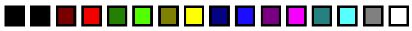
Use this mode if the basic 8 colors are enough, if your hardware supports only one bit per pixel and color or if you do not have sufficient video memory for a higher color depth. The available colors are the same as those in 111 mode.
Color mask: RGB
Available colors: 2 x 2 x 2 = 8:



GUICC_16: 4 bpp (16 colors)

This mode can be used if the basic 16 colors are enough, if the hardware supports only 4 bits per pixel or if you do not have sufficient video memory for a higher color depth.

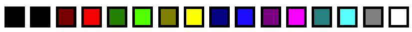
Available colors: 2 x 2 x 2 x 2 = 16:



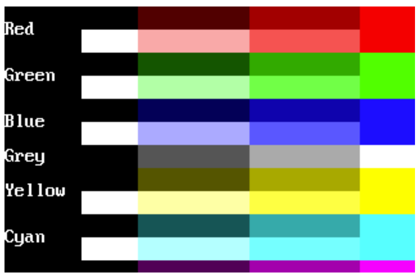
GUICC_1616I: 8 bpp (16 colors + 4 bits alpha mask)

Same colors as in GUICC_16. The lower 4 bits contain the color and the upper 4 bits are used for alpha blending.
Color mask: AAAACCCC
(AAAA = 0xF - opaque)
(AAAA = 0x0 - transparent)

Available colors: 2 x 2 x 2 x 2 = 16:

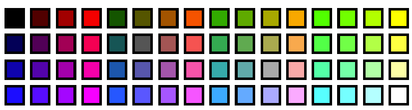


GUICC_222: 6 bpp (4 levels per color)



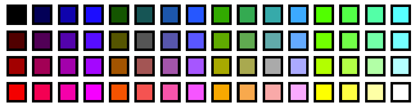
This mode is a good choice if your hardware does not have a palette for every individual color. 2 bits per pixel and color are reserved; usually 1 byte is used to store one pixel.
Color mask: BBGGRR

Available colors: 4 x 4 x 4 = 64:



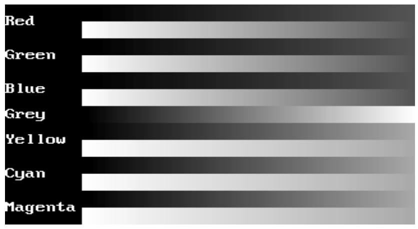
GUICC_M222: 6 bpp (4 levels per color), red and blue swapped

This mode is a good choice if your hardware does not have a palette for every individual color. 2 bits per pixel and color are reserved; usually 1 byte is used to store one pixel. The available colors are the same as those in 222 mode.
Color mask: RRGGBB
Available colors: 4 x 4 x 4 = 64:

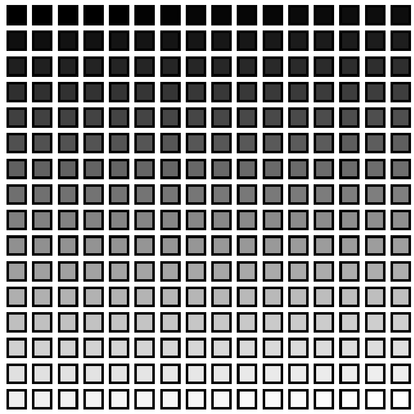


GUICC_8: 8 bpp (256 grayscales)

This mode uses 8 bpp for grayscales only. This is the smoothes possible grayscale mode.

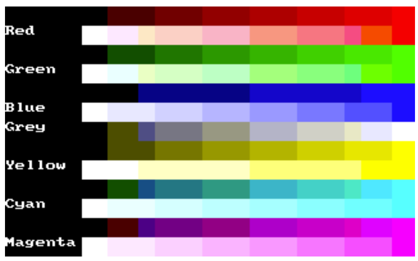


Available colors: 256 shades of gray.



GUICC_233: 8 bpp

This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. As shown in the picture, the result is 8 grades for green and red and 4 grades for blue. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: BBGGGRRR

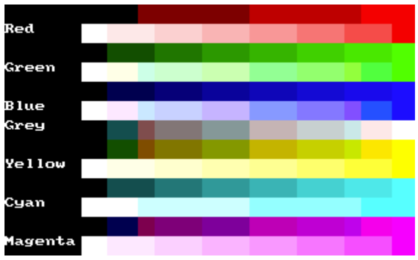


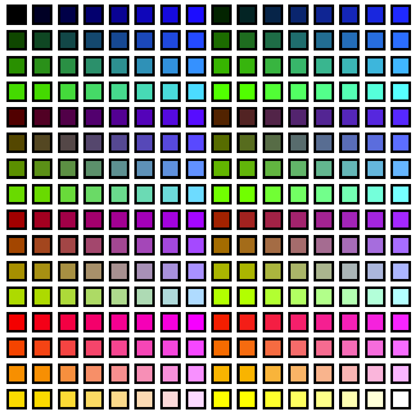
Available colors: 4 x 8 x 8 = 256:

GUICC_M233: 8 bpp, red and blue swapped

This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. The result is 8 grades for green and blue and 4 grades for red. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: RRGGGBBB

Available colors: 4 x 8 x 8 = 256:

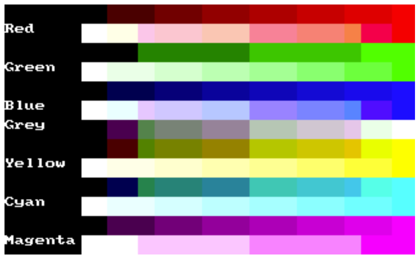




GUICC_323: 8 bpp

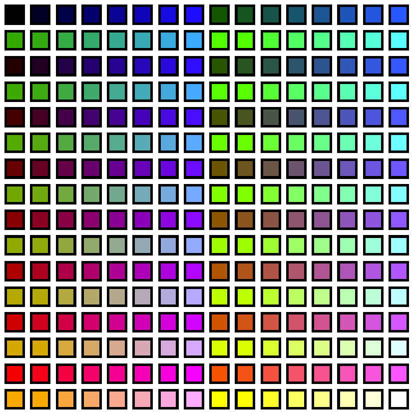
This mode supports 256 colors. 3 bits are used for the red and blue components of the color and 2 bits for the green component. As shown in the picture, the result is 8 grades for blue and red and 4 grades for green. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: BBBGGRRR

Available colors: 8 x 4 x 8 = 256:



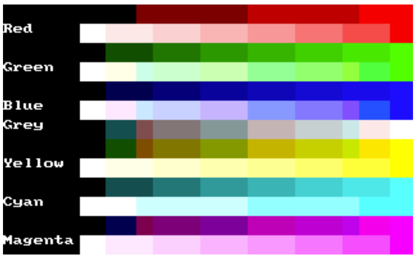
GUICC_M323: 8 bpp, red and blue swapped

This mode supports 256 colors. 3 bits are used for the red and blue components of the color and 2 bits for the green component. The available colors are the same as those in 323 mode. The result is 8 grades for red and blue and 4 grades for green. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: RRRGGBBB
Available colors: 8 x 4 x 8 = 256:

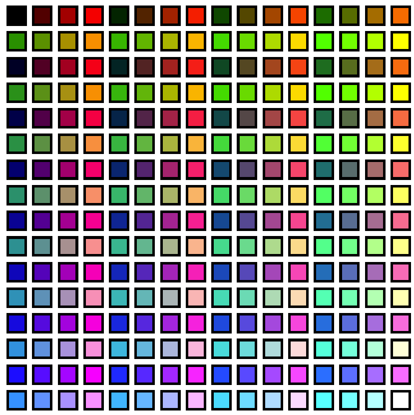


GUICC_332: 8 bpp

This mode supports 256 colors. 3 bits are used for the blue and green components of the color and 2 bits for the red component. As shown in the picture, the result is 8 grades for green and blue and 4 grades for red. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: BBBGGGRR

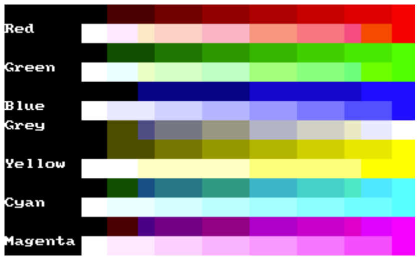


Available colors: 8 x 8 x 4 = 256:



GUICC_M332: 8 bpp, red and blue swapped

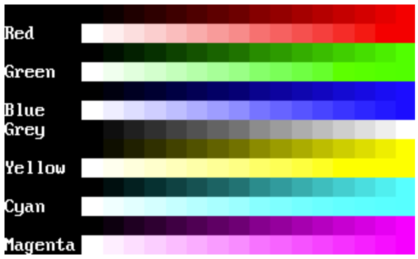
This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. The result is 8 grades for red and green and only 4 grades for blue. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: RRRGGGBB



Available colors: 8 x 8 x 4 = 256:

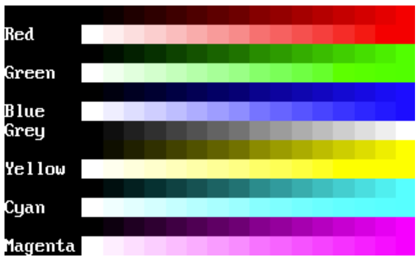
GUICC_444_12:

The red, green and blue components are each 4 bits.
Color mask: 0000BBBBGGGGRRRR
Available colors: 16 x 16 x 16 = 4096.



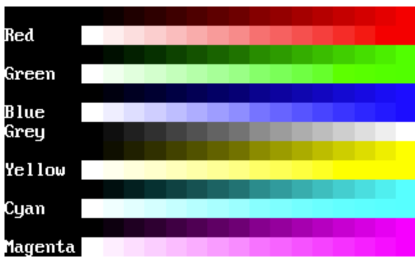
GUICC_444_16:

The red, green and blue components are each 4 bits. One bit between the color components is not used. The available colors are the same as those in 444_12 mode.
Color mask: 0BBBB0GGGG0RRRR0
Available colors: 16 x 16 x 16 = 4096.

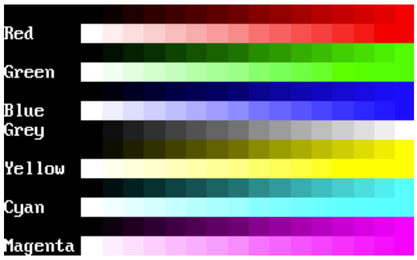


GUICC_M444_12: red and blue swapped

The red, green and blue components are each 4 bits. The available colors are the same as those in 444_12 mode.
Available colors: 16 x 16 x 16 = 4096.
Color mask: RRRRGGGGBBBB

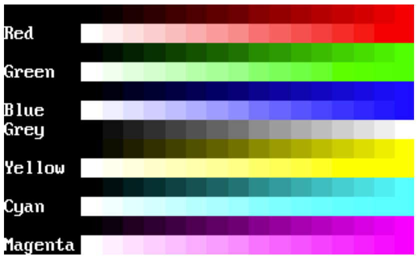


GUICC_M444_16: red and blue swapped



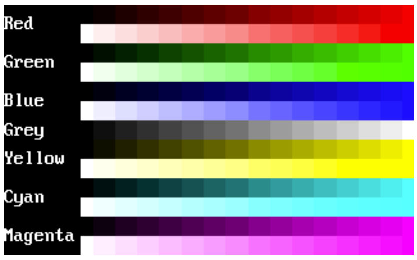
The red, green and blue components are each 4 bits. One bit between the color components is not used. The available colors are the same as those in 444_12 mode.
Color mask: 0RRRR0GGGG0BBBB0
Available colors: 16 x 16 x 16 = 4096.

GUICC_M444_12_1:



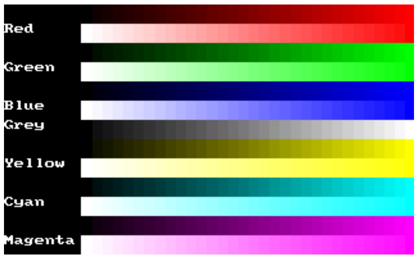
The red, green and blue components are each 4 bits. The lower 4 bits of the color mask are not used. The available colors are the same as those in 444_12 mode.
Color mask: BBBBGGGGRRRR0000
Available colors: 16 x 16 x 16 = 4096.

GUICC_M4444I: 12 bits colors + 4 bits alpha mask



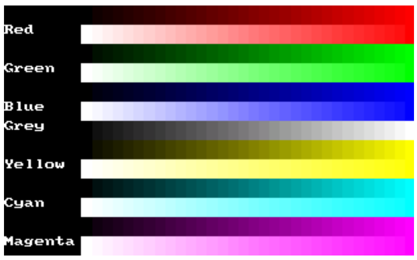
The red, green and blue components are each 4 bits, the upper 4 bits are used for alpha blending.
Color mask: AAAARRRRGGGGBBBB
(AAAA = 0xF - opaque)
(AAAA = 0x0 - transparent)
Available colors: 16 x 16 x 16 = 4096.

GUICC_555: 15 bpp



Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 15 bpp. The red, green and blue components are each 5 bits.
Color mask: BBBBBGGGGGRRRRR
Available colors: 32 x 32 x 32 = 32768.

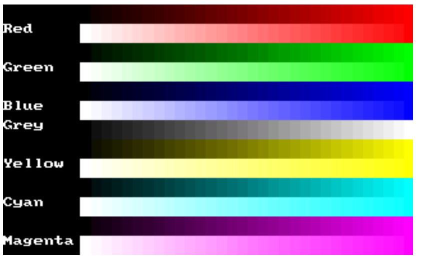
GUICC_M555: 15 bpp, red and blue swapped



Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 15 bpp. The red, green and blue components are each 5 bits. The available colors are the same as those in 555 mode.
Color mask: RRRRRGGGGGBBBBB
Available colors: 32 x 32 x 32 = 32768.

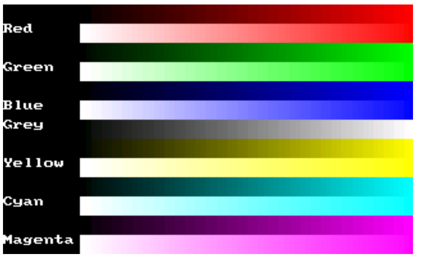
GUICC_M1555I: 15 bits colors + 1 bit transparency

The available colors are the same as those in 565 mode. The red, green and blue components are each 5 bits, the upper bit is used for transparency.
Color mask: ARRRRRGGGGGBBBBB
(A = 1 - opaque)
(A = 0 - transparent)
Available colors: 32 x 32 x 32 = 32768.



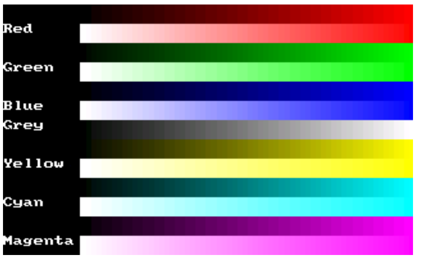
GUICC_565: 16 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The red and the blue component is 5 bits and the green component is 6 bit.
Color mask: BBBBBGGGGGGRRRRR
Available colors: 32 x 64 x 32 = 65536.



GUICC_M565: 16 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The available colors are the same as those in 565 mode.
Color sequence: RRRRRGGGGGGBBBBB
Available colors: 32 x 64 x 32 = 65536.



GUICC_556: 16 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The blue and the green component is 5 bit and the red component is 6 bit.
Color mask: BBBBBGGGGGRRRRRR
Available colors: 32 x 32 x 64 = 65536.

GUICC_M556: 16 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The red and the green component is 5 bit and the blue component is 6 bit.
Color mask: RRRRRGGGGGBBBBBB
Available colors: 32 x 32 x 64 = 65536.

GUICC_655: 16 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The red and green component is 5 bit and the blue component is 6 bit.
Color mask: BBBBBBGGGGGRRRRR
Available colors: 64 x 32 x 32 = 65536.

GUICC_M655: 16 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The blue and green component is 5 bit and the red component is 6 bit.
Color mask: RRRRRRGGGGGBBBBB

Available colors: 64 x 32 x 32 = 65536.

GUICC_666: 18 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 18 bpp. The red, green and blue component is 6 bit.
Color mask: BBBBBBGGGGGGRRRRRR
Available colors: 64 x 64 x 64 = 262144.

GUICC_M666: 18 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 18 bpp. The red, green and the blue component is 6 bit.
Color mask: RRRRRRGGGGGGBBBBBB
Available colors: 64 x 64 x 64 = 262144.

GUICC_666_9: 18 bpp

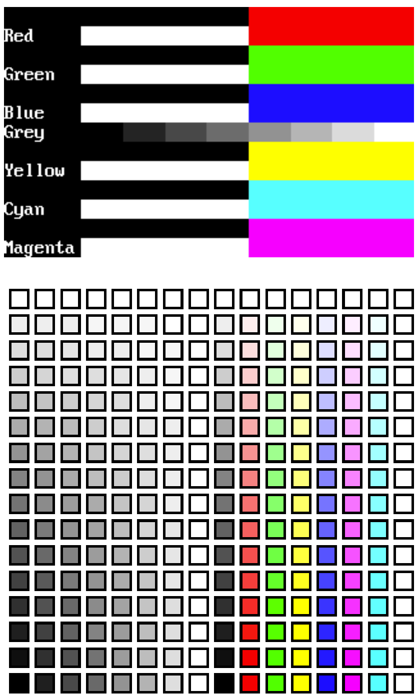
Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 18 bpp. The red, green and blue component is 6 bit.
Color mask: 0000000BBBBBBGGG0000000GGGRRRRRR
Available colors: 64 x 64 x 64 = 262144.

GUICC_M666_9: 18 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 18 bpp. The red, green and blue component is 6 bit.
Color mask: RRRRRRGGGGGGBBBBBB
Available colors: 64 x 64 x 64 = 262144.

GUICC_822216: 8 bpp, 2 levels per color + 8 grayscales + 16 levels of alpha blending

This mode can be used with a programmable color lookup table (LUT), supporting a total of 256 possible colors and alpha blending support. It supports the 8 basic colors, 8 grayscales and 16 levels of alpha blending for each color / grayscale. With other words it can be used if only a few colors are required but more levels of alpha blending.



Available colors: (2 x 2 x 2 + 8) * 16 = 256

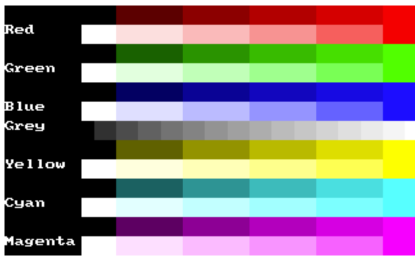
GUICC_84444: 8 bpp, 4 levels per color + 16 grayscales + 4(3) levels of alpha blending

This mode can be used with a programmable color lookup table (LUT), supporting a total of 240 possible colors and alpha blending support. 4 levels of intensity are available for each color, in addition to 16 grayscales and 4 levels of alpha blending for each color / grayscale. With other words it can be used if only a few levels of alpha blending are required and different shades of colors.

Available colors: (4 x 4 x 4 + 16) * 3 = 240

GUICC_8666: 8bpp, 6 levels per color + 16 grayscales

This mode is most frequently used with a programmable color lookup table (LUT), supporting a total of 256 possible colors using a palette. The screenshot gives an idea of the available colors; this mode contains the best choice for general purpose applications. Six levels of intensity are available for each color, in addition to 16 grayscales.



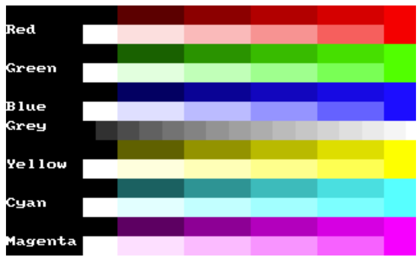
Available colors: 6 x 6 x 6 + 16 = 232:

GUICC_8666_1: 8bpp, 6 levels per color + 16 grayscales + transparency

This mode is most frequently used with MultiLayer configurations and a programmable color lookup table (LUT), supporting a total of 256 possible colors using a palette. The difference between 8666 and 86661 is, that the first color indices of the 86661 mode are not used. So the color conversion routine GUI_Color2Index does never return 0 which is used for transparency.
Available colors: 6 x 6 x 6 + 16 = 232.

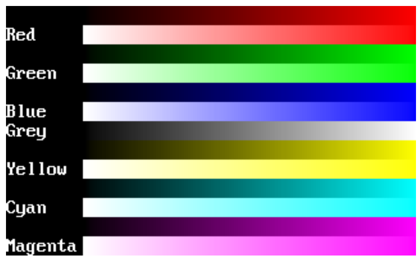
GUICC_88666I: 16bpp - 8 bits color (6 levels per color + 16 grayscales) + 8 bits alpha blending

The available colors of this mode are exactly the same as described under GUICC_8666. The upper 8 bits are used for alpha blending.
Color mask: AAAAAAAACCCCCCCC
(AAAAAAAA = 0xFF - opaque)
(AAAAAAAA = 0x00 - transparent)



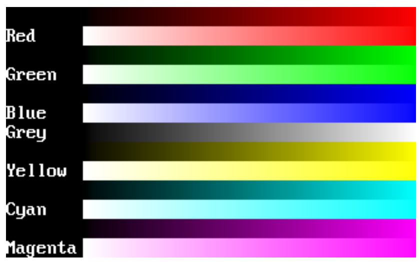
GUICC_888: 24 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 24 bpp. The red, green and blue components are each 8 bits.
Color mask: BBBBBBBBGGGGGGGGRRRRRRRR
Available colors: 256 x 256 x 256 = 16777216.



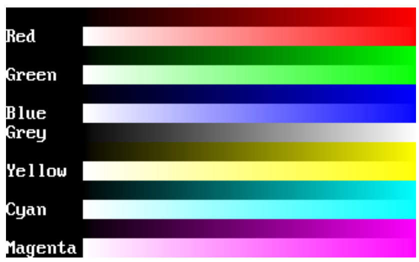
GUICC_M888: 24 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 24 bpp. The red, green and blue components are each 8 bits.
Color mask: RRRRRRRRGGGGGGGGBBBBBBBB
Available colors: 256 x 256 x 256 = 16777216.

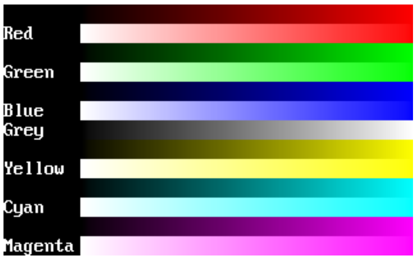


GUICC_8888: 32 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 32 bpp, where the lower 3 bytes are used for the color components and the upper byte is used for alpha blending. The red, green, blue and alpha blending components are each 8 bits.
Color mask: AAAAAAAABBBBBBBBGGGGGGGGR-RRRRRRR
Available colors: 256 x 256 x 256 = 16777216.

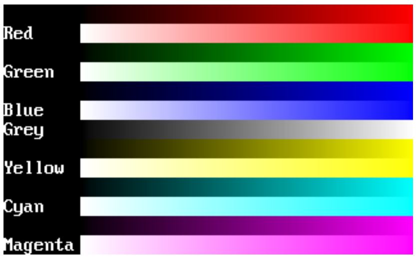


GUICC_M8888: 32 bpp, red and blue swapped



Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 32 bpp, where the lower 3 bytes are used for the color components and the upper byte is used for alpha blending. The red, green, blue and alpha blending components are each 8 bits.
Color mask: AAAAAAAARRRRRRRRGGGGGGGG-BBBBBBBB
Available colors: 256 x 256 x 256 = 16777216.

GUICC_M8888I: 32 bpp, red and blue swapped



The color mode is exactly the same as described under GUICC_M8888 with the difference, that alpha blending is inverted.
Color mask: AAAAAAAARRRRRRRRGGGGGGGG-BBBBBBBB
(AAAAAAAA = 0xFF - opaque)
(AAAAAAAA = 0x00 - transparent)

GUICC_0: Custom palette mode

How to use a custom palette mode is described in the according section on page 312.

GUICC_1_2, GUICC_1_4, ... GUICC_1_24

These color conversion routines make it possible, to use display drivers which require a color depth of more than 1bpp, with emWin packages containing no support for colors or grayscales. The routines ensure that each color of the whole palette of possible colors will be converted into black or white.

Example

If the available emWin package does not contain color- or gray scale support and only a driver, which requires index values of 16 bits is available, GUICC_1_16 can be used. This color conversion scheme ensures that each color of the whole 16 bit palette will be converted into 0xFFFF (normally white) or 0x0000 (normally black).

15.8 Application defined color conversion

If none of the fixed palette modes matches the need of color conversion this mode makes it possible to use application defined color conversion routines. The purpose of these routines is converting an RGB value into an index value for the hardware and vice versa.

Example of defining custom color conversion routines

The following example should explain how it works:

```
static unsigned _Color2Index_User(LCD_COLOR Color) {
  unsigned Index;
  /* Add code for converting the RGB value to an index value for the hardware */
  return Index;
}

static LCD_COLOR _Index2Color_User(unsigned Index) {
  LCD_COLOR Color;
  /* Add code for converting the index value into an RGB value */
  return Color;
}

static unsigned _GetIndexMask_User(void) {
  return 0xffff; /* Example for using 16 bits */
}

const LCD_API_COLOR_CONV LCD_API_ColorConv_User = {
  _Color2Index_User,
  _Index2Color_User,
  _GetIndexMask_User
};
```

The function `LCD_Color2Index_User()` is called by emWin if a RGB value should be converted into an index value for the display controller whereas the function `LCD_Index2Color_User()` is called if an index value should be converted into a RGB value.
`LCD_GetIndexMask_User()` should return a bit mask value, which has each bit set to 1 which is used by the display controller and unused bits should be set to 0. For example the index mask of `GUICC_44416` mode is 0BBBB0GGGG0RRRR0, where 0 stands for unused bits. The bit mask for this mode is 0x7BDE.

Example of using custom color conversion routines

As described in the chapter 'Configuration' a pointer to an API table is required for creating the display driver device. As shown in the example above the API table consists of function pointers to the color conversion routines.
A good location for the API table and the color conversion routines is the configuration file `LCDConf.c` located in the `Config` folder. The routines can be used as follow in the function `LCD_X_Config()` which is responsible to create the display driver device:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, &LCD_API_ColorConv_User, 0, 0);
  .
  .
  .
}
```


15.9 Custom palette mode

If none of the fixed palette modes fulfils the requirements of the application emWin is able to use a custom palette. A custom palette simply lists all the available colors in the same order as they are used by the hardware. This means that no matter what colors the display controller/display combination is able to display, emWin will be able to simulate them in the PC simulation and handle these colors correctly in the target system. Working with a custom palette requires a color depth <= 8 bpp.
A custom palette is typically configured during the initialization in the function `LCD_X_Config()` which is responsible for creating and configuring the display driver device. This requires setting the look-up table entries using the function `LCD_SetLUTEntryEx()` which in turn is called by the functions `LCD_SetLUT()` and `LCD_SetLUTEx()`. These functions are implemented in the custom palette mode module `GUICC_0.c`, but might require modification according to the used hardware. Detailed information can be found in the according function descriptions in the section “Look-up table API” on page 313.

Example

The following example should show how a custom palette can be used. It passes the palette to the function:

```
static const LCD_COLOR _aColors_16[] = {
  0x000000,  0x0000FF,  0x00FF00,  0x00FFFF,
  0xFF0000,  0xFF00FF,  0xFFFF00,  0xFFFFFF,
  0x000000,  0x000080,  0x008000,  0x008080,
  0x800000,  0x800080,  0x808000,  0x808080,
};

static const LCD_PHYSPALETTE _aPalette_16 = {
  COUNTOF(_aColors_16), _aColors_16
};

void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  .
  .
  .
  //
  // Set user palette data (only required if no fixed palette is used)
  //
  LCD_SetLUTEx(0, _aPalette_16);
}
```

Elements of structure LCD_PHYSPALETTE

Data type	Element	Description
`int`	`NumEntries`	Number of entries to be stored in the look-up table.
`const LCD_COLOR *`	`pPalEntries`	Pointer to an array of colors. The number of elements in this array has to match at least the value stored in NumEntries.

Table 15.3: LCD_PHYSPALETTE element list

15.9.1 Look-up table API

Function	Description
LCD_SetLUT()	Sets the look-up table for the currently selected layer.
LCD_SetLUTEx()	Sets the look-up table for the given layer.
LCD_SetLUTEntryEx()	Sets one entry in the look-up table.

LCD_SetLUT()

Description

Sets the look-up table for the currently selected layer. This function is defined in the module GUICC_0.c. It may require modification according to the used hardware.

Prototype

```
void LCD_SetLUT(const LCD_PHYSPALETTE * pPalette);
```

Parameter	Description
pPalette	Pointer to a LCD_PHYSPALETTE structure.

Table 15.4: LCD_SetLUT() parameter list

LCD_SetLUTEx()

Description

Sets the look-up table for the given layer. This function is defined in the module GUICC_0.c. It may require modification according to the used hardware.

Prototype

```
void LCD_SetLUTEx(int LayerIndex, const LCD_PHYSPALETTE * pPalette);
```

Parameter	Description
LayerIndex	Index of the layer to set the look-up table for.
pPalette	Pointer to an LCD_PHYSPALETTE structure.

Table 15.5: LCD_SetLUTEx() parameter list

LCD_SetLUTEntryEx()

Description

Sets one entry in the look-up table.

Prototype

```
int LCD_SetLUTEntryEx(int LayerIndex, U8 Pos, LCD_COLOR Color);
```

Parameter	Description
LayerIndex	Index of the layer the look-up entry has to be set for.
Pos	Position in the look-up table to use for this color.
Color	32-bit color value.

Table 15.6: LCD_SetLUTEntryEx() parameter list

Return value

0 on success. 1 on error.

15.10 Gamma correction

Gamma correction can simply be achieved with custom color conversion routines. The trick is converting the colors twice. Please note that gamma correction does not work within the simulation.

Color2Index - conversion

It should first make the gamma correction of the color to be converted. The result of the gamma correction then should be passed to the Color2Index-function of the desired fixed palette mode, whose result then should be returned.

Index2Color - conversion

It should first convert the index to a color with the Color2Index-function of the desired fixed palette mode. The result then should be passed to the gamma correction routine whose result then should be returned.

Example

The sample folder `LCDConf\Common\` contains the sample file `LCDConf_GammaCorrection.c`. It shows in detail how gamma correction can be used.

15.11 Color API

The following table lists the available color-related functions in alphabetical order within their respective categories. Detailed description of the routines can be found in the sections that follow.

Routine	Description
Basic functions	
GUI_GetBkColor()	Return the current background color.
GUI_GetBkColorIndex()	Return the index of the current background color.
GUI_GetColor()	Return the current foreground color.
GUI_GetColorIndex()	Return the index of the current foreground color.
GUI_SetBkColor()	Set the current background color.
GUI_SetBkColorIndex()	Set the index of the current background color.
GUI_SetColor()	Set the current foreground color.
GUI_SetColorIndex()	Set the index of the current foreground color.
Conversion functions	
GUI_CalcColorDist()	Returns the difference between 2 colors
GUI_CalcVisColorError()	Returns the difference to the next available color
GUI_Color2Index()	Convert color into color index.
GUI_Color2VisColor()	Returns the nearest available color
GUI_ColorIsAvailable()	Checks if given color is available
GUI_Index2Color()	Convert color index into color.

Table 15.7: Color API list

15.11.1 Basic functions

GUI_GetBkColor()

Description

Returns the current background color.

Prototype

```
GUI_COLOR GUI_GetBkColor(void);
```

Return value

The current background color.

GUI_GetBkColorIndex()

Description

Returns the index of the current background color.

Prototype

```
int GUI_GetBkColorIndex(void);
```

Return value

The current background color index.

GUI_GetColor()

Description

Returns the current foreground color.

Prototype

```
GUI_COLOR GUI_GetColor(void);
```

Return value

The current foreground color.

GUI_GetColorIndex()

Description

Returns the index of the current foreground color.

Prototype

```
int GUI_GetColorIndex(void);
```

Return value

The current foreground color index.

GUI_SetBkColor()

Description

Sets the current background color.

Prototype

```
GUI_COLOR GUI_SetBkColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color for background, 24-bit RGB value.

Table 15.8: GUI_SetBkColor() parameter list

Return value

The selected background color.

GUI_SetBkColorIndex()

Description

Sets the index of the current background color.

Prototype

```
int GUI_SetBkColorIndex(int Index);
```

Parameter	Description
Index	Index of the color to be used.

Table 15.9: GUI_SetBkColorIndex() parameter list

Return value

The selected background color index.

GUI_SetColor()

Description

Sets the current foreground color.

Prototype

```
void GUI_SetColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color for foreground, 24-bit RGB value.

Table 15.10: GUI_SetColor() parameter list

Return value

The selected foreground color.

GUI_SetColorIndex()

Description

Sets the index of the current foreground color.

Prototype

```
void GUI_SetColorIndex(int Index);
```

Parameter	Description
Index	Index of the color to be used.

Table 15.11: GUI_SetColorIndex() parameter list

Return value

The selected foreground color index.

15.11.2 Conversion functions

GUI_CalcColorDist()

Calculates the distance between 2 colors. The distance will be calculated by the sum of the square value from the distances of the red, green and the blue component:
Difference = $(Red1 - Red0)^2 + (Green1 - Green0)^2 + (Blue1 - Blue0)^2$

Prototype

```
U32 GUI_CalcColorDist(GUI_COLOR Color0, GUI_COLOR Color1);
```

Parameter	Description
Color0	RGB value of the first color.
Color1	RGB value of the second color.

Table 15.12: GUI_CalcColorDist() parameter list

Return value

The distance as described above.

GUI_CalcVisColorError()

Calculates the distance to the next available color. For details about the calculation, refer to “GUI_CalcColorDist()” on page 317.

Prototype

```
U32 GUI_CalcVisColorError(GUI_COLOR Color);
```

Parameter	Description
Color	RGB value of the color to be calculated.

Table 15.13: GUI_CalcVisColorError() parameter list

Return value

The distance to the next available color.

GUI_Color2Index()

Returns the index of a specified RGB color value.

Prototype

```
int GUI_Color2Index(GUI_COLOR Color);
```

Parameter	Description
Color	RGB value of the color to be converted.

Table 15.14: GUI_Color2Index() parameter list

Return value

The color index.

GUI_Color2VisColor()

Returns the next available color of the system as an RGB color value.

Prototype

```
GUI_COLOR GUI_Color2VisColor(GUI_COLOR Color);
```

Parameter	Description
Color	RGB value of the color.

Table 15.15: GUI_Color2VisColor() parameter list

Return value

The RGB color value of the nearest available color.

GUI_ColorIsAvailable()

Checks if the given color is available.

Prototype

```
char GUI_ColorIsAvailable(GUI_COLOR Color);
```

Parameter	Description
Color	RGB value of the color.

Table 15.16: GUI_ColorIsAvailable() parameter list

Return value

1 if color is available, 0 if not.

GUI_Index2Color()

Returns the RGB color value of a specified index.

Prototype

```
int GUI_Index2Color(int Index);
```

Parameter	Description
Index	Index of the color. to be converted

Table 15.17: GUI_Index2Color() parameter list

Return value

The RGB color value.

Chapter 16

Memory Devices

Memory Devices can be used in a variety of situations, mainly to prevent the display from flickering when using drawing operations for overlapping items. The basic idea is quite simple. Without the use of a Memory Device, drawing operations write directly to the display. The screen is updated as drawing operations are executed, which gives it a flickering appearance as the various updates are made. For example, if you want to draw a bitmap in the background and some transparent text in the foreground, you would first have to draw the bitmap and then the text. The effect would be a flickering of the text.

If a Memory Device is used for such a procedure, however, all drawing operations are executed in memory. The final result is displayed on the screen only when all operations have been carried out, with the advantage of no flickering. This difference can be seen in the example in the following section, which illustrates a sequence of drawing operations both with and without the use of a Memory Device.

The distinction may be summarized as follows: If no Memory Device is used, the effects of drawing operations can be seen step by step, with the disadvantage of a flickering display. With a Memory Device, the effects of all routines are made visible as a single operation. No intermediate steps can actually be seen. The advantage, as explained above, is that display flickering is completely eliminated, and this is often desirable.

Memory Devices are an additional (optional) software item and are not shipped with the emWin basic package. The software for Memory Devices is located in the subdirectory `GUI\Memdev`.

16.1 Using Memory Devices: Illustration

The following table shows screenshots of the same operations handled with and without a Memory Device. The objective in both cases is identical: a work piece is to be rotated and labeled with the respective angle of rotation (here, 10 degrees). In the first case (without a Memory Device) the screen must be cleared, then the polygon is redrawn in the new position and a string with the new label is written. In the second case (with a Memory Device) the same operations are performed in memory, but the screen is not updated during this time. The only update occurs when the routine `GUI_MEMDEV_CopyToLCD()` is called, and this update reflects all the operations at once. Note that the initial states and final outputs of both procedures are identical.

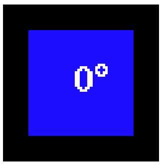
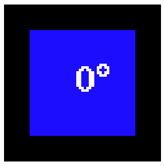
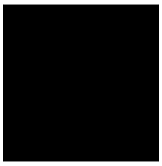
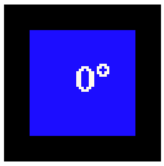
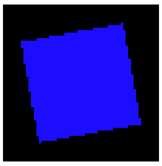
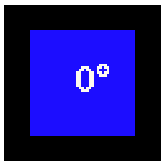
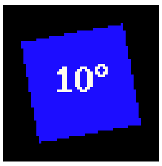
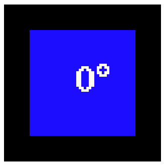
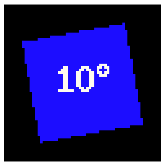
API function	Without Memory Device	With Memory Device
Step 0: Initial state		
Step 1: `GUI_Clear()`		
Step 2: `GUI_DrawPolygon()`		
Step 3: `GUI_DispString()`		
Step 4: `GUI_MEMDEV_CopyToLCD()` (only when using Memory Device)		

Table 16.1: Memory device usage

16.2 Supported color depth (bpp)

Memory Devices are available in 4 different color depth:
1 bpp, 8 bpp, 16 bpp and 32 bpp.

Creating Memory Devices "compatible" to the display

There are two ways to create Memory Devices. If they are use to avoid flickering, a Memory Device compatible to the display is created. This "compatible" Memory Device needs to have the same or a higher color depth as the display. emWin automatically selects the "right" type of Memory Device for the display if the functions `GUI_MEMDEV_Create()`, `GUI_MEMDEV_CreateEx()` are used.
The Window Manager, which also has the ability to use Memory Devices for some or all windows in the system, also uses these functions.
This way, the Memory Device with the lowest color depth (using the least memory) is automatically used.

Creating Memory Devices for other purposes

Memory Devices of any type can be created using `GUI_MEMDEV_CreateFixed()`. A typical application would be the use of a Memory Device for printing as described later in this chapter.

16.3 Memory Devices and the Window Manager

The Window Manager works seamlessly with Memory Devices. Every window has a flag which tells the Window Manager if a Memory Device should be used for rendering. This flag can be specified when creating the window or set/reset at any time.
If the Memory Device flag is set for a particular window, the WM automatically uses a Memory Device when drawing the window. It creates a Memory Device before drawing a window and deletes it after the drawing operation. If enough memory is available, the whole window fits into the size of the Memory Device created by the WM. If not enough memory is available for the complete window in one Memory Device, the WM uses 'banding' for drawing the window. Details about 'banding' are described in the documentation, chapter 'Memory Devices \ Banding Memory Device'. The memory used for the drawing operation is only allocated during the drawing operation. If there is not enough memory available when (re-)drawing the window, the window is redrawn without Memory Device.

16.4 Memory Devices and multiple layers

The Memory Device API does not offer any option to specify a layer. Memory Devices are associated with the current layer at creation. The color conversion settings of the current layer are automatically used by Memory Devices.

Example

```
//
// Create a Memory Device associated with layer 1
//
GUI_SelectLayer(1);
hMem = GUI_MEMDEV_Create(0, 0, 100, 100);
GUI_MEMDEV_Select(hMem);
GUI_DrawLine(0, 0, 99, 99);
GUI_MEMDEV_Select(0);
//
// Select layer 0
//
GUI_SelectLayer(0);
//
// The following line copies the Memory Device to layer 1 and not to layer 0
//
GUI_MEMDEV_CopyToLCD(hMem);
```


16.5 Memory requirements

If creating a Memory Device the required number of bytes depends on the color depth of the Memory Device and whether transparency support is needed or not.

Memory usage without transparency support

The following table shows the memory requirement in dependence of the system color depth for Memory Devices without transparency support.

Color depth of Memory Device	System color depth (LCD_BITSPERPIXEL)	Memory usage
1 bpp	1 bpp	1 byte / 8 pixels: (XSIZE + 7) / 8 * YSIZE
8 bpp	2, 4 and 8 bpp	XSIZE * YSIZE
16 bpp	12 and 16 bpp	2 bytes / pixel: XSIZE * YSIZE * 2
32 bpp	18, 24 and 32 bpp	4 bytes / pixel: XSIZE * YSIZE * 4

Table 16.2: Memory usage without transparency

Example:

A Memory Device of 111 pixels in X and 33 pixels in Y should be created. It should be compatible to a display with a color depth of 12 bpp and should support transparency. The required number of bytes can be calculated as follows:

Number of required bytes = (111 * 2 + (111 + 7) / 8) * 33 = 7788 bytes

Memory usage with transparency support

If a Memory Device should support transparency it needs one additional byte / 8 pixels for internal management.

Color depth of Memory Device	System color depth (LCD_BITSPERPIXEL)	Memory usage
1 bpp	1 bpp	2 byte / 8 pixels: (XSIZE + 7) / 8 * YSIZE * 2
8 bpp	2, 4 and 8 bpp	1 bytes / pixel + 1 byte / 8 pixels: (XSIZE + (XSIZE + 7) / 8) * YSIZE
16 bpp	12 and 16 bpp	2 bytes / pixel + 1 byte / 8 pixels: (XSIZE * 2 + (XSIZE + 7) / 8) * YSIZE
32 bpp	18, 24 and 32 bpp	4 bytes / pixel + 1 byte / 8 pixels: (XSIZE * 4 + (XSIZE + 7) / 8) * YSIZE

Table 16.3: Memory usage with transparency

Example:

A Memory Device of 200 pixels in X and 50 pixels in Y should be created. It should be compatible to a display with a color depth of 4bpp and should support transparency. The required number of bytes can be calculated as follows:

```
Number of required bytes = (200 + (200 + 7) / 8) * 50 = 11250 bytes
```

Memory usage with window animation functions

One static Memory Device is created for each, the given window and all of its child window. The color depth is always 32bpp.

16.6 Performance

Using Memory Devices typically does not significantly affect performance. When Memory Devices are used, the work of the driver is easier: It simply transfers bitmaps to the display controller. On systems with slow drivers (for example displays connected via serial interface), the performance is better if Memory Devices are used; on systems with a fast driver (such as memory mapped display memory, GUIDRV_Lin and others) the use of Memory Devices costs some performance.
If 'banding' is needed, the used time to draw a window increases with the number of bands. The more memory available for Memory Devices, the better the performance.

16.7 Basic functions

The following routines are those that are normally called when using Memory Devices. Basic usage is rather simple:

1. Create the Memory Device (using `GUI_MEMDEV_Create()`).
2. Activate it (using `GUI_MEMDEV_Select()`).
3. Execute drawing operations.
4. Copy the result into the display (using `GUI_MEMDEV_CopyToLCD()`).
5. Delete the Memory Device if you no longer need it (using `GUI_MEMDEV_Delete()`).

16.8 In order to be able to use Memory Devices...

Memory Devices are enabled by default. In order to optimize performance of the software, support for Memory Devices can be switched off in the configuration file `GUIConf.h` by including the following line:

```
#define GUI_SUPPORT_MEMDEV        0
```

If this line is in the configuration file and you want to use Memory Devices, either delete the line or change the define to 1.

16.9 MultiLayer / MultiDisplay configuration

As explained earlier in this chapter Memory Devices "compatible" to the display needs to have the same or a higher color depth as the display. When creating a Memory Device compatible to the display emWin "knows" the color depth of the currently selected layer/display and automatically uses the lowest color depth.

16.10 Configuration options

Type	Macro	Default	Description
B	GUI_USE_MEMDEV_1BPP_FOR_SCREEN	1	Enables the use of 1bpp Memory Devices with displays of 1bpp color depth.

Table 16.4: Configuration description

GUI_USE_MEMDEV_1BPP_FOR_SCREEN

On systems with a display color depth <= 8bpp the default color depth of Memory Devices compatible to the display is 8bpp. To enable the use of 1bpp Memory Devices with displays of 1bpp color depth the following line should be added to the configuration file GUIConf.h:

```
#define GUI_USE_MEMDEV_1BPP_FOR_SCREEN 0
```


16.11 Memory Device API

The table below lists the available routines of the emWin Memory Device API. All functions are listed in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Description
Basic functions	
GUI_MEMDEV_Clear()	Marks the Memory Device contents as unchanged
GUI_MEMDEV_ClearAlpha()	Clears all alpha value in the given memory device
GUI_MEMDEV_CopyFromLCD()	Reads back the content of the display and stores it in the given Memory Device.
GUI_MEMDEV_CopyToLCD()	Copies contents of Memory Device to LCD
GUI_MEMDEV_CopyToLCDAA()	Copies the contents of Memory Device anti-aliased.
GUI_MEMDEV_CopyToLCDAt()	Copies contents of Memory Device to LCD at the given. position
GUI_MEMDEV_Create()	Creates the Memory Device (first step).
GUI_MEMDEV_CreateEx()	Creates the Memory Device with additional creation flags.
GUI_MEMDEV_CreateFixed()	Creates a Memory Device with a given color depth.
GUI_MEMDEV_CreateFixed32()	Creates a 32bpp Memory Device.
GUI_MEMDEV_Delete()	Frees the memory used by the Memory Device.
GUI_MEMDEV_DrawPerspectiveX()	Draws the given Memory Device perspectively distorted into the current selected device.
GUI_MEMDEV_GetDataPtr()	Returns a pointer to the data area for direct manipulation.
GUI_MEMDEV_GetXSize()	Returns the X-size (width) of Memory Device.
GUI_MEMDEV_GetYSize()	Returns the Y-size (height) of Memory Device.
GUI_MEMDEV_MarkDirty()	Marks a rectangle area as dirty.
GUI_MEMDEV_PunchOutDevice()	Punches the given shape out of the Memory Device.
GUI_MEMDEV_ReduceYSize()	Reduces Y-size of Memory Device.
GUI_MEMDEV_Rotate()	Rotates and scales a Memory Device and writes the result into a Memory Device using the 'nearest neighbor' method.
GUI_MEMDEV_RotateAlpha()	Rotates and scales a Memory Device and blends in the result into a Memory Device using the 'nearest neighbor' method and the given alpha value.
GUI_MEMDEV_RotateHQ()	Rotates and scales a Memory Device and writes the result into a Memory Device using the 'high quality' method.
GUI_MEMDEV_RotateHQAlpha()	Rotates and scales a Memory Device and blends in the result into a Memory Device using the 'high quality' method and the given alpha value.
GUI_MEMDEV_RotateHQHR()	Rotates and scales a Memory Device and writes the result into a Memory Device using the 'high quality' as well as the 'high resolution' method.
GUI_MEMDEV_RotateHQT()	Rotates and scales a Memory Device and writes the result into a Memory Device using the 'high quality' method. (Optimized for images with a large amount of transparent pixels)
GUI_MEMDEV_RotateHR()	Rotates and scales a Memory Device and writes the result into a Memory Device using the 'high resolution' method.
GUI_MEMDEV_Select()	Selects a Memory Device as target for drawing operations.
GUI_MEMDEV_SerializeBMP()	Creates a BMP file from the given Memory Device.
GUI_MEMDEV_SetOrg()	Changes the origin of the Memory Device on the LCD.

Table 16.5: Memory Device API list

Routine	Description
GUI_MEMDEV_Write()	Writes the content of a Memory Device into the currently selected device considering the alpha channel.
GUI_MEMDEV_WriteAlpha()	Writes the content of a Memory Device into the currently selected device considering the alpha channel and an additional alpha value.
GUI_MEMDEV_WriteAlphaAt()	Writes the content of a Memory Device into the currently selected device using the given position considering the alpha channel and an additional alpha value.
GUI_MEMDEV_WriteAt()	Writes the content of a Memory Device into the currently selected device to the given position considering the alpha channel.
GUI_MEMDEV_WriteEx()	Writes the content of a Memory Device into the currently selected device considering an additional alpha value and scaling.
GUI_MEMDEV_WriteExAt()	Writes the content of a Memory Device into the currently selected device to the given position considering an additional alpha value and scaling.
GUI_MEMDEV_WriteOpaque()	Writes the content of a Memory Device into the currently selected device without considering the alpha channel.
GUI_MEMDEV_WriteOpaqueAt()	Writes the content of a Memory Device into the currently selected device to the given position without considering the alpha channel.
GUI_SelectLCD()	Selects the LCD as target for drawing operations.
Banding Memory Device	
GUI_MEMDEV_Draw()	Use a Memory Device for drawing.
Auto device object functions	
GUI_MEMDEV_CreateAuto()	Creates an auto device object.
GUI_MEMDEV_DeleteAuto()	Deletes an auto device object.
GUI_MEMDEV_DrawAuto()	Uses a GUI_AUTODEV object for drawing.
Measurement device object functions	
GUI_MEASDEV_ClearRect()	Clears the measurement rectangle.
GUI_MEASDEV_Create()	Creates a measurement device.
GUI_MEASDEV_Delete()	Deletes a measurement device.
GUI_MEASDEV_GetRect()	Retrieves the measurement result.
GUI_MEASDEV_Select()	Selects a measurement device as target for drawing operations.
Animation functions	
GUI_MEMDEV_FadeInDevices()	Performs fading from the background device to the foreground device.
GUI_MEMDEV_FadeOutDevices()	Performs fading from the foreground device to the background device.
GUI_MEMDEV_SetAnimationCallback()	Sets a user defined function to be called while animations are processed.
GUI_MEMDEV_SetTimePerFrame()	Sets the minimum period of one animation frame.
Animation functions (Window Manager required)	
GUI_MEMDEV_FadeInWindow()	Fades in a window by decreasing the alpha value.
GUI_MEMDEV_FadeOutWindow()	Fades out a window by increasing the alpha value.
GUI_MEMDEV_MoveInWindow()	Moves in a Window from a specified to its actual position by magnification (optionally with rotation).
GUI_MEMDEV_MoveOutWindow()	Moves out a Window from its actual to a specified position by demagnification (optionally with rotation).
GUI_MEMDEV_ShiftInWindow()	Shifts a Window in a specified direction into the screen to its actual position.
GUI_MEMDEV_ShiftOutWindow()	Shifts a Window in a specified direction from its actual position out of the screen.

Table 16.5: Memory Device API list

Routine	Description
GUI_MEMDEV_SwapWindow()	Swaps a window with the old content of the target area.
Blending, Blurring and Dithering functions	
GUI_MEMDEV_BlendColor32()	Blends a window with the given color and blending intensity.
GUI_MEMDEV_CreateBlurredDevice32()	Creates a blurred copy of the given Memory Device using the currently set blurring function.
GUI_MEMDEV_CreateBlurredDevice32HQ()	Creates a blurred copy of the given Memory Device using the high quality algorithm.
GUI_MEMDEV_CreateBlurredDevice32LQ()	Creates a blurred copy of the given Memory Device using the low quality algorithm.
GUI_MEMDEV_Dither32()	Dithers the given device using the given fixed color mode.
GUI_MEMDEV_SetBlurHQ()	Sets the blurring behavior of the function GUI_MEMDEV_CreateBlurredDevice32() to HQ.
GUI_MEMDEV_SetBlurLQ()	Sets the blurring behavior of the function GUI_MEMDEV_CreateBlurredDevice32() to LQ.
Blending and Blurring functions (Window Manager required)	
GUI_MEMDEV_BlendWinBk()	Blends the background of a window.
GUI_MEMDEV_BlurAndBlendWinBk()	Blurs and blends the background of a window.
GUI_MEMDEV_BlurWinBk()	Blurs the background of a window.

Table 16.5: Memory Device API list

16.11.1 Basic functions

GUI_MEMDEV_Clear()

Description

Marks the entire contents of a Memory Device as "unchanged".

Prototype

```
void GUI_MEMDEV_Clear(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to a Memory Device.

Table 16.6: GUI_MEMDEV_Clear() parameter list

Additional information

The next drawing operation with `GUI_MEMDEV_CopyToLCD()` will then write only the bytes modified between `GUI_MEMDEV_Clear()` and `GUI_MEMDEV_CopyToLCD()`.

GUI_MEMDEV_ClearAlpha()

Description

Clears all alpha values in the given Memory Device with the help of an 1bpp Memory Device mask. The Memory Device mask must have the same dimensions as the given `hMemData` Memory Device.
To use the Memory Device mask, an area has to be specified where the alpha values should be cleared. To do so basic drawing functions should be used. The function sets all pixels of the data memory device to opaque, which have the index value 1 in the mask memory device.

Prototype

```
int GUI_MEMDEV_ClearAlpha(GUI_MEMDEV_Handle hMemData,
```



```
GUI_MEMDEV_Handle hMemMask);
```

Parameter	Description
hMemData	Handle to a Memory Device. Must be a 32bpp Memory Device.
hMemMask	Optional handle to a Memory Device mask. Must be an 1bpp Memory Device with the same size as the hMemData handle.

Table 16.7: GUI_MEMDEV_ClearAlpha() parameter list

Return value

0 on success. 1 on error.

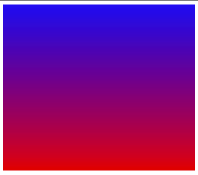
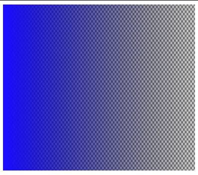
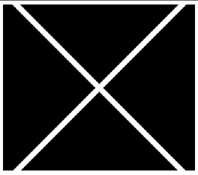
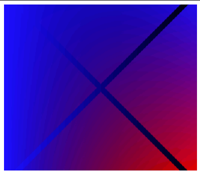
Additional information

If `hMemMask` is 0, the alpha values of the whole memory device will be set to opaque. No transparency remains then.

Example

```
#include "GUI.h"

/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  GUI_MEMDEV_Handle hMemData;
  GUI_MEMDEV_Handle hMemMask;

  GUI_Init();
  //
  // Background
  //
  GUI_SetBkColor(GUI_DARKBLUE);
  GUI_Clear();
  GUI_DrawGradientV(0, 0, 320, 240, GUI_BLUE, GUI_RED);
  //
  // Mask device
  //
  hMemMask = GUI_MEMDEV_CreateFixed(0, 0, 320, 240, GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_1,  GUICC_1);
  GUI_MEMDEV_Select(hMemMask);
  GUI_SetPenSize(8);
  GUI_DrawLine(0, 240, 320, 0);
  GUI_DrawLine(0, 0, 320, 240);
  GUI_MEMDEV_Select(0);
  //
  // Data Device
  //
  hMemData = GUI_MEMDEV_CreateFixed(0, 0, 320, 240, GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUICC_8888);
  GUI_MEMDEV_Select(hMemData);
  GUI_Clear();
  GUI_DrawGradientH(0, 0, 320, 240, GUI_BLUE, GUI_TRANSPARENT);
  GUI_MEMDEV_Select(0);
  //
  // Calling GUI_MEMDEV_ClearAlpha()
  //
  GUI_MEMDEV_ClearAlpha(hMemData, hMemMask);
  //
  // Result
  //
  GUI_MEMDEV_Write(hMemData);
  while (1) {
    GUI_Delay(100);
  }
}
```


Screenshots

Background	Data	Mask	Result
			

GUI_MEMDEV_CopyFromLCD()

Description

Reads back the content of the display and stores it in the given Memory Device.

Prototype

```
void GUI_MEMDEV_CopyFromLCD(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to a Memory Device.

Table 16.8: GUI_MEMDEV_CopyFromLCD() parameter list

GUI_MEMDEV_CopyToLCD()

Description

Copies the contents of a Memory Device from memory to the LCD.

Prototype

```
void GUI_MEMDEV_CopyToLCD(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to a Memory Device.

Table 16.9: GUI_MEMDEV_CopyToLCD() parameter list

Additional information

This function ignores the clipping area of the Window Manager as well as the alpha channel. Therefor using this function from within a paint event is not recommended. In order to display a Memory Device regarding the clipping area as well as the alpha channel, the function `GUI_MEMDEV_WriteAt()` should be used instead.

GUI_MEMDEV_CopyToLCDAA()

Description

Copies the contents of a Memory Device (antialiased) to the LCD.

Prototype

```
void GUI_MEMDEV_CopyToLCDAA(GUI_MEMDEV_Handle MemDev);
```

Parameter	Description
hMem	Handle to a Memory Device.

Table 16.10: GUI_MEMDEV_CopyToLCDAA() parameter list

Additional information

The device data is handled as antialiased data. A matrix of 2x2 pixels is converted to 1 pixel. The intensity of the resulting pixel depends on how many pixels are set in the matrix.

Example

Creates a Memory Device and selects it for output. A large font is then set and a text is written to the Memory Device:

```
GUI_MEMDEV_Handle hMem = GUI_MEMDEV_Create(0,0,60,32);
GUI_MEMDEV_Select(hMem);
GUI_SetFont(&GUI_Font32B_ASCII);
GUI_DispString("Text");
GUI_MEMDEV_CopyToLCDAA(hMem);
```

Screenshot of above example

GUI_MEMDEV_CopyToLCDAt()

Description

Copies the contents of a Memory Device to the LCD at the given position.

Prototype

```
void GUI_MEMDEV_CopyToLCDAt(GUI_MEMDEV_Handle hMem, int x, int y);
```

Parameter	Description
hMem	Handle to a Memory Device.
x	Position in X
y	Position in Y

Table 16.11: GUI_MEMDEV_CopyToLCDAt() parameter list

GUI_MEMDEV_Create()

Description

Creates a Memory Device.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_Create(int x0, int y0, int xSize, int ySize);
```

Parameter	Description
x0	X-position of the Memory Device.
y0	Y-position of the Memory Device.
xSize	X-size of the Memory Device.
ySize	Y-size of the Memory Device.

Table 16.12: GUI_MEMDEV_Create() parameter list

Return value

Handle of the created Memory Device. If the routine fails the return value is 0.

GUI_MEMDEV_CreateEx()

Description

Creates a Memory Device.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateEx(int x0, int y0, int xSize, int xSize,
```

```
int Flags);
```

Parameter	Description
x0	x-position of the Memory Device.
y0	y-position of the Memory Device.
xsize	x-size of the Memory Device.
ysize	y-size of the Memory Device.
Flags	See table below.

Table 16.13: GUI_MEMDEV_CreateEx() parameter list

Permitted values for parameter Flags	
GUI_MEMDEV_HASTRANS (recommended)	Default: The Memory Device is created with a transparency flag which ensures that the back-ground will be drawn correctly.
GUI_MEMDEV_NOTRANS	Creates a Memory Device without transparency. The user must make sure that the background is drawn correctly. This way the Memory Device can be used for non-rectangular areas. An other advantage is the higher speed: Using this flag accelerates the Memory Device app. 30 - 50%.

Return value

Handle of the created Memory Device. If the routine fails the return value is 0.

GUI_MEMDEV_CreateFixed()

Description

Creates a Memory Device with dedicated color depth color conversion.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateFixed(int x0, int y0,
                                         int xSize, int ySize, int Flags,
                                         const tLCDDEV_APIList * pMemDevAPI,
                                         const LCD_API_COLOR_CONV * pColorConvAPI);
```

Parameter	Description
x0	X-position of Memory Device.
y0	Y-position of Memory Device.
xsize	X-size of Memory Device.
ysize	Y-size of Memory Device.
Flags	See table below.
pMemDevAPI	See table below.
pColorConvAPI	See table below.

Table 16.14: GUI_MEMDEV_CreateFixed() parameter list

Permitted values for parameter Flags	
GUI_MEMDEV_HASTRANS (recommended)	Default: The Memory Device is created with a transparency flag which ensures that the back-ground will be drawn correctly.
GUI_MEMDEV_NOTRANS	Creates a Memory Device without transparency. The user must make sure that the background is drawn correctly. This way the Memory Device can be used for non-rectangular areas. An other advantage is the higher speed: Using this flag accelerates the Memory Device app. 30 - 50%.

Parameter pMemDevAPI	
Defines the color depth of the Memory Device in bpp. The color depth of the Memory Device should be equal or greater than the required bits for the color conversion routines. A Memory Device with a 1bpp color conversion (GUI_COLOR_CONV_1) for example requires at least a Memory Device with 1bpp color depth. The available Memory Devices are 1bpp, 8bpp, 16bpp and 32bpp Memory Devices. So an 1bpp Memory Device should be used. If using a 4 bit per pixel color conversion (GUI_COLOR_CONV_4) at least 4bpp are needed for the Memory Device. In this case an 8bpp Memory Device should be used.	
Permitted values	
GUI_MEMDEV_APILIST_1	Create Memory Device with 1bpp color depth (1 byte per 8 pixels) Use if the specified color conversion requires 1bpp.
GUI_MEMDEV_APILIST_8	Create Memory Device with 8bpp color depth (1 byte per pixel) Use if the specified color conversion requires 8bpp or less.
GUI_MEMDEV_APILIST_16	Create Memory Device with 16bpp color depth (1 U16 per pixel) Use if the specified color conversion requires more than 8 bpp. (High color modes)
GUI_MEMDEV_APILIST_32	Create Memory Device with 32bpp color depth (1 U32 per pixel) Use if the specified color conversion requires more than 16 bpp. (True color modes)

Parameter pColorConvAPI	
This parameter defines the desired color conversion. For more details about the used bits per pixel and the color conversion, refer to the chapter “Colors” on page 291.	
Permitted values	
GUICC_1	Fixed palette mode 1. (black/white)
GUICC_2	Fixed palette mode 2. (4 gray scales)
GUICC_4	Fixed palette mode 4. (16 gray scales)
GUICC_565	Fixed palette mode 565.
GUICC_M565	Fixed palette mode M565.
GUICC_8666	Fixed palette mode 8666.
GUICC_888	Fixed palette mode 888.
GUICC_8888	Fixed palette mode 8888.

Return value

Handle to the created Memory Device. If the routine fails the return value is 0.

Additional information

This function can be used if a Memory Device with a specified color conversion should be created. This could make sense if for example some items should be printed on a printer device. The `Sample` folder contains the code example `MEMDEV_Printing.c` which shows how to use the function to print something in 1bpp color conversion mode.

Example

The following example shows how to create a Memory Device with 1bpp color depth:

```
GUI_MEMDEV_Handle hMem;
hMem = GUI_MEMDEV_CreateFixed(0, 0, 128, 128, 0,
                              GUI_MEMDEV_APILIST_1, // Used API list
                              GUI_COLOR_CONV_1);    // Black/white color conversion
GUI_MEMDEV_Select(hMem);
```


GUI_MEMDEV_CreateFixed32()

Description

Creates a Memory Device with a color depth of 32bpp and GUICC_8888 color conversion.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateFixed32(int x0,    int y0,
                                           int xSize, int ySize);
```

Parameter	Description
x0	X-position of Memory Device.
y0	Y-position of Memory Device.
xsize	X-size of Memory Device.
ysize	Y-size of Memory Device.

Table 16.15: GUI_MEMDEV_CreateFixed32() parameter list

Return value

Handle to the created Memory Device. If the routine fails the return value is 0.

Additional information

This function makes it more easy to create a 32 bpp memory device which is often required.

GUI_MEMDEV_Delete()

Description

Deletes a Memory Device.

Prototype

```
void GUI_MEMDEV_Delete(GUI_MEMDEV_Handle MemDev);
```

Parameter	Description
hMem	Handle to the Memory Device which has to be deleted.

Table 16.16: GUI_MEMDEV_Delete() parameter list

GUI_MEMDEV_DrawPerspectiveX()

Description

Draws the given Memory Device perspectively distorted into the currently selected device.

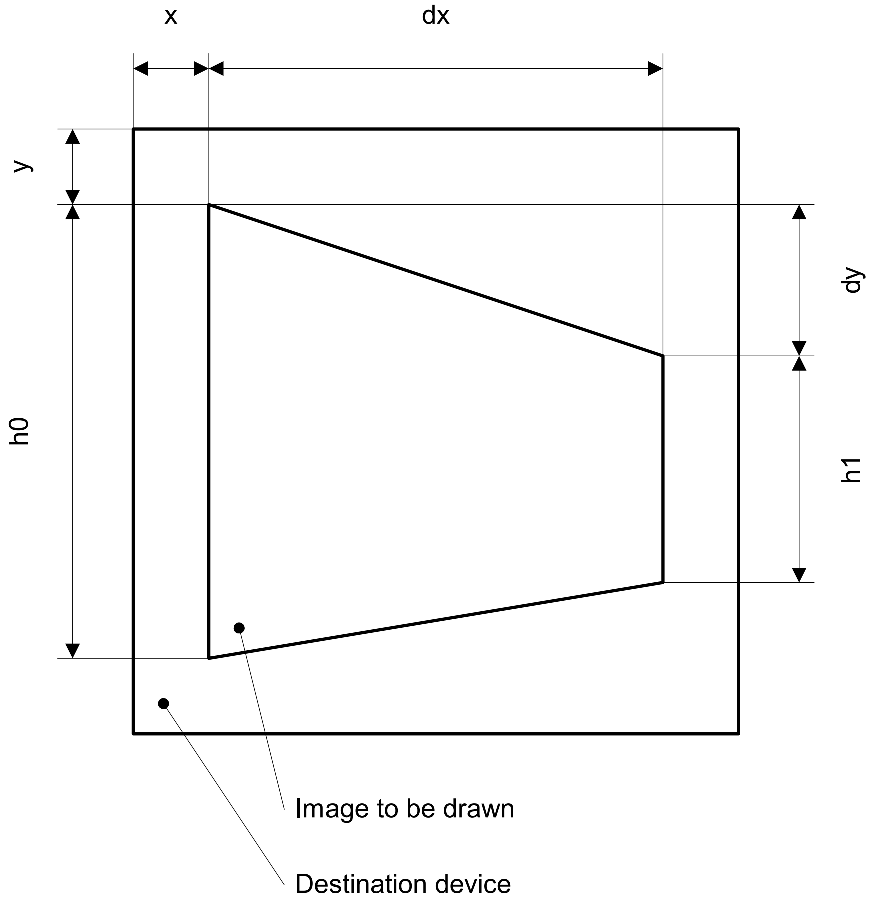
Prototype

```
void GUI_MEMDEV_DrawPerspectiveX(GUI_MEMDEV_Handle hMem, int x, int y,
                                 int h0, int h1, int dx, int dy);
```

Parameter	Description
hMem	Handle to source Memory Device with the image to be drawn.
x	Horizontal start position in pixels.
y	Vertical start position in pixels.
h0	Height of the leftmost edge of the image to be drawn.
h1	Height of the rightmost edge of the image to be drawn.
dx	Width of the image to be drawn.
dy	Position in y from the topmost pixel at the right relative to the topmost pixel at the left.

Table 16.17: GUI_MEMDEV_DrawPerspectiveX() parameter list

The picture below explains the parameters more detailed:



Additional information

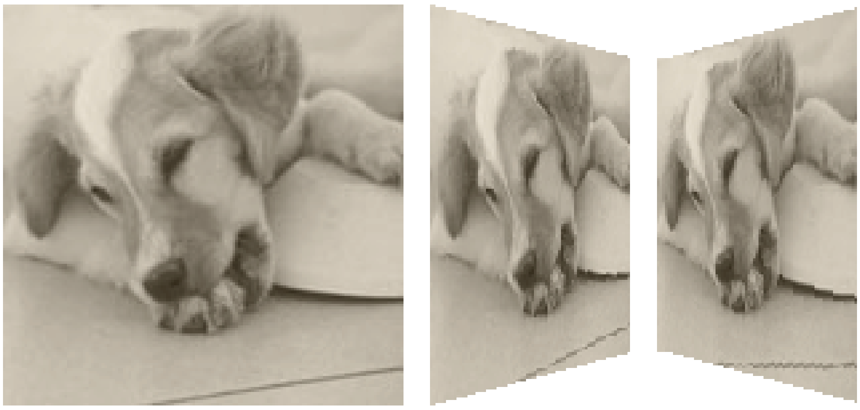
The function draws the contents of the given Memory Device into the currently selected device. The origin of the source device should be (0, 0). Size and distortion of the new image is defined by the parameters `dx`, `dy`, `h0` and `h1`.
Note that the function currently only works with Memory Devices with 32-bpp color depth and a system color depth of 32 bpp.

Example

The following example shows how to use the function:

```
GUI_MEMDEV_Handle hMem0, hMem1, hMem2;
hMem0 = GUI_MEMDEV_CreateFixed(0, 0, 150, 150, GUI_MEMDEV_NOTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
hMem1 = GUI_MEMDEV_CreateFixed(0, 0,  75, 150, GUI_MEMDEV_HASTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
hMem2 = GUI_MEMDEV_CreateFixed(0, 0,  75, 150, GUI_MEMDEV_HASTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
GUI_MEMDEV_Select(hMem0);
GUI_JPEG_Draw(_aJPEG, sizeof(_aJPEG), 0, 0);
GUI_MEMDEV_Select(hMem1);
GUI_MEMDEV_DrawPerspectiveX(hMem0, 0,  0, 150, 110, 75,  20);
GUI_MEMDEV_Select(hMem2);
GUI_MEMDEV_DrawPerspectiveX(hMem0, 0, 20, 110, 150, 75, -20);
GUI_MEMDEV_CopyToLCDAt(hMem0,   0, 10);
GUI_MEMDEV_CopyToLCDAt(hMem1, 160, 10);
GUI_MEMDEV_CopyToLCDAt(hMem2, 245, 10);
```


Screenshot of the above example



GUI_MEMDEV_GetDataPtr()

Description

Returns a pointer to the data area (image area) of a Memory Device. This data area can then be manipulated without the use of GUI functions; it can for example be used as output buffer for a JPEG or video decompression routine.

Prototype

```
void * GUI_MEMDEV_GetDataPtr(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.18: GUI_MEMDEV_GetDataPtr() parameter list

Additional information

The device data is stored from the returned address onwards. An application modifying this data has to take extreme caution that it does not overwrite memory outside of this data area.

Warning: Allocating dynamic memory could cause invalid data pointers!

It should be kept sure, that no memory is allocated during the pointer is used. Allocating memory could cause invalid pointers because memory cleaning operations.

Organization of the data area:

The pixels are stored in the mode "native" to the display (or layer) for which they are intended. For layers with 8 bpp or less, 8 bits (1 byte) are used per pixel; for layers with more than 8 and less or equal 16 bpp, a 16 bit value (U16) is used for one pixel. The memory is organized in reading order which means: First byte (or U16), stored at the start address, represents the color index of the pixel in the upper left corner (y=0, x=0); the next pixel, stored right after the first one, is the one to the left at (y=0, x=1). (Unless the Memory Device area is only 1 pixel wide). The next line is stored right after the first line in memory, without any kind of padding. Endian mode is irrelevant, it is assumed that 16 bit units are accessed as 16 bit units and not as 2 separate bytes. The data area is comprised of (xSize * ySize) pixels, so
xSize * ySize bytes for 8bpp or lower Memory Devices,
2 * xSize * ySize bytes (accessed as xSize * ySize units of 16 bits) for 16 bpp Memory Devices.

GUI_MEMDEV_GetXSize()

Description

Returns the X-size (width) of a Memory Device.

Prototype

```
int  GUI_MEMDEV_GetXSize(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.19: GUI_MEMDEV_GetXSize() parameter list

GUI_MEMDEV_GetYSize()

Description

Returns the Y-size (height) of a Memory Device in pixels.

Prototype

```
int  GUI_MEMDEV_GetYSize(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.20: GUI_MEMDEV_GetYSize() parameter list

GUI_MEMDEV_MarkDirty()

Description

Marks a rectangle area as dirty.

Prototype

```
void GUI_MEMDEV_MarkDirty(GUI_MEMDEV_Handle hMem,
                          int x0, int y0, int x1, int y1);
```

Parameter	Description
hMem	Handle to the Memory Device.
x0	x-coordinate of the upper left corner.
y0	y-coordinate of the upper left corner.
x1	x-coordinate of the lower right corner.
y1	y-coordinate of the lower right corner.

Table 16.21: GUI_MEMDEV_MarkDirty() parameter list

GUI_MEMDEV_PunchOutDevice()

Description

Punches out a shape of a Memory Device defined by a 8bpp mask Memory Device. The mask device must consist of 8bpp index values which define the intensity of the pixels to be used:

- Intensity 0 means 100% transparent.
- Intensity 255 means 100% opaque.

Intensity values between 0 and 255 mean semi transparency. The behavior of the function depends on the draw mode:

- GUI_DM_TRANS - Pixel becomes semi transparent
- other value - Pixel is mixed with the current background color

The punching operation will be done in the given device hMemData.

Prototype

```
int GUI_MEMDEV_PunchOutDevice(GUI_MEMDEV_Handle hMemData,
```



```
GUI_MEMDEV_Handle hMemMask);
```

Parameter	Description
hMemData	Memory Device which should be punched out. Must be a 32 bit Memory Device.
hMemMask	Handle to the Memory Device mask, which contains the intensity values.

Table 16.22: GUI_MEMDEV_PunchOutDevice() parameter list

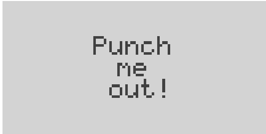
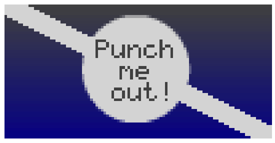
Return value

0 on success. 1 on error.

Example

```
#include "GUI.h"

/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  GUI_MEMDEV_Handle hMemData;
  GUI_MEMDEV_Handle hMemMask;
  GUI_RECT          Rect;

  GUI_Init();
  //
  // Background
  //
  GUI_SetBkColor(GUI_DARKBLUE);
  GUI_Clear();
  GUI_DrawGradientV(0, 0, 99, 49, GUI_DARKGRAY, GUI_DARKBLUE);
  GUI_SetColor(GUI_WHITE);
  //
  // Mask device
  //
  hMemMask = GUI_MEMDEV_CreateFixed(0, 0, 99, 49, GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_8,  GUICC_8);
  GUI_SetDrawMode(GUI_DM_TRANS);
  GUI_MEMDEV_Select(hMemMask);
  GUI_SetBkColor(GUI_BLACK);
  GUI_Clear();
  GUI_AA_FillCircle(49, 24, 20);
  GUI_SetPenSize(8);
  GUI_DrawLine(0, 0, 99, 49);
  //
  // Data Device
  //
  hMemData = GUI_MEMDEV_CreateFixed(0, 0, 99, 49, GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUICC_8888);
  GUI_MEMDEV_Select(hMemData);
  GUI_SetBkColor(GUI_LIGHTGRAY);
  GUI_Clear();
  Rect.x0 = 6;
  Rect.y0 = 0;
  Rect.x1 = 99;
  Rect.y1 = 49;
  GUI_SetColor(GUI_DARKGRAY);
  GUI_DispStringInRectEx("Punch\r\nme\r\nout!", &Rect,
                         GUI_TA_HCENTER | GUI_TA_VCENTER, 20, GUI_ROTATE_0);
  //
  // Result
  //
  GUI_MEMDEV_Select(0);
  GUI_MEMDEV_PunchOutDevice(hMemData, hMemMask);
  GUI_MEMDEV_Write(hMemData);
  while (1) {
    GUI_Delay(100);
  }
}
```


Screenshots

Background	Data	Mask	Result
			

GUI_MEMDEV_ReduceYSize()

Description

Reduces the Y-size of a Memory Device.

Prototype

```
void GUI_MEMDEV_ReduceYSize(GUI_MEMDEV_Handle hMem, int YSize);
```

Parameter	Description
hMem	Handle to Memory Device.
YSize	New Y-size of the Memory Device.

Table 16.23: GUI_MEMDEV_ReduceYSize() parameter list

Additional information

Changing the size of the Memory Device is more efficient than deleting and then recreating it.

GUI_MEMDEV_Rotate()

GUI_MEMDEV_RotateAlpha()

GUI_MEMDEV_RotateHQ()

GUI_MEMDEV_RotateHQAlpha()

GUI_MEMDEV_RotateHQHR()

GUI_MEMDEV_RotateHQT()

GUI_MEMDEV_RotateHR()

General Description

The functions rotate and scale the given source Memory Device. The source device will be rotated and scaled around its center and then shifted by the given amount of pixels. The result is saved into the given destination Memory Device. All these functions have similar postfixes containing the sequences 'HQ', 'HQT', 'HR' and 'Alpha' which are explained in the following:

Description 'HQ'

HQ stands for "High Quality". The functions which are named HQ use a more complex algorithm for calculating the destination pixel data. The HQ-algorithm can be used to achieve accurate results. The functions without the HQ addition use the 'nearest neighbor' method which is fast, but less accurate.

Description 'HQT'

HQT stands for "High Quality Transparency". The HQT algorithm improves the performance when rotating Memory Devices containing completely transparent pixels. The more completely transparent pixels the Memory Device contains, the more significant the performance boost gets. This function is still a HQ function and therefor produces results of the same accuracy.

Description 'HR'

HR stands for "High Resolution". The functions named HR use a precision of 8 subpixels. This makes it possible to display a Memory Device much more precisely on the screen.

Description 'Alpha'

The 'Alpha' functions allow to use an alpha value for blending in the source device into the destination device. A value between 0 and 255 can be used, where 0 means completely visible and 255 completely transparent. Of course alpha values of the source device will be considered.

Prototypes

```
void GUI_MEMDEV_Rotate    (GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                            int dx, int dy, int a, int Mag);

void GUI_MEMDEV_RotateHQ  (GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                            int dx, int dy, int a, int Mag);

void GUI_MEMDEV_RotateHQT (GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                            int dx, int dy, int a, int Mag);
```

Parameter	Description
hSrc	Handle of Memory Device to be rotated and scaled.
hDst	Handle of destination device.
dx	Distance in pixels for shifting the image in X.
dy	Distance in pixels for shifting the image in Y.
a	Angle to be used for rotation in degrees * 1000.
Mag	Magnification factor * 1000.

Table 16.24: GUI_MEMDEV_Rotate() / GUI_MEMDEV_RotateHQ() / GUI_MEMDEV_RotateHQT() parameter list

Prototypes Alpha

```
void GUI_MEMDEV_RotateAlpha  (GUI_MEMDEV_Handle hSrc,
                               GUI_MEMDEV_Handle hDst,
                               int dx, int dy, int a, int Mag, U8 Alpha);

void GUI_MEMDEV_RotateHQAlpha(GUI_MEMDEV_Handle hSrc,
                               GUI_MEMDEV_Handle hDst,
                               int dx, int dy, int a, int Mag, U8 Alpha);
```

Parameter	Description
hSrc	Handle of Memory Device to be rotated and scaled.
hDst	Handle of destination device.
dx	Distance in pixels for shifting the image in X.
dy	Distance in pixels for shifting the image in Y.
a	Angle to be used for rotation in degrees * 1000.
Mag	Magnification factor * 1000.
Alpha	Alpha value to be used for blending in the device.

Table 16.25: GUI_MEMDEV_RotateAlpha() / GUI_MEMDEV_RotateHQAlpha() parameter list

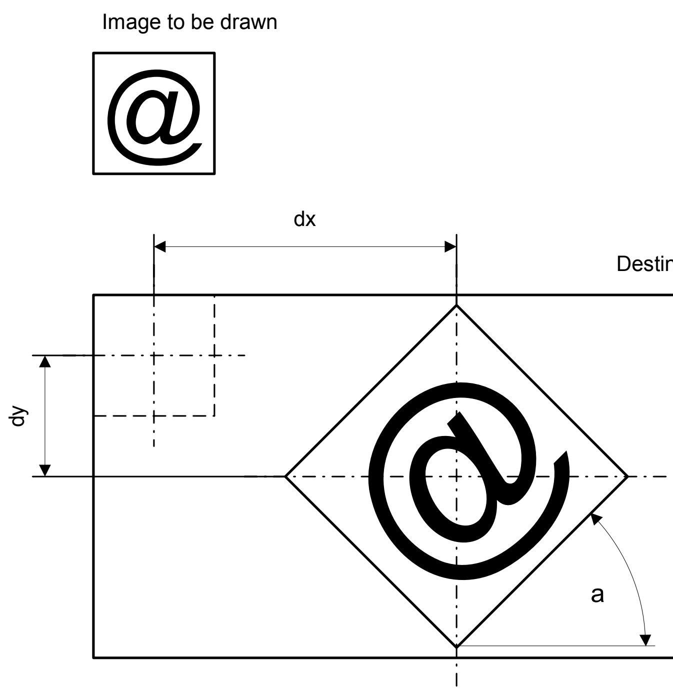
Prototypes HR

```
void GUI_MEMDEV_RotateHQHR(GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                           I32 dx, I32 dy, int a, int Mag);

void GUI_MEMDEV_RotateHR  (GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                           I32 dx, I32 dy, int a, int Mag);
```

Parameter	Description
hSrc	Handle of Memory Device to be rotated and scaled.
hDst	Handle of destination device.
dx	High resolution distance in pixels for shifting the image in X.
dy	High resolution distance in pixels for shifting the image in Y.
a	Angle to be used for rotation in degrees * 1000.
Mag	Magnification factor * 1000.

Table 16.26: GUI_MEMDEV_RotateHQHR() / GUI_MEMDEV_RotateHR() parameter list

The following picture gives a more detailed impression of the parameters:



Additional information

Both Memory Devices, source and destination, need to be created using a color depth of 32bpp. Further `GUI_MEMDEV_NOTRANS` should be used as `Flags` parameter when creating the devices.
If it is intended to preserve transparency, the according areas in both Memory Devices need to be filled with transparency before calling a rotate function.
The `Sample` folder contains the `MEMDEV_ZoomAndRotate.c` application which shows in detail how the function can be used.

Performance advantage of GUI_MEMDEV_RotateHQT()

The following table shows an approximation of the performance in comparison to `GUI_MEMDEV_RotateHQ()` in dependence of the percentage of transparent pixels:

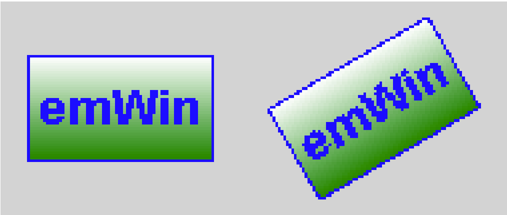
Percentage of transparent pixels	Performance advantage
0%	- 3%
10%	0%
50%	+21%
90%	+74%

Table 16.27: Performance advantages of GUI_MEMDEV_RotateHQT()

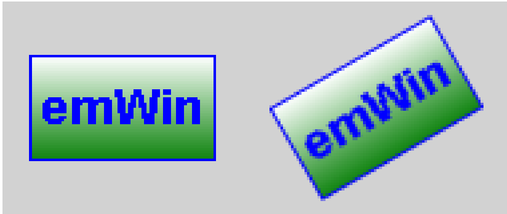
Example

```
GUI_MEMDEV_Handle hMemSource;
GUI_MEMDEV_Handle hMemDest;
GUI_RECT RectSource = {0, 0, 69, 39};
GUI_RECT RectDest   = {0, 0, 79, 79};
hMemSource = GUI_MEMDEV_CreateFixed(RectSource.x0, RectSource.y0,
                                    RectSource.x1 - RectSource.x0 + 1,
                                    RectSource.y1 - RectSource.y0 + 1,
                                    GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUI_COLOR_CONV_888);
hMemDest   = GUI_MEMDEV_CreateFixed(RectDest.x0,   RectDest.y0,
                                    RectDest.x1   - RectDest.x0   + 1,
                                    RectDest.y1   - RectDest.y0   + 1,
                                    GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUI_COLOR_CONV_888);
GUI_MEMDEV_Select(hMemSource);
GUI_DrawGradientV(RectSource.x0, RectSource.y0,
                  RectSource.x1, RectSource.y1,
                  GUI_WHITE, GUI_DARKGREEN);
GUI_SetColor(GUI_BLUE);
GUI_SetFont(&GUI_Font20B_ASCII);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringInRect("emWin", &RectSource, GUI_TA_HCENTER | GUI_TA_VCENTER);
GUI_DrawRect(0, 0, RectSource.x1, RectSource.y1);
GUI_MEMDEV_Select(hMemDest);
GUI_Clear();
GUI_MEMDEV_Select(0);
GUI_MEMDEV_RotateHQ(hMemSource, hMemDest,
                    (RectDest.x1 - RectSource.x1) / 2,
                    (RectDest.y1 - RectSource.y1) / 2,
                    30 * 1000,
                    1000);
GUI_MEMDEV_CopyToLCDAt(hMemSource,  10, (RectDest.y1 - RectSource.y1) / 2);
GUI_MEMDEV_CopyToLCDAt(hMemDest,   100, 0);
```

Screenshot of the above example using GUI_MEMDEV_Rotate()

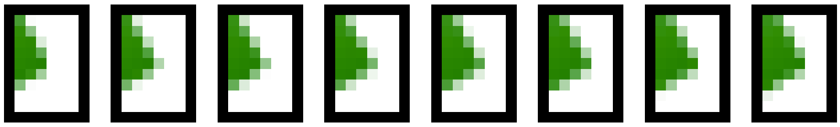


Screenshot of the above example using GUI_MEMDEV_RotateHQ()



Screenshot of the above example using GUI_MEMDEV_RotateHQHR()

This screenshot shows the 8 steps to move an antialiased corner one pixel to the right using subpixels.



GUI_MEMDEV_Select()

Description

Activates a Memory Device (or activates LCD if handle is 0).

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_Select(GUI_MEMDEV_Handle hMemDev);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.28: GUI_MEMDEV_Select() parameter list

Return value

Previously selected device. 0, if the display was selected.

GUI_MEMDEV_SerializeBMP()

Description

Creates a BMP file from the given Memory Device.

Prototype

```
void GUI_MEMDEV_SerializeBMP(GUI_MEMDEV_Handle          hDev,
                             GUI_CALLBACK_VOID_U8_P * pfSerialize,
                             void                   * p);
```

Parameter	Description
hDev	Handle to Memory Device.
pfSerialize	Pointer to a user defined serialization function. See prototype below.
p	Pointer to user defined data passed to the serialization function.

Table 16.29: GUI_MEMDEV_SerializeBMP() parameter list

Prototype of GUI_CALLBACK_VOID_U8_P

```
void GUI_CALLBACK_VOID_U8_P(U8 Data, void * p);
```

Additional information

To create a BMP file the color depth of the given Memory Device is used. In case it is 32bpp the resulting BMP file will consist of valid alpha data which is recognized by the Bitmap Converter.

An example for serialization can be found in the description of "GUI_BMP_Serialize()" on page 166.

GUI_MEMDEV_SetOrg()

Description

Changes the origin of the Memory Device on the LCD.

Prototype

```
void GUI_MEMDEV_SetOrg(GUI_MEMDEV_Handle hMem, int x0, int y0);
```

Parameter	Description
hMem	Handle to Memory Device.
x0	Horizontal position (of the upper left pixel).
y0	Vertical position (of the upper left pixel).

Table 16.30: GUI_MEMDEV_SetOrg() parameter list

Additional information

This routine can be helpful when the same device is used for different areas of the screen or when the contents of the Memory Device are to be copied into different areas.
Changing the origin of the Memory Device is more efficient than deleting and then recreating it.

GUI_MEMDEV_Write()

Description

Writes the content of the given Memory Device into the currently selected device.

Prototype

```
void GUI_MEMDEV_Write(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.31: GUI_MEMDEV_Write() parameter list

Additional information

In case of writing a 32 bpp memory device with alpha channel the alpha values will be considered for mixing the content of the given memory device with the content of the currently selected device.

GUI_MEMDEV_WriteAlpha()

Description

Writes the content of the given Memory Device into the currently selected device using alpha blending.

Prototype

```
void GUI_MEMDEV_WriteAlpha(GUI_MEMDEV_Handle hMem, int Alpha);
```

Parameter	Description
hMem	Handle to Memory Device.
Alpha	Alpha blending factor, 0 - 255

Table 16.32: GUI_MEMDEV_WriteAlpha() parameter list

Additional information

Alpha blending means mixing 2 colors with a given intensity. This function makes it possible to write semi-transparent from one Memory Device into an other Memory Device. The Alpha-parameter specifies the intensity used when writing to the currently selected device.
In case of writing a 32 bpp memory device with alpha channel the alpha values will also be considered.

GUI_MEMDEV_WriteAlphaAt()

Description

Writes the content of the given Memory Device into the currently selected device at the specified position using alpha blending.

Prototype

```
void GUI_MEMDEV_WriteAlphaAt(GUI_MEMDEV_Handle hMem,
                             int Alpha, int x, int y);
```

Parameter	Description
hMem	Handle to Memory Device.
Alpha	Alpha blending factor, 0 - 255
x	Position in X
y	Position in Y

Table 16.33: GUI_MEMDEV_WriteAlphaAt() parameter list

Additional information

(See GUI_MEMDEV_WriteAlpha())

GUI_MEMDEV_WriteAt()

Description

Writes the content of the given Memory Device into the currently selected device at the specified position.

Prototype

```
void GUI_MEMDEV_WriteAt(GUI_MEMDEV_Handle hMem, int x, int y);
```

Parameter	Description
hMem	Handle to Memory Device.
x	Position in X
y	Position in Y

Table 16.34: GUI_MEMDEV_WriteAt() parameter list

Additional information

(See GUI_MEMDEV_Write())

GUI_MEMDEV_WriteEx()

Description

Writes the content of the given Memory Device into the currently selected device at position (0, 0) using alpha blending and scaling.

Prototype

```
void GUI_MEMDEV_WriteEx(GUI_MEMDEV_Handle hMem,
```

```
int xMag, int yMag, int Alpha);
```

Parameter	Description
hMem	Handle to Memory Device.
xMag	Scaling factor for X-axis * 1000.
yMag	Scaling factor for Y-axis * 1000.
Alpha	Alpha blending factor, 0 - 255.

Table 16.35: GUI_MEMDEV_WriteEx() parameter list

Additional information

A negative scaling factor mirrors the output. Also Refer to "GUI_MEMDEV_WriteExAt()" below.

GUI_MEMDEV_WriteExAt()

Description

Writes the content of the given Memory Device into the currently selected device at the specified position using alpha blending and scaling.

Prototype

```
void GUI_MEMDEV_WriteExAt(GUI_MEMDEV_Handle hMem,
                         int x, int y, int xMag, int yMag, int Alpha);
```

Parameter	Description
hMem	Handle to Memory Device.
x	Position in X.
y	Position in Y.
xMag	Scaling factor for X-axis * 1000.
yMag	Scaling factor for Y-axis * 1000.
Alpha	Alpha blending factor, 0 - 255.

Table 16.36: GUI_MEMDEV_WriteExAt() parameter list

Additional information

A negative scaling factor mirrors the output.

Example

The following example creates 2 Memory Devices: hMem0 (40x10) and hMem1 (80x20). A small white text is drawn at the upper left position of hMem0 and hMem1. Then the function GUI_MEMDEV_WriteEx() writes the content of hMem0 to hMem1 using mirroring and magnifying:

```
GUI_MEMDEV_Handle hMem0, hMem1;

GUI_Init();
hMem0 = GUI_MEMDEV_Create(0, 0, 40, 10);
hMem1 = GUI_MEMDEV_Create(0, 0, 80, 20);
GUI_MEMDEV_Select(hMem0);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispString("Text");
GUI_MEMDEV_Select(hMem1);
GUI_SetBkColor(GUI_RED);
GUI_Clear();
GUI_DispStringAt("Text", 0, 0);
GUI_MEMDEV_WriteExAt(hMem0, 0, 0, -2000, -2000, 160);
GUI_MEMDEV_CopyToLCD(hMem1);
```


Screenshot of the above example

GUI_MEMDEV_WriteOpaque()

Description

Writes the content of the given Memory Device into the currently selected device.

Prototype

```
void GUI_MEMDEV_WriteOpaque(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.37: GUI_MEMDEV_WriteOpaque() parameter list

Additional information

In case of writing a 32 bpp memory device with alpha channel the alpha values will not be considered.

GUI_MEMDEV_WriteOpaqueAt()

Description

Writes the content of the given Memory Device into the currently selected device at the specified position.

Prototype

```
void GUI_MEMDEV_Write(GUI_MEMDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to Memory Device.

Table 16.38: GUI_MEMDEV_WriteOpaqueAt() parameter list

Additional information

In case of writing a 32 bpp memory device with alpha channel the alpha values will not be considered.

GUI_SelectLCD()

Description

Selects the LCD as target for drawing operations.

Prototype

```
void GUI_SelectLCD(void))
```

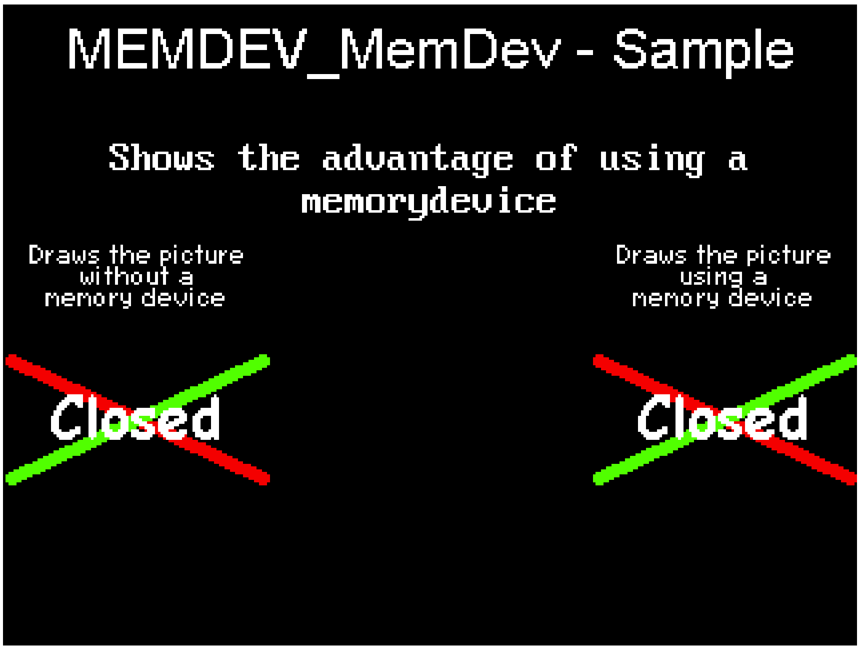
Example for using a Memory Device

The `Sample` folder contains the following example which shows how Memory Devices can be used:

- `MEMDEV_MemDev.c`

This example demonstrates the use of a Memory Device. Some items are written to a Memory Device and then copied to the display. Note that several other examples also make use of Memory Devices and may also be helpful to get familiar with them.

Screenshot of the above example:



16.11.2 Banding Memory Device

A Memory Device is first filled by executing the specified drawing functions. After filling the device, the contents are drawn to the LCD. There may be note enough memory available to store the complete output area at once, depending on your configuration. A banding Memory Device divides the drawing area into bands, in which each band covers as many lines as possible with the currently available memory.

GUI_MEMDEV_Draw()

Description

Drawing function to avoid flickering.

Prototype

```
int GUI_MEMDEV_Draw(GUI_RECT * pRect, GUI_CALLBACK_VOID_P * pfDraw,
                    void     * pData, int                   NumLines,
                    int        Flags);
```

Parameter	Description
pRect	Pointer to a GUI_RECT structure for the used LCD area.
pfDraw	Pointer to a callback function for executing the drawing.
pData	Pointer to a data structure used as parameter for the callback function.
NumLines	0 (recommended) or number of lines for the Memory Device.
Flags	See table below.

Table 16.39: GUI_MEMDEV_Draw() parameter list

Permitted values for parameter Flags	
GUI_MEMDEV_HASTRANS	Default: The Memory Device is created with a transparency flag which ensures that the background will be drawn correctly.
GUI_MEMDEV_NOTRANS (recommended)	Creates a Memory Device without transparency. The user must make sure that the background is drawn correctly. Should be used for optimization purposes only.

Return value

0 if successful, 1 if the routine fails.

Additional information

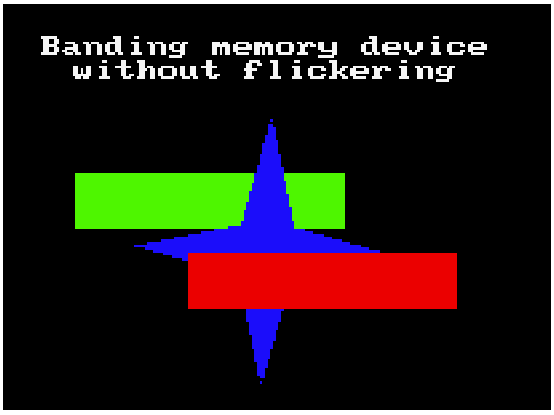
If the parameter `NumLines` is 0, the number of lines in each band is calculated automatically by the function. The function then iterates over the output area band by band by moving the origin of the Memory Device.

Example for using a banding Memory Device

The `Sample` folder contains the following example which shows how the function can be used:

- `MEMDEV_Banding.c`

Screenshot of above example



16.11.3 Auto device object

Memory Devices are useful when the display must be updated to reflect the movement or changing of items, since it is important in such applications to prevent the LCD from flickering. An auto device object is based on the banding Memory Device, and may be more efficient for applications such as moving indicators, in which only a small part of the display is updated at a time.
The device automatically distinguishes which areas of the display consist of fixed objects and which areas consist of moving or changing objects that must be updated. When the drawing function is called for the first time, all items are drawn. Each further call updates only the space used by the moving or changing objects. The actual drawing operation uses the banding Memory Device, but only within the necessary space. The main advantage of using an auto device object (versus direct usage of a banding Memory Device) is that it saves computation time, since it does not keep updating the entire display.

GUI_MEMDEV_CreateAuto()

Description

Creates an auto device object.

Prototype

```
int GUI_MEMDEV_CreateAuto(GUI_AUTODEV * pAutoDev);
```

Parameter	Description
`pAutoDev`	Pointer to a GUI_AUTODEV object.

Table 16.40: GUI_MEMDEV_CreateAuto() parameter list

Return value

Currently 0, reserved for later use.

GUI_MEMDEV_DeleteAuto()

Description

Deletes an auto device object.

Prototype

```
void GUI_MEMDEV_DeleteAuto(GUI_AUTODEV * pAutoDev);
```

Parameter	Description
pAutoDev	Pointer to a GUI_AUTODEV object.

Table 16.41: GUI_MEMDEV_DeleteAuto() parameter list

GUI_MEMDEV_DrawAuto()

Description

Executes a specified drawing routine using a banding Memory Device.

Prototype

```
int GUI_MEMDEV_DrawAuto(GUI_AUTODEV          * pAutoDev,
                        GUI_AUTODEV_INFO     * pAutoDevInfo,
                        GUI_CALLBACK_VOID_P * pfDraw,
                        void                 * pData);
```

Parameter	Description
pAutoDev	Pointer to a GUI_AUTODEV object.
pAutoDevInfo	Pointer to a GUI_AUTODEV_INFO object.
pfDraw	Pointer to the user-defined drawing function which is to be executed.
pData	Pointer to a data structure passed to the drawing function.

Table 16.42: GUI_MEMDEV_DrawAuto() parameter list

Return value

0 if successful, 1 if the routine fails.

Additional information

The `GUI_AUTODEV_INFO` structure contains the information about what items must be drawn by the user function:

```
typedef struct {
  char DrawFixed;
} GUI_AUTODEV_INFO;
```

`DrawFixed` is set to 1 if all items have to be drawn. It is set to 0 when only the moving or changing objects have to be drawn. We recommend the following procedure when using this feature:

```
typedef struct {
  GUI_AUTODEV_INFO AutoDevInfo; /* Information about what has to be drawn */
  /* Additional data used by the user function */
  ...
} PARAM;

static void Draw(void * p) {
  PARAM * pParam = (PARAM *)p;
  if (pParam->AutoDevInfo.DrawFixed) {
    /* Draw fixed background */
    ...
  }
  /* Draw moving objects */
  ...
  if (pParam->AutoDevInfo.DrawFixed) {
    /* Draw fixed foreground (if needed) */
    ...
  }
}
```

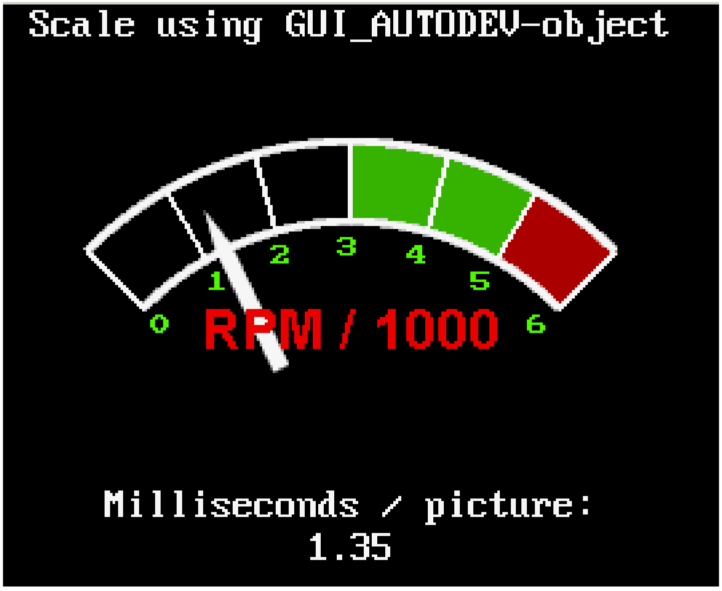


```
void main(void) {
  PARAM Param;                         /* Parameters for drawing routine */
  GUI_AUTODEV AutoDev;                 /* Object for banding Memory Device */
  /* Set/modify information for drawing routine */
  ...
  GUI_MEMDEV_CreateAuto(&AutoDev); /* Create GUI_AUTODEV-object */
  GUI_MEMDEV_DrawAuto(&AutoDev,      /* Use GUI_AUTODEV-object for drawing */
                      &Param.AutoDevInfo,
                      &Draw,
                      &Param);
  GUI_MEMDEV_DeleteAuto(&AutoDev); /* Delete GUI_AUTODEV-object */
}
```

Example for using an auto device object

The example MEMDEV_AutoDev.c demonstrates the use of an auto device object. It can be found as MEMDEV_AutoDev.c. A scale with a moving needle is drawn in the background and a small text is written in the foreground. The needle is drawn with the antialiasing feature of emWin. High-resolution antialiasing is used here to improve the appearance of the moving needle. For more information, see the chapter “Antialiasing” on page 1019.

Screenshot of above example



16.11.4 Measurement device object

Measurement devices are useful when you need to know the area used to draw something. Creating and selecting a measurement device as target for drawing operations makes it possible to retrieve the rectangle used for drawing operations.

GUI_MEASDEV_ClearRect()

Description

Call this function to clear the measurement rectangle of the given measurement device.

Prototype

```
void GUI_MEASDEV_ClearRect(GUI_MEASDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to measurement device.

Table 16.43: GUI_MEASDEV_ClearRect() parameter list

GUI_MEASDEV_Create()

Description
Creates a measurement device.

Prototype
```
GUI_MEASDEV_Handle GUI_MEASDEV_Create(void);
```

Return value
The handle of the measurement device.

GUI_MEASDEV_Delete()

Description
Deletes a measurement device.

Prototype
```
void GUI_MEASDEV_Delete (GUI_MEASDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to measurement device.

Table 16.44: GUI_MEASDEV_Delete() parameter list

GUI_MEASDEV_GetRect()

Description
Retrieves the result of the drawing operations.

Prototype
```
void GUI_MEASDEV_GetRect(GUI_MEASDEV_Handle hMem, GUI_RECT *pRect);
```

Parameter	Description
hMem	Handle to measurement device.
pRect	Pointer to GUI_RECT-structure to store result.

Table 16.45: GUI_MEASDEV_GetRect() parameter list

GUI_MEASDEV_Select()

Description
Selects a measurement device as target for drawing operations.

Prototype
```
void GUI_MEASDEV_Select (GUI_MEASDEV_Handle hMem);
```

Parameter	Description
hMem	Handle to measurement device.

Table 16.46: GUI_MEASDEV_Select() parameter list

Example
The following example shows the use of a measurement device. It creates a measurement device, draws a line and displays the result of the measurement device:

```
void MainTask(void) {
  GUI_MEASDEV_Handle hMeasdev;
  GUI_RECT Rect;
  GUI_Init();
  hMeasdev = GUI_MEASDEV_Create();
  GUI_MEASDEV_Select(hMeasdev);
  GUI_DrawLine(10, 20, 30, 40);
  GUI_SelectLCD();
```



```
    GUI_MEASDEV_GetRect(hMeasdev, &Rect);
    GUI_MEASDEV_Delete(hMeasdev);
    GUI_DispString("X0:");
    GUI_DispDec(Rect.x0, 3);
    GUI_DispString(" Y0:");
    GUI_DispDec(Rect.y0, 3);
    GUI_DispString(" X1:");
    GUI_DispDec(Rect.x1, 3);
    GUI_DispString(" Y1:");
    GUI_DispDec(Rect.y1, 3);
}
```

Screenshot of the above example:

X0:010 Y0:020 X1:030 Y1:040

16.11.5 Animation functions

Animations can be used to inject some life into the application. They will always help to let the user's eye smoothly capture what happens. All animation functions require 32-bit devices.

GUI_MEMDEV_FadeInDevices()

Description

Performs fading in one device over another Memory Device.

Prototype

```
int GUI_MEMDEV_FadeInDevices(GUI_MEMDEV_Handle hMem0,
                             GUI_MEMDEV_Handle hMem1,
                             int               Period);
```

Parameter	Description
hMem0	Handle of background Memory Device.
hMem1	Handle of Memory Device to be faded in.
Period	Time period in which the fading is processed.

Table 16.47: GUI_MEMDEV_FadeInDevices() parameter list

Return value

0 if successful, 1 if the function fails.

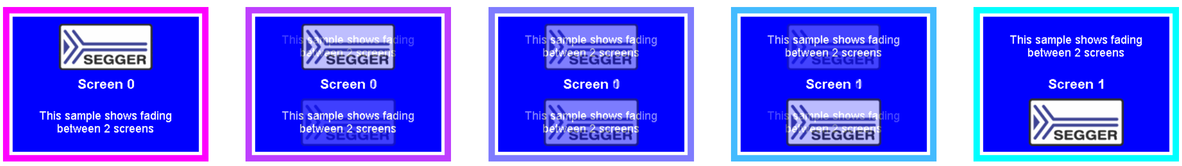
Additional Information

This function requires hMem0 and hMem1 to be of the same size and to be located at the same position on the screen.

Example

An example application using fading functions can be found in the file MEMDEV_FadingPerformance.c which is located in the folder "Sample\Tutorial".

Screenshots



GUI_MEMDEV_FadeOutDevices()

Description

Performs fading out one device overlaying another Memory Device.

Prototype

```
int GUI_MEMDEV_FadeOutDevices(GUI_MEMDEV_Handle hMem0,
                              GUI_MEMDEV_Handle hMem1,
                              int               Period);
```

Parameter	Description
hMem0	Handle of background Memory Device.
hMem1	Handle of Memory Device to be faded out.
Period	Time period in which the fading is processed.

Table 16.48: GUI_MEMDEV_FadeOutDevices() parameter list

Return value

0 if successful, 1 if the function fails.

Additional Information

This function requires hMem0 and hMem1 to be of the same size and to be located at the same position on the screen.

GUI_MEMDEV_SetAnimationCallback()

Description

Sets a user defined callback function to be called while animations are processed. The function should contain code to determine whether processing of the current animation shall go on or abort.

Prototype

```
void GUI_MEMDEV_SetAnimationCallback(
                                GUI_ANIMATION_CALLBACK_FUNC * pCbAnimation,
                                void                        * pVoid);
```

Parameter	Description
pCbAnimation	Pointer to the user defined callback function.
pVoid	Data pointer.

Table 16.49: GUI_MEMDEV_SetAnimationCallback() parameter list

Additional Information

The callback function is called every time an animation function has just copied the actual step to the screen.

Example

The following example shows the use of a GUI_ANIMATION_CALLBACK_FUNC, which gives the possibility to react on PID events:

```
static int _cbAnimation(int TimeRem, void * pVoid) {
  int Pressed;

  if (TimeRem /* Insert Condition */) {
    /* ... React on remaining Time ... */
  }
  Pressed = _GetButtonState();
  if (Pressed) {
    return 1;   // Button was     pressed, stop     animation
  } else {
    return 0;   // Button was not pressed, continue animation
  }
}

void main(void) {
  GUI_Init();
  GUI_MEMDEV_SetAnimationCallback(_cbAnimation, (void *)&_Pressed);
  while (1) {
    /* Do animations... */
  }
}
```


GUI_MEMDEV_SetTimePerFrame()

Description

Sets the minimum time used for one animation frame. If the process of drawing requires less time `GUI_X_Delay()` is called with the time difference.

Prototype

```
void GUI_MEMDEV_SetTimePerFrame(unsigned TimePerFrame);
```

Parameter	Description
`TimePerFrame`	Minimum time used for one animatin frame.

Table 16.50: GUI_MEMDEV_SetTimePerFrame() parameter list

16.11.6 Animation functions (Window Manager required)

The following animation functions require usage of the Window Manager.

GUI_MEMDEV_FadeInWindow()

GUI_MEMDEV_FadeOutWindow()

Description

Fades in/out a window by decreasing/increasing the alpha value

Prototype

```
int GUI_MEMDEV_FadeInWindow (WM_HWIN hWin, int Period);
int GUI_MEMDEV_FadeOutWindow(WM_HWIN hWin, int Period);
```

Parameter	Description
`hWin`	Handle to the window which has to be faded in/out
`Period`	Time period in which the fading is processed

Table 16.51: GUI_MEMDEV_FadeInWindow() / GUI_MEMDEV_FadeOutWindow() parameter list

Return value

0 if successful, 1 if the function fails.

Additional Information

After the window has been faded the desktop and its child windows are validated.

Example

An example application using the fading functions for windows can be found in the file `SKINNING_NestedModal.c` which is located in the folder `Sample\Tutorial`.

Screenshots

GUI_MEMDEV_MoveInWindow()

GUI_MEMDEV_MoveOutWindow()

Description

Moves a window into/out of the screen. First the window is drawn minimized/maximized at the specified position/its actual position and then moved to its actual position/the specified position while magnifying to its actual size/demagnifying. The window can be spun clockwise as well as counterclockwise while it is moving.

Prototype

```
int GUI_MEMDEV_MoveInWindow (WM_HWIN hWin, int x,    int y,
                                           int a180, int Period);
int GUI_MEMDEV_MoveOutWindow(WM_HWIN hWin, int x,    int y,
                                           int a180, int Period);
```

Parameter	Description
`hWin`	Handle to the window which has to be moved
`x`	Position in x from/to where the window is moved
`y`	Position in y from/to where the window is moved
`a180`	Count of degrees the window will be spun for: a180 = 0 -> no spinning a180 > 0 -> clockwise a180 < 0 -> counterclockwise
`Period`	Time period in which the moving is processed

Table 16.52: GUI_MEMDEV_MoveInWindow / GUI_MEMDEV_MoveOutWindow() parameter list

Return value

0 if successful, 1 if the function fails.

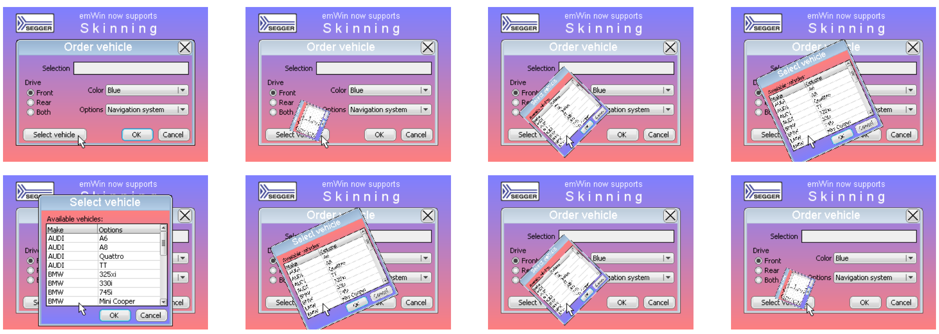
Additional Information

After the window has been moved the desktop and its child windows are validated. `GUI_MEMDEV_MoveInWindow()` / `GUI_MEMDEV_MoveOutWindow()` requires approximately 1 MB of dynamic memory to run properly in QVGA mode.

Example

An example application using the functions `GUI_MEMDEV_MoveInWindow()` and `GUI_MEMDEV_MoveOutWindow()` can be found in the file `SKINNING_NestedModal.c` which is located in the folder `Sample\Tutorial` folder.

Screenshots



GUI_MEMDEV_ShiftInWindow()

GUI_MEMDEV_ShiftOutWindow()

Description

Shifts a Window in a specified direction into/out of the screen to/from its actual position.

Prototype

```
int GUI_MEMDEV_ShiftInWindow (WM_HWIN hWin, int Period, int Direction);
int GUI_MEMDEV_ShiftOutWindow(WM_HWIN hWin, int Period, int Direction);
```

Parameter	Description
`hWin`	Handle to the window which has to be shifted.
`Period`	Time period in which the shifting is processed.
`Direction`	See permitted values for this parameter below.

Table 16.53: GUI_MEMDEV_ShiftInWindow / GUI_MEMDEV_ShiftOutWindow() parameter list

Permitted values for parameter `Direction`	
`GUI_MEMDEV_EDGE_LEFT`	Shift window to the left.
`GUI_MEMDEV_EDGE_RIGHT`	Shift window to the right.
`GUI_MEMDEV_EDGE_TOP`	Shift window to the top.
`GUI_MEMDEV_EDGE_BOTTOM`	Shift window to the bottom.

Return value

0 if successful, 1 if the function fails.

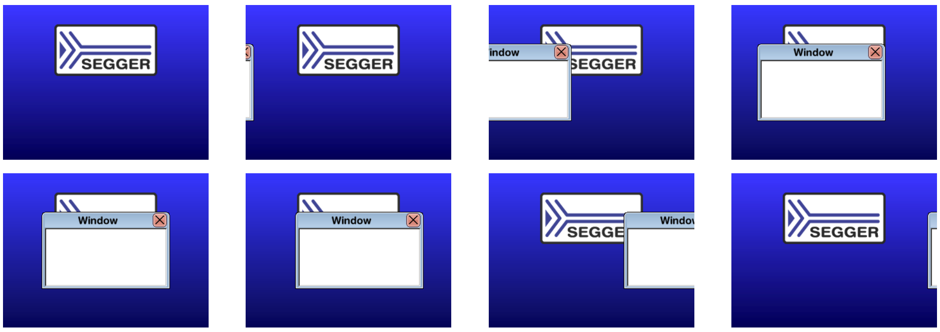
Additional Information

After the window has been shifted the desktop and its child windows are validated. `GUI_MEMDEV_ShiftInWindow()` and `GUI_MEMDEV_ShiftOutWindow()` require approximately 1 MB of dynamic memory to run properly in QVGA mode.

Example

An example application using the functions `GUI_MEMDEV_ShiftInWindow()` and `GUI_MEMDEV_ShiftOutWindow()` can be found in the file `SKINNING_Notepad.c` which is located in the folder `Sample\Tutorial` folder.

Screenshots



GUI_MEMDEV_SwapWindow()

Description

Swaps a window with the old content of the target area.

Prototype

```
int GUI_MEMDEV_SwapWindow(WM_HWIN hWin, int Period, int Edge);
```

Parameter	Description
hWin	Handle to the window which has to be shifted.
Period	Time period in which the shifting is processed.
Edge	See permitted values for this parameter below.

Table 16.54: GUI_MEMDEV_SwapWindow() parameter list

Permitted values for parameter Direction	
GUI_MEMDEV_EDGE_LEFT	Shift window to the left.
GUI_MEMDEV_EDGE_RIGHT	Shift window to the right.
GUI_MEMDEV_EDGE_TOP	Shift window to the top.
GUI_MEMDEV_EDGE_BOTTOM	Shift window to the bottom.

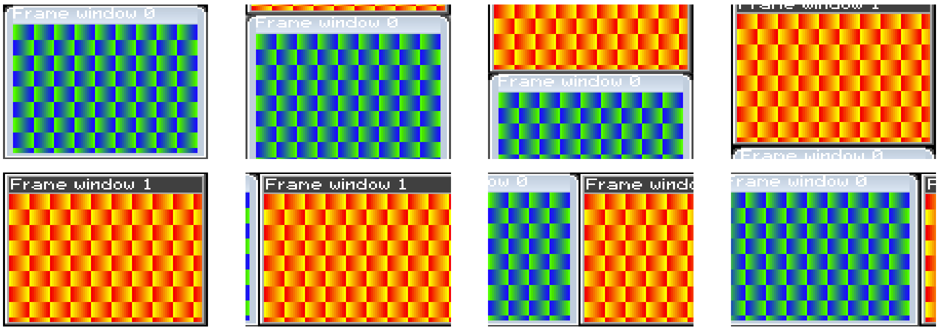
Return value

0 if successful, 1 if the function fails.

Additional Information

After the window has been swapped the desktop and its child windows are validated. GUI_MEMDEV_SwapWindow() requires approximately 1 MB of dynamic memory to run properly in QVGA mode.

Screenshots



16.11.7 Blending and Blurring functions

GUI_MEMDEV_BlendColor32()

Description

Blends a window with the given color and blending intensity.

Prototype

```
int GUI_MEMDEV_BlendColor32(GUI_MEMDEV_Handle hMem, U32 BlendColor,
                            U8                BlendIntens);
```

Parameter	Description
hMem	Handle to the Memory Device which has to be blended.
BlendColor	Color which is used for the blending effect.
BlendIntens	Intensity of the blending effect. Should be 0 (no blending) - 255 (full blending).

Table 16.55: GUI_MEMDEV_BlendColor32() parameter list

Return value

0 on success. 1 on error.

GUI_MEMDEV_CreateBlurredDevice32()

Description

Creates a blurred copy of the given Memory Device using the currently set blurring function.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateBlurredDevice32(GUI_MEMDEV_Handle hMem,
                                                    U8 Depth);
```

Parameter	Description
hMem	Handle to the Memory Device which has to be blurred.
Depth	Depth of the blurring effect. Should be specified with 1-10.

Table 16.56: GUI_MEMDEV_CreateBlurredDevice32() parameter list

Return value

Handle of the blurred Memory Device.

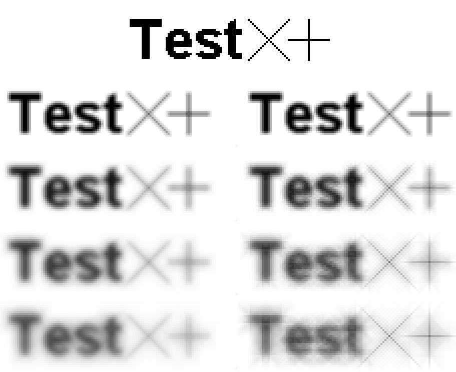
Additional information

The source Memory Device should consist of a color depth of 32 bpp. The resulting Memory Device will be of the same size at 32 bpp.
Information about memory usage and performance can be found in the descriptions of the ...HQ() and ...LQ()-function.
This function works according to the currently set blurring quality. In order to change the quality, the functions `GUI_MEMDEV_SetBlurHQ()` and `GUI_MEMDEV_SetBlurLQ()` can be used. The default quality is high.

Comparison

This screenshot shows the same elements without effect in the top row, blurred at high quality in the left column and blurred at low quality in the right column. The blurring depth was set as follows:

Column No.	Blurring depth
1st column	0
2nd column	1
3rd column	3
4th column	5
5th column	7



Performance

The performance is given relative to the time it takes to create a blurred device at high quality using a blurring depth of 1.

Blurring depth	High Quality	Low Quality
1	1	1.32
3	3.54	2.01
5	8.65	2.65
7	16.16	3.26

Table 16.57: Blurring performance

According to the values creating a blurred device at high quality using a blurring depth of 5 takes approximately half the time it would take to create a blurred device at high quality using a blurring depth of 7.

GUI_MEMDEV_CreateBlurredDevice32HQ()

Description

Creates a blurred copy of the given Memory Device at high quality.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateBlurredDevice32HQ(
                                                   GUI_MEMDEV_Handle hMem,
                                                   U8                Depth);
```

Parameter	Description
hMem	Handle to the Memory Device which has to be blurred.
Depth	Depth of the blurring effect. Should be specified with 1-10.

Table 16.58: GUI_MEMDEV_CreateBlurredDevice32HQ() parameter list

Return value

Handle of the blurred Memory Device. 0, if the function fails.

Additional information

The source Memory Device should consist of a color depth of 32 bpp. The resulting Memory Device will be of the same size at 32 bpp. This routine requires an addition of 16 bytes per pixel plus memory to allocate iterator arrays which are used to accelerate pixel addressing. The required memory for the iterator arrays depends on the blurring depth to perform. The number of bytes is calculated as follows:

```
Size = (1 + Depth * (Depth - 1) * 4) * (3 * sizeof(int) + 4)
```

A screenshot can be found under “Comparison” on page 357.

GUI_MEMDEV_CreateBlurredDevice32LQ()

Description

Creates a blurred copy of the given Memory Device at low quality.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateBlurredDevice32LQ(
                                                   GUI_MEMDEV_Handle hMem,
                                                   U8                Depth);
```

Parameter	Description
hMem	Handle to the Memory Device which has to be blurred.
Depth	Depth of the blurring effect. Should be specified with 1-10.

Table 16.59: GUI_MEMDEV_CreateBlurredDevice32LQ() parameter list

Return value

Handle of the blurred Memory Device. 0, if the function fails.

Additional information

The source Memory Device should consist of a color depth of 32 bpp. This is a creating function. The created Memory Device will be of the same size at 32 bpp. Beyond that no additional memory is required. A screenshot can be found under “Comparison” on page 357.

GUI_MEMDEV_Dither32()

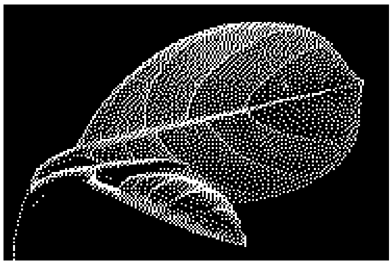
Before	After
	

Table 16.60: Memory device before and after dithering

Description

The function dithers the given memory device using the given fixed palette mode. Please note that the function does not reduce the color depth of the memory device. If dithered images with a reduced color depth (and less storage requirement) are desired, the bitmap converter should be used to dither the images.

Prototype

```
int GUI_MEMDEV_Dither32(GUI_MEMDEV_Handle hMem,
                        const LCD_API_COLOR_CONV * pColorConvAPI);
```

Parameter	Description
hMem	Handle to the Memory Device which has to be dithered.
pColorConvAPI	Depth of the blurring effect. Should be specified with 1-10.

Table 16.61: GUI_MEMDEV_CreateBlurredDevice32LQ() parameter list

Return value

0 on success, 1 on error.

Additional information

The function works only with memory devices having a color depth of 32bpp.

GUI_MEMDEV_SetBlurHQ()

Description

Sets the blurring quality to high.

Prototype

```
void GUI_MEMDEV_SetBlurHQ(void);
```

Additional information

Setting the blurring quality affects the function `GUI_MEMDEV_CreateBlurredDevice32()` which in turn is called by other functions. (e.g. `GUI_MEMDEV_BlurWinBk()`).

GUI_MEMDEV_SetBlurLQ()

Description

Sets the blurring quality to low.

Prototype

```
void GUI_MEMDEV_SetBlurLQ(void);
```

Additional information

Additional information are stated under "GUI_MEMDEV_SetBlurHQ()" on page 359.

16.11.8 Blurring and Blending functions (Window Manager required)

GUI_MEMDEV_BlendWinBk()

Description

Blends the background of a window within the given period from its initial state to the given blending intensity.

Prototype

```
int GUI_MEMDEV_BlendWinBk(WM_HWIN hWin, int Period, U32 BlendColor,
                          U8      BlendIntens);
```

Parameter	Description
hWin	Handle to the window which has to be blended.
Period	Effect Period.
BlendColor	Color which is used for the background to be blended with.
BlendIntens	Final intensity of the blending effect. Should be 0 (no blending) - 255 (full blending).

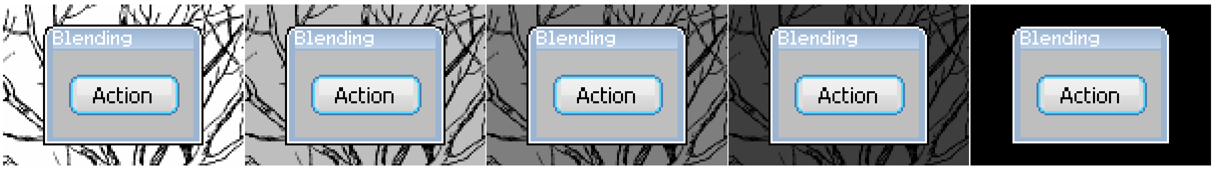
Table 16.62: GUI_MEMDEV_BlendWinBk() parameter list

Return value

0 on success. 1 on error.

Screenshots

The following screenshots show the background window being blended in 5 steps. The blending is performed using GUI_BLACK as BlendColor. BlendIntens is given with the highest possible value of 255.



GUI_MEMDEV_BlurAndBlendWinBk()

Description

Blurs and blends the background of a window within the given period from its initial state to the given blurring value and blending intensity.

Prototype

```
int GUI_MEMDEV_BlurAndBlendWinBk(WM_HWIN hWin, int Period, U8 BlurDepth,
                                 U32     BlendColor,        U8 BlendIntens);
```

Parameter	Description
hWin	Handle to the window which has to be blurred.
Period	Effect Period.
BlurDepth	Final depth of the blurring effect. Should be specified with 1-10.
BlendColor	Color which is used for the background to be blended with.
BlendIntens	Final intensity of the blending effect. Should be 0 (no blending) - 255 (full blending).

Table 16.63: GUI_MEMDEV_BlurAndBlendWinBk() parameter list

Return value

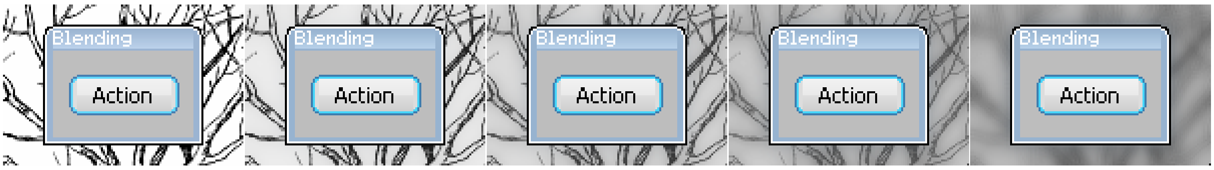
0 on success. 1 on error.

Additional information

The blurring quality can be changed using the functions GUI_MEMDEV_SetBlurHQ() or GUI_MEMDEV_SetBlurLQ().

Screenshots

The following screenshots show the background window being blurred and blended in 5 steps. The used values are 10 as blurring depth and 64 as blending intensity. The blending color is GUI_WHITE.



GUI_MEMDEV_BlurWinBk()

Description

Blurs the background of a window within the given period from its initial state to the given blurring value.

Prototype

```
int GUI_MEMDEV_BlurWinBk(WM_HWIN hWin, int Period, U8 BlurDepth);
```

Parameter	Description
hWin	Handle to the window which has to be blurred.
Period	Effect Period.
BlurDepth	Final depth of the blurring effect. Should be specified with 1-10.

Table 16.64: GUI_MEMDEV_BlurWinBk() parameter list

Return value

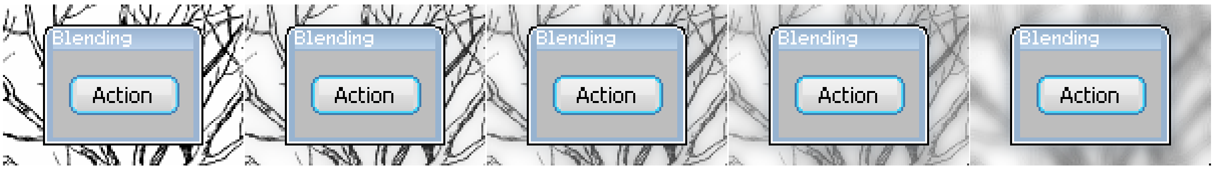
0 on success. 1 on error.

Additional information

The blurring quality can be changed using the functions GUI_MEMDEV_SetBlurHQ() or GUI_MEMDEV_SetBlurLQ().

Screenshots

The following screenshots show the background window being blurred in 5 steps. The used blurring depth is 10.



Chapter 17

Execution Model: Single Task / Multitask

emWin has been designed from the beginning to be compatible with different types of environments. It works in single task and in multitask applications, with a proprietary operating system or with any commercial RTOS such as embOS or uC/OS.

17.1 Supported execution models

We have to basically distinguish between 3 different execution models:

Single task system (superloop)

The entire program runs in one superloop. Normally, all software components are periodically called. Interrupts must be used for real time parts of the software since no real time kernel is used.

Multitask system: one task calling emWin

A real time kernel (RTOS) is used, but only one task calls emWin functions. From the graphic software’s point of view, it is the same as being used in a single task system.

Multitask system: multiple tasks calling emWin

A real time kernel (RTOS) is used, and multiple tasks call emWin functions. This works without a problem as long as the software is made thread-safe, which is done by enabling multitask support in the configuration and adapting the kernel interface routines. For popular kernels, the kernel interface routines are readily available.

17.2 Single task system (superloop)

17.2.1 Description

The entire program runs in one superloop. Normally, all components of the software are periodically called. No real time kernel is used, so interrupts must be used for real time parts of the software. This type of system is primarily used in smaller systems or if real time behavior is not critical.

17.2.2 Superloop example (without emWin)

```
void main (void) {
  HARDWARE_Init();

  /* Init software components */
  XXX_Init();
  YYY_Init();

  /* Superloop: call all software components regularily */
  while (1) {
    /* Exec all compontents of the software */
    XXX_Exec();
    YYY_Exec();
  }
}
```

17.2.3 Advantages

No real time kernel is used (-> smaller ROM size, just one stack -> less RAM for stacks), no preemption/synchronization problems.

17.2.4 Disadvantages

The superloop type of program can become hard to maintain if it exceeds a certain program size. Real time behavior is poor, since one software component cannot be interrupted by any other component (only by interrupts). This means that the reaction time of one software component depends on the execution time of all other components in the system.

17.2.5 Using emWin

There are no real restrictions regarding the use of emWin. As always, `GUI_Init()` has to be called before you can use the software. From there on, any API function can be used. If the Window Manager's callback mechanism is used, then an emWin update function has to be called regularly. This is typically done by calling the `GUI_Exec()` from within the superloop. Blocking functions such as `GUI_Delay()` and `GUI_ExecDialog()` should not be used in the loop since they would block the other software modules.
The default configuration, which does not support multitasking (`#define GUI_OS 0`) can be used; kernel interface routines are not required.

17.2.6 Superloop example (with emWin)

```
void main (void) {
  HARDWARE_Init();

  /* Init software components */
  XXX_Init();
  YYY_Init();
  GUI_Init();          /* Init emWin */

  /* Superloop: call all software components regularily */
  while (1) {
    /* Exec all compontents of the software */
    XXX_Exec();
    YYY_Exec();
    GUI_Exec();          /* Exec emWin for functionality like updating windows */
  }
}
```

17.3 Multitask system: one task calling emWin

17.3.1 Description

A real time kernel (RTOS) is used. The user program is split into different parts, which execute in different tasks and typically have different priorities. Normally the real time critical tasks (which require a certain reaction time) will have the highest priorities. **One single task** is used for the user interface, which calls emWin functions. This task usually has the lowest priority in the system or at least one of the lowest (some statistical tasks or simple idle processing may have even lower priorities).
Interrupts can, but do not have to be used for real time parts of the software.

17.3.2 Advantages

The real time behavior of the system is excellent. The real time behavior of a task is affected only by tasks running at higher priority. This means that changes to a program component running in a low priority task do not affect the real time behavior at all. If the user interface is executed from a low priority task, this means that changes to the user interface do not affect the real time behavior. This kind of system makes it easy to assign different components of the software to different members of the development team, which can work to a high degree independently from each other.

17.3.3 Disadvantages

You need to have a real time kernel (RTOS), which costs money and uses up ROM and RAM (for stacks). In addition, you will have to think about task synchronization and how to transfer information from one task to another.

17.3.4 Using emWin

If the Window Manager's callback mechanism is used, then an emWin update function (typically `GUI_Exec()`, `GUI_Delay()`) has to be called regularly from the task calling emWin. Since emWin is only called by one task, to emWin it is the same as being used in a single task system.
The default configuration, which does not support multitasking (`#define GUI_OS 0`) can be used; kernel interface routines are not required. You can use any real time kernel, commercial or proprietary.

17.4 Multitask system: multiple tasks calling emWin

17.4.1 Description

A real time kernel (RTOS) is used. The user program is split into different parts, which execute in different tasks with typically different priorities. Normally the real time critical tasks (which require a certain reaction time) will have the highest priorities. **Multiple tasks** are used for the user interface, calling emWin functions. These tasks typically have low priorities in the system, so they do not affect the real time behavior of the system.
Interrupts can, but do not have to be used for real time parts of the software.

17.4.2 Advantages

The real time behavior of the system is excellent. The real time behavior of a task is affected only by tasks running at higher priority. This means that changes of a program component running in a low priority task do not affect the real time behavior at all. If the user interface is executed from a low priority task, this means that changes on the user interface do not affect the real time behavior. This kind of system makes it easy to assign different components of the software to different members of the development team, which can work to a high degree independently from each other.

17.4.3 Disadvantages

You have to have a real time kernel (RTOS), which costs money and uses up some ROM and RAM (for stacks). In addition, you will have to think about task synchronization and how to transfer information from one task to another.

17.4.4 Using emWin

If the Window Manager's callback mechanism is used, then an emWin update function (typically `GUI_Exec()`, `GUI_Delay()`) has to be called regularly from one or more tasks calling emWin.
The default configuration, which does not support multitasking (`#define GUI_OS 0`) can **NOT** be used. The configuration needs to enable multitasking support and define a maximum number of tasks from which emWin is called (excerpt from `GUIConf.h`):

```
#define GUI_OS       1      // Enable multitasking support
#define GUI_MAXTASK  5      // Max. number of tasks that may call emWin
```

Kernel interface routines are required, and need to match the kernel being used. You can use any real time kernel, commercial or proprietary. Both the macros and the routines are discussed in the following chapter sections.

17.4.5 Recommendations

- Call the emWin update functions (that is, `GUI_Exec()`, `GUI_Delay()`) from just one task. It will help to keep the program structure clear. If you have sufficient RAM in your system, dedicate one task (with the lowest priority) to updating emWin. This task will continuously call `GUI_Exec()` as shown in the example below and will do nothing else.
- Keep your real time tasks (which determine the behavior of your system with respect to I/O, interface, network, etc.) separate from tasks that call emWin. This will help to assure best real time performance.
- If possible, use only one task for your user interface. This helps to keep the program structure simple and simplifies debugging. (However, this is not required and may not be suitable in some systems.)

17.4.6 Example

This excerpt shows the dedicated emWin update task. It is taken from the example `MT_Multitasking`, which is included in the examples shipped with emWin:

```
/*********************************************************************
*
*          GUI background processing
*
* This task does the background processing.
* The main job is to update invalid windows, but other things such as
* evaluating mouse or touch input may also be done.
*/
void GUI_Task(void) {
  while(1) {
    GUI_Exec();           /* Do the background work ... Update windows etc.) */
    GUI_X_ExecIdle();     /* Nothing left to do for the moment ... Idle processing */
  }
}
```

17.5 Configuration functions for multitasking support

The following table shows the configuration functions available for a multitask system with multiple tasks calling emWin:

Routine	Description
`GUI_SetSignalEventFunc()`	Sets a function that signals an event.
`GUI_SetWaitEventFunc()`	Sets a function that waits for an event.
`GUI_SetWaitEventTimedFunc()`	Sets a function that waits for an event for a given period of time.

Table 17.1: Configuration functions for multitasking support

GUI_SetSignalEventFunc()

Description

Sets a function that signals an event.

Prototype

```
void GUI_SetSignalEventFunc(GUI_SIGNAL_EVENT_FUNC pfSignalEvent);
```

Parameter	Description
`pfSignalEvent`	Pointer to a function that signals an event.

Table 17.2: GUI_SetSignalEventFunc() parameter list

Definition of GUI_SIGNAL_EVENT_FUNC

```
typedef void (* GUI_SIGNAL_EVENT_FUNC)(void);
```


Additional information

Per default the GUI needs to periodically check for events unless a function is defined which waits and one that triggers an event. This function sets the function which triggers an event. It makes only sense in combination with `GUI_SetWaitEventFunc()`. and `GUI_SetWaitEventTimedFunc()`. The advantage of using these functions instead of polling is the reduction of CPU load of the waiting task to 0% while it waits for input. If the function has been specified as recommended and the user gives the system any input (keyboard or pointer input device) the specified function should signal an event.
It is recommended to specify the function `GUI_X_SignalEvent()` for the job.

Example

```
GUI_SetSignalEventFunc(GUI_X_SignalEvent);
```

GUI_SetWaitEventFunc()

Description

Sets a function which waits for an event.

Prototype

```
void GUI_SetWaitEventFunc(GUI_WAIT_EVENT_FUNC pfWaitEvent);
```

Parameter	Description
pfWaitEvent	Pointer to a function that waits for an event.

Table 17.3: GUI_SetWaitEventFunc() parameter list

Definition of GUI_SIGNAL_EVENT_FUNC

```
typedef void (* GUI_WAIT_EVENT_FUNC)(void);
```

Additional information

Per default the GUI needs to periodically check for events unless a function is defined which waits and one that triggers an event. This function sets the function which waits for an event. Makes only sense in combination with `GUI_SetSignalEventFunc()` and `GUI_SetWaitEventTimedFunc()`. The advantage of using these functions instead of polling is the reduction of CPU load of the waiting task to 0% while it waits for input. If the function has been specified as recommended and the system waits for user input the defined function should wait for an event signaled from the function specified by `GUI_SetSignalEventFunc()`.
It is recommended to specify the function `GUI_X_WaitEvent()` for the job.

Example

```
GUI_SetWaitEventFunc(GUI_X_WaitEvent);
```

GUI_SetWaitEventTimedFunc()

Description

Defines a function which waits for an event for a dedicated period of time.

Prototype

```
void GUI_SetWaitEventTimedFunc(GUI_WAIT_EVENT_TIMED_FUNC pfWaitEventTimed);
```

Parameter	Description
pfWaitEventTimed	Pointer to a function that waits for an event.

Table 17.4: GUI_SetWaitEventTimedFunc() parameter list

Definition of GUI_WAIT_EVENT_TIMED_FUNC

```
typedef void (* GUI_WAIT_EVENT_TIMED_FUNC)(int Period);
```

Parameter	Description
Period	Period in ms to wait for an event.

Table 17.5: GUI_WAIT_EVENT_TIMED_FUNC parameter list

Additional information

Per default the GUI needs to periodically check for events unless a function is defined which waits and one that triggers an event. This function sets the function which waits for an event if a timer is active. Makes only sense in combination with GUI_SetSignalEventFunc() and GUI_SetWaitEventFunc(). If the function has been specified as recommended and the system waits for user input during a timer is active the defined function should wait until the timer expires or an event signaled from the function set by GUI_SetSignalEventFunc().
It is recommended to specify the function GUI_X_WaitEventTimed() for the job.

Example

```
GUI_SetWaitEventTimedFunc(GUI_X_WaitEventTimed);
```

17.6 Configuration macros for multitasking support

The following table shows the configuration macros used for a multitask system with multiple tasks calling emWin:

Type	Macro	Default	Description
N	GUI_MAXTASK	4	Defines the maximum number of tasks from which emWin is called when multitasking support is enabled.
B	GUI_OS	0	Activate to enable multitasking support.
F	GUI_X_SIGNAL_EVENT	-	Defines a function that signals an event. (Obsolete)
F	GUI_X_WAIT_EVENT	GUI_X_ExecIdle	Defines a function that waits for an event. (Obsolete)
F	GUI_X_WAIT_EVENT_TIMED	-	Defines a function that waits for an event for a dedicated period of time. (Obsolete)

Table 17.6: Configuration macros for multitasking support

GUI_MAXTASK

Description

Defines the maximum number of tasks from which emWin is called to access the display.

Type

Numerical value.

Additional information

This symbol is only relevant when GUI_OS is activated. If working with a pre-compiled library the function GUITASK_SetMaxTask() should be used instead. Further information can be found in the function description "GUITASK_SetMaxTask()" on page 1206.

GUI_OS

Description

Enables multitasking support by activating the module GUITask.

Type

Binary switch

0: inactive, multitask support disabled (default)
1: active, multitask support enabled

GUI_X_SIGNAL_EVENT

Description

Defines a function that signals an event.

Type

Function replacement

Additional information

Per default the GUI needs to periodically check for events unless a function is defined which waits and one that triggers an event. This macro defines the function which triggers an event. It makes only sense in combination with `GUI_X_WAIT_EVENT`. The advantage of using the macros `GUI_X_SIGNAL_EVENT` and `GUI_X_WAIT_EVENT` instead of polling is the reduction of CPU load of the waiting task to 0% while it waits for input. If the macro has been defined as recommended and the user gives the system any input (keyboard or pointer input device) the defined function should signal an event.
It is recommended to specify the function `GUI_X_SignalEvent()` for the job.

Example

```
#define GUI_X_SIGNAL_EVENT GUI_X_SignalEvent
```

GUI_X_WAIT_EVENT

Description

Defines a function which waits for an event.

Type

Function replacement

Additional information

Per default the GUI needs to periodically check for events unless a function is defined which waits and one that triggers an event. This macro defines the function which waits for an event. Makes only sense in combination with `GUI_X_SIGNAL_EVENT`. The advantage of using the macros `GUI_X_SIGNAL_EVENT` and `GUI_X_WAIT_EVENT` instead of polling is the reduction of CPU load of the waiting task to 0% while it waits for input. If the macro has been defined as recommended and the system waits for user input the defined function should wait for an event signaled from the function defined by the macro `GUI_X_SIGNAL_EVENT`.
It is recommended to specify the function `GUI_X_WaitEvent()` for the job.

Example

```
#define GUI_X_WAIT_EVENT GUI_X_WaitEvent
```

GUI_X_WAIT_EVENT_TIMED

Description

Defines a function which waits for an event for a dedicated period of time.

Type

Function replacement

Additional information

Per default the GUI needs to periodically check for events unless a function is defined which waits and one that triggers an event. This macro defines the function which waits for an event if a timer is active. Makes only sense in combination with `GUI_X_SIGNAL_EVENT`. If the macro has been defined as recommended and the sys-

tem waits for user input during a timer is active the defined function should wait until the timer expires or an event signaled from the function defined by the macro GUI_X_SIGNAL_EVENT.
It is recommended to specify the function GUI_X_WaitEventTimed() for the job.

Example

```
#define GUI_X_WAIT_EVENT_TIMED GUI_X_WaitEventTimed
```

17.7 Kernel interface API

An RTOS usually offers a mechanism called a resource semaphore, in which a task using a particular resource claims that resource before actually using it. The display is an example of a resource that needs to be protected with a resource semaphore. emWin uses the macro GUI_USE to call the function GUI_Use() before it accesses the display or before it uses a critical internal data structure. In a similar way, it calls GUI_Unuse() after accessing the display or using the data structure. This is done in the module GUITask.c.
GUITask.c in turn uses the GUI kernel interface routines shown in the table below. These routines are prefixed GUI_X_ since they are high-level (hardware-dependent) functions. They must be adapted to the real time kernel used in order to make the emWin task (or thread) safe. Detailed descriptions of the routines follow, as well as examples of how they are adapted for different kernels.

Routine	Description
GUI_X_GetTaskID()	Return a unique, 32-bit identifier for the current task/thread.
GUI_X_InitOS()	Initialize the kernel interface module (create a resource semaphore/ mutex).
GUI_X_Lock()	Lock the GUI (block resource semaphore/mutex).
GUI_X_SignalEvent()	Signals an event.
GUI_X_Unlock()	Unlock the GUI (unblock resource semaphore/mutex).
GUI_X_WaitEvent()	Waits for an event.
GUI_X_WaitEventTimed()	Waits a given period for an event.

Table 17.7: Kernel interface API list

GUI_X_GetTaskID()

Description

Returns a unique ID for the current task.

Prototype

```
U32 GUI_X_GetTaskID(void);
```

Return value

ID of the current task as a 32-bit integer.

Additional information

Used with a real-time operating system.
It does not matter which value is returned, as long as it is unique for each task/ thread using the emWin API and as long as the value is always the same for each particular thread.

GUI_X_InitOS()

Description

Creates the resource semaphore or mutex typically used by GUI_X_Lock() and GUI_X_Unlock().

Prototype

```
void GUI_X_InitOS(void)
```

GUI_X_Lock()

Description

Locks the GUI.

Prototype

```
void GUI_X_Lock(void);
```

Additional information

This routine is called by the GUI before it accesses the display or before using a critical internal data structure. It blocks other threads from entering the same critical section using a resource semaphore/mutex until `GUI_X_Unlock()` has been called.
When using a real time operating system, you normally have to increment a counting resource semaphore.

GUI_X_SignalEvent()

Description

Signals an event.

Prototype

```
void GUI_X_SignalEvent(void);
```

Additional information

This function is optional, it is used only via the macro `GUI_X_SIGNAL_EVENT` or the function `GUI_SetSignalEventFunc()`.

GUI_X_Unlock()

Description

Unlocks the GUI.

Prototype

```
void GUI_X_Unlock(void);
```

Additional information

This routine is called by the GUI after accessing the display or after using a critical internal data structure.
When using a real time operating system, you normally have to decrement a counting resource semaphore.

GUI_X_WaitEvent()

Description

Waits for an event.

Prototype

```
void GUI_X_WaitEvent(void);
```

Additional information

This function is optional, it is used only via the macro `GUI_X_WAIT_EVENT` or the function `GUI_SetWaitEventFunc()`.

GUI_X_WaitEventTimed()

Description

Waits for an event for the given period.

Prototype

```
void GUI_X_WaitEventTimed(int Period);
```

Parameter	Description
Period	Period in ms to be used.

Table 17.8: GUI_X_WaitEventTimed()

Additional information

This function is optional, it is used only via the macro `GUI_X_WAIT_EVENT_TIMED` or the function `GUI_SetWaitEventTimedFunc()`.

17.8 Examples

Kernel interface routines for embOS

The following example shows an adaption for embOS (excerpt from file `GUI_X_embOS.c` located in the folder `Sample\GUI_X`):

```
#include "RTOS.H"

static OS_TASK* _pGUITask;
static OS_RSEMA _RSema;

void GUI_X_InitOS(void)    { OS_CreateRSema(&_RSema);    }
void GUI_X_Unlock(void)    { OS_Unuse(&_RSema);          }
void GUI_X_Lock(void)      { OS_Use(&_RSema);            }
U32  GUI_X_GetTaskId(void) { return (U32)OS_GetTaskID(); }

void GUI_X_WaitEvent(void) {
  _pGUITask = OS_GetpCurrentTask();
  OS_WaitEvent(1);
}

void GUI_X_SignalEvent(void) {
  if (_pGUITask) {
    OS_SignalEvent(1, _pGUITask);
  }
}

void GUI_X_WaitEventTimed(int Period) {
  static OS_TIMER Timer;
  static int Initialized;

  if (Period > 0) {
    if (Initialized != 0) {
      OS_DeleteTimer(&Timer);
    }
    Initialized = 1;
    OS_CreateTimer(&Timer, GUI_X_SignalEvent, Period);
    OS_StartTimer(&Timer);
    GUI_X_WaitEvent();
  }
}
```

Kernel interface routines for uC/OS

The following example shows an adaption for uC/OS (excerpt from file `GUI_X_uCOS.c` located in the folder `Sample\GUI_X`):

```
#include "INCLUDES.H"

static OS_EVENT * pDispSem;
static OS_EVENT * pGUITask;

U32   GUI_X_GetTaskId(void) { return ((U32)(OSTCBCur->OSTCBPrio)); }
void  GUI_X_Unlock(void)    { OSSemPost(pDispSem);                 }
```



```
void  GUI_X_InitOS(void)    {
  pDispSem = OSSemCreate(1);
  pGUITask = OSSemCreate(0);
}

void  GUI_X_Lock(void)      {
  INT8U err;
  OSSemPend(pDispSem, 0, &err);
}
```

Kernel interface routines for Win32

The following is an excerpt from the Win32 simulation for emWin. When using the emWin simulation, there is no need to add these routines, as they are already in the library.
Note: cleanup code has been omitted for clarity.

```
/*********************************************************************
*
*       emWin - Multitask interface for Win32
*
**********************************************************************

  The following section consisting of 4 routines is used to make
  emWin thread safe with WIN32
*/

static HANDLE hMutex;

void GUI_X_InitOS(void) {
  hMutex = CreateMutex(NULL, 0, "emWinSim - Mutex");
}

unsigned int GUI_X_GetTaskId(void) {
  return GetCurrentThreadId();
}

void GUI_X_Lock(void) {
  WaitForSingleObject(hMutex, INFINITE);
}

void GUI_X_Unlock(void) {
  ReleaseMutex(hMutex);
}
```


Chapter 18

The Window Manager (WM)

When using the emWin Window Manager (WM), everything which appears on the display is contained in a window -- a rectangular area on the screen. A window can be of any size, and you can display multiple windows on the screen at once, even partially or entirely in front of other windows.
The Window Manager supplies a set of routines which allow you to easily create, move, resize, and otherwise manipulate any number of windows. It also provides lower-level support by managing the layering of windows on the display and by alerting your application to display changes that affect its windows.
The emWin Window Manager is a separate (optional) software item and is not included in the emWin basic package. The software for the Window Manager is located in the subdirectory `GUI\WM`.

18.1 Description of terms

Windows are rectangular in shape, defined by their origin (the X- and Y-coordinates of the upper left corner) as well as their X- and Y-sizes (width and height, respectively). A window in emWin:

- is rectangular.
- has a Z-position.
- may be hidden or shown.
- may have valid and/or invalid areas.
- may or may not have transparency.
- may or may not have a callback routine.

Active window

The window which is currently being used for drawing operations is referred to as the active window. It is not necessarily the same as the topmost window.

Callback routines

Callback routines are defined by the user program, instructing the graphic system to call a specific function when a specific event occurs. Normally they are used to automatically redraw a window when its content has changed.

Child windows

A child window is one that is defined relative to another window, called the parent. Whenever a parent window moves, its child or children move correspondingly. A child window is always completely contained within its parent, and will be clipped if necessary. Multiple child windows with the same parent are considered "siblings" to one another.

Client area

The client area of a window is simply its usable area. If a window contains a frame or title bar, then the client area is the rectangular inner area. If there is no such frame, then the coordinates of the client area are identical to those of the window itself.

Clipping, clip area

Clipping is the process of limiting output to a window or part of it.
The clip area of a window is its visible area. This is the window area minus the area obstructed by siblings of higher Z-order, minus any part that does not fit into the visible area of the parent window.

Coordinates

Coordinates are usually 2 dimensional coordinates, expressed in units of pixels. A coordinate consists of 2 values. The first value specifies the horizontal component (also called the x-coordinate), the second value specifies the vertical component (also called the y-coordinate).

Current window

See active window.

Desktop coordinates

Desktop coordinates are coordinates of the desktop window. The upper left position (the origin) of the display is (0,0).

Desktop window

The desktop window is automatically created by the Window Manager, and always covers the entire display area. It is always the bottommost window, and when no other window has been defined, it is the default (active) window. All windows are descendants (children, grandchildren, etc.) of the desktop window.

Early clipping

This is the default clipping mode. In this mode clipping is performed before windows receive paint events. In case the current window needs to be clipped, it will receive more than one WM_PAINT message within a single drawing process.
In the late clipping mode, windows always receive only one single WM_PAINT message. In this mode clipping is performed within the drawing operations.

Handle

When a new window is created, the WM assigns it a unique identifier called a handle. The handle is used in any further operations performed on that particular window.

Hiding windows

A hidden window is not visible, although it still exists (has a handle). When a window is created, it is hidden by default if no create flag is specified. Showing a window makes it visible; hiding it makes it invisible.

Late clipping

See early clipping.

Parent coordinates

Parent coordinates are window coordinates relative to the parent window. The upper left position (the origin) of the window is (0,0).

Parent windows

See child windows.

Showing windows

See Hiding windows.

Siblings

See child windows.

Transparency

A window that has transparency contains areas that are not redrawn with the rest of the window. These areas operate as though the window behind "shows through" them. In this case, it is important that the window behind is redrawn before the window with transparency. The WM automatically handles redrawing in the correct order.

Validation/invalidation

A valid window is a fully updated window which does not need redrawing.
An invalid window does not yet reflect all updates and therefore needs to be redrawn, either completely or partially. When changes are made that affect a particular window, the WM marks that window as invalid. The next time the window is redrawn (either manually or by a callback routine) it will be validated.

Window coordinates

Window coordinates are coordinates of a window. The upper left position (the origin) of the window is (0,0).

Z-position, bottom/top

Although a window is displayed on a two-dimensional screen in terms of X and Y, the WM also manages what is known as a Z-position, or depth coordinate -- a position in a virtual third dimension which determines its placement from background to foreground. Windows can therefore appear on top of or beneath one another.
Setting a window to the bottom will place it "underneath" all of its sibling windows (if any); setting it to the top will place it "on top of" its siblings. When a window is created, it is set to the top by default if no create flag is specified.

18.2 Callback mechanism, invalidation, rendering and keyboard input

The WM may be used with or without callback routines. In most cases, using callbacks is preferable.
The idea behind the callback mechanism that emWin offers for windows and window objects (widgets) is that of an event-driven system. As in most windowing systems, the principle is that the flow of control is not just from the user program to the graphic system, but also from the user program to the graphic system and back up to the user program by means of the callback routines provided by the user program. This mechanism -- often characterized as the Hollywood principle ("Don't call us, we'll call you!") -- is needed by the Window Manager mainly in order to trigger the redrawing of windows. This contrasts with classical programming, but it makes it possible to exploit the invalidation logic of the Window Manager.

18.2.1 Rendering without callbacks

You do not have to use callback routines, but in doing so, the WM loses the ability to manage redrawing (updating) of the windows. It is also possible to mix; for example, having some windows use callbacks and others not. However, if a window does not use the callback mechanism, your application is responsible for updating its contents.

Warning: When not using the callback mechanism, it is user responsibility to manage screen updates!

18.2.2 Rendering using callbacks

In order to create a window with a callback, you must have a callback routine. The routine is used as part of the `WM_CreateWindow()` function when creating the window (the `cb` parameter).
All callback routines must have the following prototype:

Prototype

```
void Callback(WM_MESSAGE * pMsg);
```

Parameter	Description
pMsg	Pointer to a data structure of type WM_MESSAGE.

Table 18.1: Callback function parameter list

The action performed by the callback routine depends on the type of message it receives. The prototype above is usually followed by a `switch` statement, which defines different behaviors for different messages using one or more `case` statements (typically at least `WM_PAINT`).

Processing the WM_PAINT message

When a window receives a `WM_PAINT` message, it should repaint itself. Before sending this message to the window, the WM makes sure it is selected.
A non transparent window (default!) has to repaint its entire invalid area.
The easiest way is to repaint the entire area of the window. The clipping mechanism of the WM makes sure that only the invalid area will be redrawn. In order to accelerate the drawing process, it can make sense to only repaint the invalid area. How to get the invalid area is described later in this chapter (Information is part of the message).
A transparent window on the other hand does not have to redraw the entire invalid area; it can leave the window area partially untouched. This untouched area will then be transparent.
Before the WM sends a `WM_PAINT` message to a transparent window, the area below has been redrawn (by sending a `WM_PAINT` message to the window(s) below).

Warning: Certain things should not be done when processing WM_PAINT

When processing the WM_PAINT message, the callback routine should do nothing but redrawing the contents of the window. When processing the `WM_PAINT` event, the following functions may not be called: `WM_SelectWindow()`, `WM_Paint()`, `WM_DeleteWindow()` and `WM_CreateWindow()`. Also any other functions which changes the properties of a window may not be called: `WM_Move()`, `WM_Resize()`, ...

Example

Creates a callback routine to automatically redraw a window:

```
void WinHandler(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(0xFF00);
    GUI_Clear();
    GUI_DispStringAt("Hello world",0,0);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}
```

The messages `WM_PRE_PAINT` and `WM_POST_PAINT` are sent directly before and after `the WM_PAINT` messages are processed.

18.2.3 Overwriting callback functions

The default behavior of widgets and windows in emWin is defined in their callback functions. If the behavior of a widget has to be changed, or if the functionality of a window needs to be enhanced to meet custom needs, it is recommended to overwrite the internal callback function. This is done in a few simple steps:

Step 1: Creating a custom callback function

The first step is to implement a function using the following prototype:

```
void Callback(WM_MESSAGE * pMsg);
```

Step 2: Messages

The second step is to implement a reaction to certain messages.
Since custom callback functions do not need to handle all possible messages, it is recommended to make use of a `switch` / `case` condition. This makes it possible to easily add or remove one message specific code, without affecting another. The parameter `pMsg` contains the Id of the message (`pMsg->MsgId`). A complete list of messages handled by the Window Manager may be reviewed under “List of messages” on page 388.

Step 3: Processing the default callback

The third step is to make sure all messages which are not handled by the custom callback function, are handled by the internal (default) callback function. The recommended way to do this is to use the default case of the switch / case condition to call the internal callback function.

Internal callback functions are different for each type of window. The internal callback functions for widgets are named <WIDGET>_Callback().
All other types of windows use the function WM_DefaultProc() for message handling.

```
switch (pMsg->MsgId) {
case WM_CREATE:
  .
  .
  .
  break;
case WM_PAINT:
  .
  .
  .
  break;
case WM_SIZE:
  .
```



```
  .
  .
  break;
default:
  <WIDGET>_Callback(pMsg);
}
```

Step 4: Setting the custom callback function to be used

The last step to do is setting the newly created callback function to be used by a window or widget. This is done with a simple call of WM_SetCallback(). Detailed information can be found under "WM_SetCallback()" on page 425.

18.2.4 Background window redrawing and callback

During initialization of the Window Manager, a window containing the whole LCD area is created as a background window. The handle of this window is WM_HBKWIN. The WM does not redraw areas of the background window automatically, because there is no default background color. That means if you create a further window and then delete it, the deleted window will still be visible. The routine WM_SetDesktopColor() needs to be specified in order to set a color for redrawing the background window.
You can also set a callback function to take care of this problem. If a window is created and then deleted as before, the callback routine will trigger the WM to recognize that the background window is no longer valid and redraw it automatically. For more information on using a callback routine to redraw the background, see the example at the end of the chapter.

18.2.5 Invalidation

Invalidation of a window or a part of it tells the WM that the invalid area of the window should be redrawn the next time GUI_Exec() or GUI_Delay() is called. The invalidation routines of emWin do not redraw the invalid part of a window. They only manage the invalid areas of the windows.

The invalid area of a window

The WM uses just one rectangle per window to store the smallest rectangle containing the entire invalid area. If for example a small part in the upper left corner and a small part in the lower right corner becomes invalid, the complete window is invalidated.

Why using invalidation

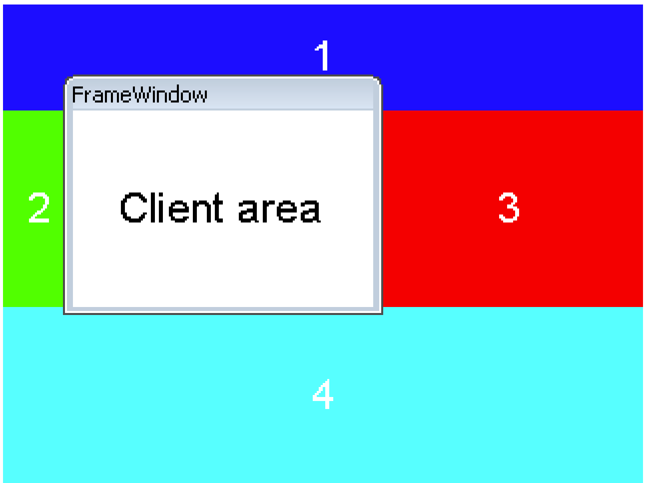
The advantage of using window invalidation in opposite of drawing each window immediately is that the window will be drawn only one time even if it is invalidated more than one time. If for example several properties of a window need to be changed (for example the background color, the font and the size of the window) it takes more time to draw the window immediately after each property has been changed than drawing the window only one time after all properties have been changed.

Redrawing of invalid windows

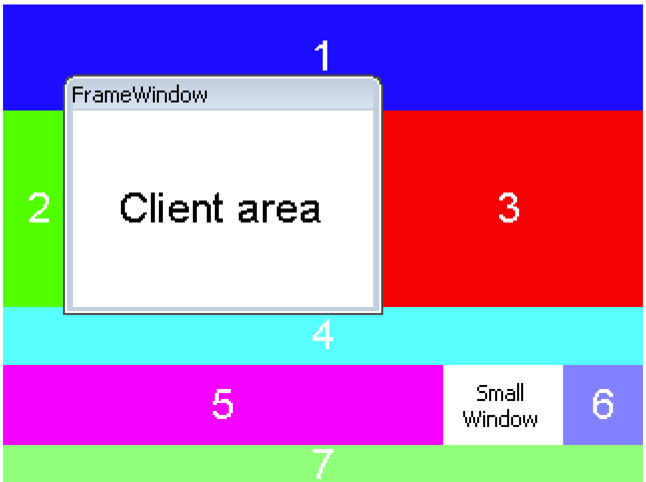
The function GUI_Exec() redraws all invalid windows. This is done by sending one or more WM_PAINT messages to each invalid window.

18.2.6 Tiling mechanism

More than one message will be send if a window is covered by an other window, for example by a child window. The WM cuts the non covered area of the window to be drawn into a number of sub rectangles. During that process it sets the clipping area to each of the rectangular areas and sends a WM_PAINT message to the window.



The sample on the right shows the tiling mechanism required for drawing the background window covered by a FRAMEWINDOW widget. The widget consists of 2 windows, the main window and the client area. The main window is a transparent window which is drawn on top of the background and has no effect on the tiling algorithm whereas the client area is opaque and causes tiling of the background. The WM generates tiles around the client area from the top to the bottom and from the left to the right. The tiling algorithm implies that the number of tiles increases with the number of covered areas.



On the right the screenshot shows the above sample with an additional small window at the bottom right edge. When starting the drawing process of a non transparent window the WM first sends a WM_PRE_PAINT message. After that the window receives a WM_PAINT message for each tile. And after drawing the last tile the WM sends a final WM_POST_PAINT message.
The more areas covering a non transparent window the more tiles are required for drawing the window.

Drawing without tiling

Under some circumstances it may could make sense to supress tiling. That could be achieved by using the flag WM_CF_LATE_CLIP explained later in this chapter.

18.2.7 Rendering of transparent windows

If a transparent window needs to be drawn, the WM automatically makes sure, that the background of the window is drawn before the transparent window receives a WM_PAINT message. This is done by redrawing all window areas below the invalid area of the transparent window first before sending a WM_PAINT message to the transparent window.
To make sure the Window Manager can handle the redrawing of transparent windows it is necessary to redraw the window in reaction to the WM_PAINT message. Otherwise it can not be guaranteed that the appearance of a transparent window will be correctly.
The use of transparent windows is more CPU-intensive than the use of non transparent windows. If performance is a problem, trying to avoid transparent windows may be an option.

18.2.8 Automatic use of Memory Devices

The default behavior of the Window Manager is sending a WM_PAINT message to each window which needs to be redrawn. This can cause flickering effects. To suppress these ’per window’ flickering effects Memory Devices can be used automatically for the drawing operation. This can be achieved by setting the flag WM_CF_MEMDEV when creating the window, by setting the default creation flags with WM_SetCreateFlags() or by using the function WM_EnableMemdev(). The WM then redirects the output of the WM_PAINT message into a Memory Device which is copied to the display once the

actual drawing was performed. If not enough memory for the whole window is available banding is used automatically. The according Memory Device is created internally just before the `WM_PAINT` message is sent and is removed immediately after the drawing is finished.
In case a transparent window should be drawn, the below area will also be drawn into the Memory Device.
Detailed information about Memory Devices can be found in the chapter “Memory Devices” on page 319.

18.2.9 Automatic use of multiple frame buffers

The WM is able to use automatically multiple frame buffers if they are available. This can be achieved by the function `WM_MULTIBUF_Enable()`. If enabled the Window Manager redirects the output of all drawing functions to the invisible back buffer before it draws the invalid windows. After the last invalid window has been drawn the WM makes the back buffer visible. This feature is available only if the display driver supports multiple buffers and if there is enough RAM to store at least 2 frame buffers. More information can be found in the chapter “Multiple Buffering” on page 935.

18.2.10 Automatic use of display driver cache

The WM automatically uses the display driver cache if available. If available it locks the buffer before it starts to draw the invalid windows. After the last window has been drawn the WM unlocks the cache.

18.2.11 Keyboard input

The Window Manager handles keyboard input automatically. It polls the keyboard buffer and sends according keyboard messages to the currently focussed window. The keyboard buffer can be filled using the function `GUI_StoreKeyMsg()` which is described on page 999.

18.3 Motion support

Motion support enables the ability to move windows by gestures. It can be used with any pointer input device (PID) like a touch screen, a mouse or a joystick. If motion support is enabled the respective window can be put into movement simply with a gesture. After releasing the PID the movement is decelerated within a specified period. Movement operations can be also initiated by API functions instead of gestures.

18.3.1 Enabling motion support of the WM

First of all motion support needs to be enabled before it can be used. This can be done by calling the function `WM_MOTION_Enable()` once. Without calling this function once the motion support functions won't work.

18.3.2 Basic motion support for a window

To be able to use motion support for a window it needs to be enabled for each window which should be moveable. In case of a moveable parent window with several child windows motion support needs only be enabled for the parent window.
There are 2 possibilities to achieve basic motion support for a window:

18.3.2.1 Using creation flags

To achieve movability for a window it can be created with one or more or-combined creation flags. The following table shows the available creation flags:

Flag	Description
WM_CF_MOTION_R	Enables circular moveability for objects within the window.
WM_CF_MOTION_X	Enables movability for the X axis.
WM_CF_MOTION_Y	Enables movability for the Y axis.

Table 18.2: Creation flags

Example

```
WM_HWIN hWin;
hWin = WM_CreateWindowAsChild(0, 0, 40, 40, hParent,
                              WM_CF_SHOW | WM_CF_MOTION_X | WM_CF_MOTION_Y, cbWin, 0);
```

Of course the motion flags can also be used with widget creation functions.

18.3.2.2 Using API function

To achieve movability for a window after it has been created without movability flags the function `WM_MOTION_SetMoveable()` explained later in this chapter can be used.

18.3.3 Advanced motion support

Advanced motion support describes circular movement of windows or edge snapping. In order to make use of advanced motion features the callback function of the moveable window has to be adapted. In case the WM recognizes PID movement it sends a `WM_MOTION` message to the window. In order to achieve advanced motion support an appropriate reaction to the `WM_MOTION` message needs to be implemented.

18.3.3.1 WM_MOTION message and WM_MOTION_INFO

As explained in the message description “WM_MOTION” on page 393 the Data.p element of the `WM_MOTION` message points to a `WM_MOTION_INFO` structure.

The element `Cmd` of this structure contains information about the current operation. The following table shows the possible values of the element `Cmd`:

Flag	Description
WM_MOTION_INIT	Sent to a window to initiate a motion operation.
WM_MOTION_MOVE	Sent to a window to achieve custom moving operations.
WM_MOTION_GETPOS	Sent to get the current position of custom moving operations.

Table 18.3: Motion flags description

WM_MOTION_INIT

If a PID move has been detected by the WM it first checks if there is any visible window available under the PID position which is already 'moveable'. This makes it possible to achieve moving operations for windows which are partially or totally covered by child windows. If the WM does not find an already moveable window it sends the command to the 'top window' of the PID position.
If the window is not already 'moveable' when receiving this command the element `Flags` of the `WM_MOTION_INFO` structure can be used to enable motion support. The creation flags explained earlier can be used here to achieve automatic motion support. The `Flags` element simply needs to be OR-combined with the desired flag(s).

WM_MOTION_INIT and custom motion support

Custom motion support means that the moving operations are not done automatically by the WM but by the callback routine of the window. This can be useful if for example radial motions are required. To achieve custom motion support the `Flags` element needs to be OR-combined with the flag `WM_MOTION_MANAGE_BY_WINDOW`.

WM_MOTION_MOVE

Sent to a window with custom motion support enabled. The elements `dx` and `dy` of the `WM_MOTION_INFO` structure can be used to achieve the custom moving operation.

WM_MOTION_GETPOS

Sent to a window with custom motion support enabled. The task of the callback routine here is returning the current position. This needs to be done with the elements `xPos` and `yPos` of the `WM_MOTION_INFO` structure.

Snapping

The elements `SnapX` and `SnapY` of the `WM_MOTION_INFO` structure can be used to achieve snapping. These values determine a kind of grid for snapping. This means that the deceleration of the movement operation will stop exactly on a grid position. Also if there currently is no movement and the window is only released it will snap into the next grid position.

Examples

The sample folder contains the sample WM_RadialMenu.c which can be used to get an overview about how advanced motion support can be used. A second sample WM_Motion.c shows how to use simple motion support.

18.3.4 Motion support for circular moves

That kind of motion support can be used to turn items arround the center of a window. Windows can not be rotated. To enable support for circular moves the flag `WM_CF_MOTION_R` should be used which should be set within the `WM_MOTION_INIT` message. Moving the items need to be managed by the application. Because of that also the flag `WM_MOTION_MANAGE_BY_WINDOW` explained earlier must be set. The values in 1/10 degrees to be used are passed to the application in the element '`da`' of the `WM_MOTION_INFO` structure available within the message `WM_MOTION_MOVE`. The KNOB widget for example is completely based on motion support for circular moves.

Example:

```
static void _OnMotion(WM_HWIN hWin, WM_MOTION_INFO * pInfo) {
  ...
  switch (pInfo->Cmd) {
```

```
    case WM_MOTION_INIT:
      pInfo->Flags  = WM_CF_MOTION_R | WM_MOTION_MANAGE_BY_WINDOW;
      break;
    case WM_MOTION_MOVE:
      _DoMotion(hObj, pInfo->da);
      break;
    ...
    }
  }
```


18.4 ToolTips

A ToolTip in emWin is a small window with one line of text, which appears in conjunction with a pointer input device (PID), usually a mouse. The user hovers the PID over a 'tool', without clicking it, and a small ToolTip window with information about the item being hovered over may appear. After a short time the window disappears automatically. ToolTips make sense for active elements like any kind of button or similar widgets/windows, which can be used as tool for changing something. But they can be used with any kind of window.

18.4.1 How they work

A ToolTip belongs to a particular parent (or grandparent) window. When the PID hovers over a tool window without any motion, after a specified time (PERIOD_FIRST) the ToolTip window occurs. If the PID remains over the tool without motion, the ToolTip automatically disappears after a specified period of no motion (PERIOD_SHOW). It remains until the PID does not move for this period. If the PID is clicked or hovers out of the tool window the ToolTip disappears. If the PID remains in the parent area and the PID then hovers again over a tool of the same parent, the ToolTip occurs immediately after a very short period (PERIOD_NEXT) of no motion. If the PID moves out of the parent area, the next time a ToolTip occurs is again after PERIOD_FIRST. Appearance and timing can be configured at runtime.

18.4.2 Creating ToolTips

(The functions and structures mentioned in the following are described in detail later in this chapter under "WM API: ToolTip related functions" on page 436.)
The function WM_TOOLTIP_Create() should be used for creating a ToolTip object. It requires a handle to the parent (or grand parent) window. Optional a pointer to an array of TOOLTIP_INFO structures can be passed which is used for adding the desired tools to the ToolTip object. These structures should contain the IDs of the tools and the text to be shown. Alternatively the function WM_TOOTIP_AddTool() can be used to add the tools. This makes sense if the tool window does not have an Id.

Elements of structure TOOLTIP_INFO

Data type	Element	Description
int	Id	Id of the ToolTip.
const char *	pText	String containing the text of the ToolTip.

Table 18.4: TOOLTIP_INFO element list

18.4.2.1 Creating ToolTips for dialog items

As mentioned above the TOOLTIP_INFO structure is used to address the desired tools by its IDs. Because the items of a dialog normally have an Id this is quite easy.

Example

The following sample shows how it works:

```
#include "DIALOG.h"

#define ID_BUTTON_0 (GUI_ID_USER + 0x01)
#define ID_BUTTON_1 (GUI_ID_USER + 0x02)

static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Framewin",         0, 0,  0, 320, 240, 0, 0, 0 },
  { BUTTON_CreateIndirect, "Button 0", ID_BUTTON_0, 5,  5,  80,  20, 0, 0, 0 },
  { BUTTON_CreateIndirect, "Button 1", ID_BUTTON_1, 5, 30,  80,  20, 0, 0, 0 },
};

static const TOOLTIP_INFO _aInfo[] = {
  { ID_BUTTON_0, "I am Button 0" },
  { ID_BUTTON_1, "I am Button 1" },
};

static void _ShowDialog(void) {
```



```
  WM_HWIN hWin;
  WM_TOOLTIP_HANDLE hInfo;

  hWin = GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), 0,
                             WM_HBKWIN, 0, 0);
  hInfo = WM_TOOLTIP_Create(hWin, _aInfo, GUI_COUNTOF(_aInfo));
  while (1) {
    GUI_Delay(100);
  }
}
```

18.4.2.2 Creating ToolTips for simple windows

Because simple windows normally do not have an Id, there exists a function for adding tools without using IDs. The function `WM_TOOLTIP_AddTool()` can be used to do this by passing the tool window handle and the required text to be shown.

Example

The following example shows how it works:

```
#include <stddef.h>
#include "WM.h"

static void _cbParent(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(GUI_BLUE);
    GUI_Clear();
    GUI_DispString("Parent window");
    break;
  }
}

static void _cbTool(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(GUI_RED);
    GUI_Clear();
    GUI_DispString("Tool window");
    break;
  }
}

void MainTask(void) {
  WM_HWIN hTool, hParent;
  WM_TOOLTIP_HANDLE hToolTip;

  GUI_Init();
  WM_SetDesktopColor(GUI_BLACK);
  hParent = WM_CreateWindow(0, 0, 200, 100, WM_CF_SHOW, _cbParent, 0);
  hTool   = WM_CreateWindowAsChild(20, 20, 100, 50, hParent, WM_CF_SHOW, _cbTool, 0);
  hToolTip = WM_TOOLTIP_Create(hParent, NULL, 0);
  WM_TOOLTIP_AddTool(hToolTip, hTool, "I am a ToolTip");
  while (1) {
    GUI_Delay(100);
  }
}
```


18.5 Messages

The following section shows which system messages are used by emWin, how to use the message data and how to use application defined messages.

18.5.1 Message structure

When a callback routine is called, it receives the message specified as its pMsg parameter. This message is actually a WM_MESSAGE data structure, with elements defined as follows.

Elements of structure WM_MESSAGE

Data type	Element	Description
int	MsgId	Type of message. See list of messages below.
WM_HWIN	hWin	Destination window.
WM_HWIN	hWinSrc	Source window.
union	Data	Data union. See elements below.
Elements of Data		
void *	p	Data pointer.
int	v	Data value.

Table 18.5: WM_MESSAGE element list

18.5.2 List of messages

The following messages are defined by emWin.

Message Id (MsgId)	Description
System defined messages	
WM_CREATE	Sent immediately after a window has been created, giving the window the chance to initialize and create any child windows.
WM_DELETE	Sent just before a window is deleted, telling the window to free its data structures (if any).
WM_GET_ACCEPT_FOCUS	Sent to a window to determine if the window is able to receive the input focus.
WM_GET_ID	Sent to a window to request the Id of the window.
WM_INIT_DIALOG	Sent to a dialog window immediately after the creation of the dialog.
WM_KEY	Sent to the window currently containing the focus if a key has been pressed.
WM_MOVE	Sent to a window immediately after it has been moved.
WM_NOTIFY_PARENT	Informs a parent window that something has occurred in one of its child windows.
WM_NOTIFY_VIS_CHANGED	Sent to a window if its visibility has been changed.
WM_PAINT	Sent to a window after it has become invalid and it should be redrawn.
WM_POST_PAINT	Sent to a window after the last WM_PAINT message was processed.
WM_PRE_PAINT	Sent to a window before the first WM_PAINT message is sent.
WM_SET_FOCUS	Sent to a window if it gains or loses the input focus.
WM_SET_ID	Sent to a window to change the window Id.
WM_SIZE	Sent to a window after its size has changed.
WM_TIMER	Sent to a window after a timer has expired.
Pointer input device (PID) messages	
WM_MOTION	Sent to a window to achieve advanced motion support.

Table 18.6: Message list

Message Id (MsgId)	Description
WM_MOUSEOVER	Sent to a window if a pointer input device touches the outline of a window. Only sent if mouse support is enabled.
WM_MOUSEOVER_END	Sent to a window if a pointer input device has been moved out of the outline of a window. Only sent if mouse support is enabled.
WM_PID_STATE_CHANGED	Sent to the window pointed by the pointer input device when the pressed state has been changed.
WM_TOUCH	Sent to a window once a pointer input device is pressed, pressed and moved or released over its area.
WM_TOUCH_CHILD	Sent to a parent window if a child window has been touched by the pointer input device.
Notification codes	
WM_NOTIFICATION_CHILD_DELETED	This notification message will be sent from a window to its parent before it is deleted.
WM_NOTIFICATION_CLICKED	This notification message will be sent when the window has been clicked.
WM_NOTIFICATION_GOT_FOCUS	This notification message will be sent once a window receives and accepts the focus.
WM_NOTIFICATION_LOST_FOCUS	This notification message will be sent when the window has lost the focus.
WM_NOTIFICATION_MOVED_OUT	This notification message will be sent when the pointer was moved out of the window while it is clicked.
WM_NOTIFICATION_RELEASED	This notification message will be sent when a clicked widget has been released.
WM_NOTIFICATION_SCROLL_CHANGED	This notification message will be sent when the scroll position of an attached SCROLLBAR widget has changed.
WM_NOTIFICATION_SCROLLBAR_ADDED	This notification message will be sent when a SCROLLBAR widget has been added to the window.
WM_NOTIFICATION_SEL_CHANGED	This notification message will be sent when the selection of a widget has changed.
WM_NOTIFICATION_VALUE_CHANGED	This notification message will be sent when a widget specific value has changed.
User defined messages	
WM_USER	The WM_USER constant could be used by applications to define private messages, usually of the form (WM_USER + X), where X is an integer value.

Table 18.6: Message list

18.5.3 System-defined messages

These kind of messages are sent by the GUI library. Do not send system defined messages from the user application to a window or a widget.

WM_CREATE

Description

This message is sent immediately after a window has been created, giving the window the chance to initialize and create any child windows.

Data

This message contains no data.

WM_DELETE

Description

This message is sent just before a window is deleted, telling the window to free its data structures (if any).

Data

This message contains no data.

WM_GET_ID

Description

This message is sent to a window to request it's Id. All emWin widgets handle this message. Application defined windows should handle this message in their callback routine. Otherwise this message will be ignored.

Data

The callback routine of the window should store the Id in the `Data.v` value.

WM_INIT_DIALOG

Description

This message is sent to a window immediately after the creation of the dialog and before the dialog is displayed. Dialog procedures typically use this message to initialize widgets and carry out any other initialization tasks that affect the appearance of the dialog box.

Data

This message contains no data.

WM_KEY

Description

Sent to the window currently containing the focus if a key has been pressed.

Data

The `Data.p` pointer of the message points to a `WM_KEY_INFO` structure.

Elements of structure WM_KEY_INFO

Data type	Element	Description
`int`	`Key`	The key which has been pressed.
`int`	`PressedCnt`	> 0 if the key has been pressed, 0 if the key has been released.

Table 18.7: WM_KEY_INFO element list

WM_MOVE

Description

This message is sent to a window immediately after it has been moved. If the window has any child windows, they will be moved first. Also each child window will receive this message after it has been moved. The message is sent regardless if the window is visible or not.

Data

The `Data.p` pointer of the message points to a `WM_KEY_INFO` structure.

Elements of structure WM_MOVE_INFO

Data type	Element	Description
int	dx	Difference between old and new position on the x-axis.
int	dy	Difference between old and new position on the y-axis.

Table 18.8: WM_MOVE_INFO element list

WM_NOTIFY_PARENT

Description

Informs a parent window that something has occurred in one of its child window. These messages are typically sent by widgets to their parent windows to give them a chance to react on the event.

Data

The `Data.v` value of the message contains the notification code of the message. For more information about the notification codes, refer to the appropriate widget.

WM_NOTIFY_VIS_CHANGED

Description

This message is sent to a window if its visibility is changed and the configuration switch `WM_SUPPORT_NOTIFY_VIS_CHANGED` is set to 1. The visibility of a window changes if

- obstruction changes: The window is partially or totally covered or uncovered by a higher level window (a window which is displayed on top of the window),
- the window is deleted or
- the window changes from not hidden to hidden or reverse.

Typical application

Applications which show a video in a window using a hardware decoder. The hardware decoder can write directly into the display, bypassing emWin, if the window containing the video is completely visible. If the visibility changes, the hardware decoder needs to be reprogrammed.

Example

The following shows a typical reaction on this message:

```
case WM_NOTIFY_VIS_CHANGED:
  if (WM_IsCompletelyVisible(WM_GetClientWindow(pMsg->hWin))) {
    ...
  }
```

The `Sample` folder of emWin contains the example `WM_Video.c` which shows how to use the message.

Data

This message contains no data.

WM_PAINT

Description

The WM sends this message to a window if it has become invalid (partially or complete) and needs to be drawn. When a window receives a `WM_PAINT` message, it should repaint itself. Before sending this message to the window, the WM makes sure it is selected. More details about how to react on the `WM_PAINT` message is described earlier in this chapter under "Using callback routines".

Data

The `Data.p` pointer of the message points to a `GUI_RECT` structure containing the invalid rectangle of the window in screen coordinates. This information could be used to optimize the paint function.

WM_POST_PAINT

Description

The WM sends this message to a window right after the last WM_PAINT message was processed.

Data

This message contains no data.

WM_PRE_PAINT

Description

The WM sends this message to a window before the first WM_PAINT is sent.

Data

This message contains no data.

WM_SET_FOCUS

Description

Sent to a window if it gains or loses the input focus.

Data

If the window gains the input focus, the Data.v value is set to 1. If the window 'accepts' the input focus, it should set the Data.v value to 0 in reaction on this message.
If the window loses the input focus, the Data.v value is set to 0.

WM_SET_ID

Description

Sent to a window to change the Id. All emWin widgets handle this message. Application defined windows should handle this message in their callback routine. Otherwise this message will be ignored.

Data

The Data.v value contains the new Id of the window.

WM_SIZE

Description

Sent to a window after its size has changed. Gives the window the chance to reposition its child windows (if any).

Data

This message contains no data.

WM_TIMER

Description

This message will be sent to a window when a timer created by `WM_CreateTimer()` has expired.

Data

The Data.v value contains the handle of the expired timer.

18.5.4 Pointer input device (PID) messages

These kind of messages are sent by the GUI library in reaction of PID input. Do not send this messages from the user application to a window or a widget.

WM_MOTION

Description

A WM_MOTION message is sent to a window to achieve advanced motion support. It is sent if a pointer input device is moved over a moveable window and to initiate a moving operation.
Detailed information about Motion Support can be found in the section “Motion support” on page 383.

Data

The Data.p pointer of the message points to a WM_MOTION_INFO structure.

Elements of structure WM_MOTION_INFO

Data type	Element	Description
int	Cmd	Details can be found in the section “Motion support” on page 383.
int	dx	Distance in X to be used to move the window.
int	dy	Distance in Y to be used to move the window.
int	da	Distance in 1/10 dregrees to be used to move an item.
int	xPos	Used to return the current position in X for custom moving operations.
int	yPos	Used to return the current position in Y for custom moving operations.
int	Period	Duration of the moving operation after the PID has been released.
int	SnapX	Raster size in X for snapping operations, 0 if no snapping is required.
int	SnapY	Raster size in Y for snapping operations, 0 if no snapping is required.
int	FinalMove	Set to 1 on the final moving operation.
U32	Flags	To be used to enable motion support.

Table 18.9: WM_MOTION_INFO element list

WM_MOUSEOVER

Description

A WM_MOUSEOVER message is sent to a window if a pointer input device touches the outline of a window. It is sent only if mouse support is enabled. This message is not sent to disabled windows.
To enable mouse support, add the following line to the file GUIConf.h:

```
#define GUI_SUPPORT_MOUSE 1
```

Data

The Data.p pointer of the message points to a GUI_PID_STATE structure.

Elements of structure GUI_PID_STATE

Data type	Element	Description
int	x	Horizontal position of the PID in window coordinates.
int	y	Vertical position of the PID in window coordinates.
U8	Pressed	Is always set to 0 when receiving a WM_MOUSEOVER message.

Table 18.10: GUI_PID_STATE element list

WM_MOUSEOVER_END

Description

A WM_MOUSEOVER_END message is sent to a window if the mouse pointer has been moved out of the window. It is sent only if mouse support is enabled. This message is not sent to disabled windows.

Data

The Data.p pointer of the message points to a GUI_PID_STATE structure. For details about this structure, refer to the message WM_MOUSEOVER.

WM_PID_STATE_CHANGED

Description

Sent to the window affected by the pointer input device when the pressed state has changed. The affected window is the visible window at the input position. With other words: If the user releases for example the touch screen over a window, the pressed state changes from 1 (pressed) to 0 (unpressed). In this case a WM_PID_STATE_CHANGED message is sent to the window. This message is sent before the touch message is sent. An invisible window does not receive this message. Transparent windows are handled the same way as visible windows.
This message is not sent to disabled windows.

Data

The Data.p pointer of the message points to a WM_PID_STATE_CHANGED_INFO structure.

Elements of structure WM_PID_STATE_CHANGED_INFO

Data type	Element	Description
int	x	Horizontal position of the PID in window coordinates.
int	y	Vertical position of the PID in window coordinates.
U8	State	Pressed state (> 0 if PID is pressed).
U8	StatePrev	Previous pressed state

Table 18.11: WM_PID_STATE_CHANGED_INFO element list

WM_TOUCH

Description

A WM_TOUCH message is sent to a window once the PID
- is pressed.
- is moved in pressed state.
- is released.

Windows receive this message, if one of the actions above happens over the visible area and if they are not disabled.

Data

The Data.p pointer of the message points to a GUI_PID_STATE structure. Data.p == NULL means that the PID was moved out of bounds in pressed state.

Elements of structure GUI_PID_STATE

Data type	Element	Description
int	x	Horizontal position of the PID in window coordinates.
int	y	Vertical position of the PID in window coordinates.
U8	Pressed	If the message is originated by a touch screen this value can be 0 (unpressed) or 1 (pressed). If the message is originated by a mouse each bit represents a mouse button (0 for unpressed and 1 for pressed state): - Bit 0 represents the first button (normally the left button) - Bit 1 represents the second button (normally the right button) - Bit 2 represents the third button (normally the middle button) The remaining bits can be used for further buttons.

Table 18.12: GUI_PID_STATE element list

WM_TOUCH_CHILD

Description

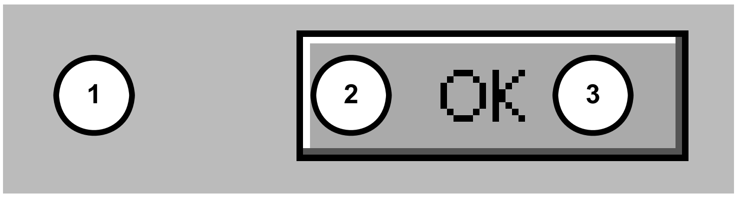
This message is sent to the parent window if the outline of a child window has been touched with a pointer input device in pressed or unpressed state. This message is not sent to disabled windows.

Data

The `Data.p` pointer of the message points to the touch message sent to the child window. Details about the message data can be found under “WM_TOUCH” on page 394.

Example

The following example explains what happens if a pointer input device is dragged over a dialog with a button:



Position	Description
1	The pointer input device (PID) is pressed at this position. This causes the WM to send the following WM_PID_STATE_CHANGED message to the window at this position: x = Horizontal position in desktop coordinates. y = Vertical position in desktop coordinates. State = 1 StatePrev = 0 The WM also sends a WM_TOUCH message with the same x and y coordinates to the window: x = Horizontal position in desktop coordinates. y = Vertical position in desktop coordinates. Pressed = 1
2	The PID is dragged to this position. The window below (the button) will receive no WM_PID_STATE_CHANGED message, because the PID remains in pressed state. The WM only sends a WM_TOUCH message to the window: x = Horizontal position in desktop coordinates. y = Vertical position in desktop coordinates. Pressed = 1
3	The PID is released at this position. This causes the WM to send the following WM_PID_STATE_CHANGED message to the window at this position: x = Horizontal position in desktop coordinates. y = Vertical position in desktop coordinates. State = 0 StatePrev = 1 The WM also sends a WM_TOUCH message with the same x and y coordinates to the window: x = Horizontal position in desktop coordinates. y = Vertical position in desktop coordinates. Pressed = 0

Table 18.13: WM_TOUCH example

18.5.5 System-defined notification codes

A message of this type is sent from a window to its parent window to notify it of a change in the child window. This gives the parent window the chance to react on this event. The message contains a hWinSrc element which is a handle to the widget which caused the message. Detailed information about which notification messages are utilized by a widget can be found in the according Widget description in the chapter “Widgets (window objects)” on page 449.

Note: Do not send system defined notification codes from the user application to a window.

WM_NOTIFICATION_CHILD_DELETED

This notification message will be sent from a window to its parent before it is deleted.

WM_NOTIFICATION_CLICKED

This notification message will be sent when the window has been clicked.

WM_NOTIFICATION_LOST_FOCUS

This notification message will be sent when the window has lost the focus.

WM_NOTIFICATION_MOVED_OUT

This notification message will be sent when the pointer was moved out of the window while it is clicked.

WM_NOTIFICATION_RELEASED

This notification message will be sent when a clicked widget has been released.

WM_NOTIFICATION_SCROLL_CHANGED

This notification message will be sent when the scroll position of an attached SCROLLBAR widget has changed.

WM_NOTIFICATION_SCROLLBAR_ADDED

This notification message will be sent when a SCROLLBAR widget has been added to the window.

WM_NOTIFICATION_SEL_CHANGED

This notification message will be sent when the selection of a widget has changed.

WM_NOTIFICATION_VALUE_CHANGED

This notification message will be sent when a widget specific value has changed.

18.5.6 Application-defined messages

The application program can define additional messages for its own usage. In order to ensure that they custom message Ids do not equal the Ids which are predefined in emWin, user-defined messages start numbering at `WM_USER`. Defining custom messages is recommended as follows:

```
#define MY_MESSAGE_AAA (WM_USER + 0)
#define MY_MESSAGE_BBB (WM_USER + 1)
```


18.6 Configuration options

Type	Macro	Default	Description
B	WM_SUPPORT_NOTIFY_VIS_CHANGED	0	Enables the WM to send a WM_NOTIFY_VIS_CHANGED message to a window if its visibility is changed.
B	WM_SUPPORT_TRANSPARENCY	1	Enable support for transparent windows. If set to 0 the additional code for transparency support is not included.

Table 18.14: Configuration options

WM_SUPPORT_NOTIFY_VIS_CHANGED

Per default emWin does not inform windows if their visibility has changed. If enabled, the WM sends WM_NOTIFY_VIS_CHANGED messages.

WM_SUPPORT_TRANSPARENCY

Per default emWin supports transparent windows. This means per default the additional code used to handle transparent windows is linked if the WM is used. If the application does not use transparent windows the memory requirement of the application can be reduced if WM_SUPPORT_TRANSPARENCY is set to 0.

18.7 WM API

The following table lists the available routines of the emWin Window Manager API. All functions are listed in alphabetical order within their respective categories. Detailed descriptions of the routines can be found later in the chapter.

Routine	Description
	Basic functions
WM_Activate()	Activates the Window Manager.
WM_AttachWindow()	Attaches a window to a new parent window.
WM_AttachWindowAt()	Attaches a window to a new parent window at the given position.
WM_BroadcastMessage()	Sends a message to all existing windows.
WM_BringToBottom()	Places a window behind its siblings.
WM_BringToTop()	Places a window in front of its siblings.
WM_ClrHasTrans()	Clears the transparency flag.
WM_CreateWindow()	Creates a window.
WM_CreateWindowAsChild()	Creates a child window.
WM_Deactivate()	Deactivates the Window Manager.
WM_DefaultProc()	Default routine to handle messages.
WM_DeleteWindow()	Deletes a window.
WM_DetachWindow()	Detaches a window from its parent window.
WM_DisableWindow()	Disable the given window.
WM_EnableWindow()	Sets the window state to enabled (default).
WM_Exec()	Redraws invalid windows by executing callbacks (all jobs).
WM_Exec1()	Redraws one invalid window by executing one callback (one job only).
WM_ForEachDesc()	Iterates over all descendants of a window.
WM_GetActiveWindow()	Returns handle of the active window.
WM_GetCallback()	Returns a pointer to the callback function of a window.
WM_GetClientRect()	Returns the size of the active window.
WM_GetClientRectEx()	Returns the size of a window.
WM_GetDesktopWindow()	Returns the window handle of the desktop window
WM_GetDesktopWindowEx()	Returns the window handle of the specified desktop window
WM_GetDialogItem()	Returns the window handle of a dialog box item (widget).
WM_GetFirstChild()	Returns handle of a window’s first child window.
WM_GetFocussedWindow()	Returns the handle of the window with the input focus.
WM_GetHasTrans()	Returns current value of the has transparency flag.
WM_GetInvalidRect()	Returns the invalid rectangle of the given window.
WM_GetModalLayer()	Returns the current modal layer.
WM_GetNextSibling()	Returns the handle of a window’s next sibling.
WM_GetOrgX()	Returns the origin in X of the active window.
WM_GetOrgY()	Returns the origin in Y of the active window.
WM_GetParent()	Returns handle of a window’s parent window.
WM_GetPrevSibling()	Returns the handle of a window’s previous sibling.
WM_GetStayOnTop()	Returns current value of the stay on top flag.
WM_GetUserData()	Retrieves the user data of a window
WM_GetWindowOrgX()	Returns the origin in X of a window.
WM_GetWindowOrgY()	Returns the origin in Y of a window.
WM_GetWindowRect()	Returns the screen coordinates of the active window.
WM_GetWindowRectEx()	Returns the screen coordinates of a window.
WM_GetWindowSizeX()	Returns the horizontal size (width) of a window.
WM_GetWindowSizeY()	Returns the vertical size (height) of a window.

Table 18.15: Window Manager API list

Routine	Description
WM_HasCaptured()	Checks if the given window has captured mouse- and touch-screen-input.
WM_HasFocus()	Checks if the given window has the input focus.
WM_HideWindow()	Makes a window invisible.
WM_InvalidateArea()	Invalidates a certain section of the display.
WM_InvalidateRect()	Invalidates a part of a window.
WM_InvalidateWindow()	Invalidates a window.
WM_IsCompletelyCovered()	Checks if a window is completely covered or not.
WM_IsCompletelyVisible()	Checks if a window is completely visible or not.
WM_IsEnabled()	Returns if a window is enabled or not.
WM_IsVisible()	Returns if a window is visible or not.
WM_IsWindow()	Determine whether a specified handle is a valid window handle.
WM_MakeModal()	Changes the window to a 'modal' window.
WM_MoveChildTo()	Sets the position of a window in window coordinates.
WM_MoveTo()	Sets the position of a window in desktop coordinates.
WM_MoveWindow()	Moves a window to another position.
WM_NotifyParent()	Sends a WM_NOTIFY_PARENT message to the parent of the given window.
WM_Paint()	Draws or redraws a window immediately.
WM_PaintWindowAndDescs()	Draws a given window and all descendant windows immediately.
WM_ReleaseCapture()	Stops capturing mouse- and touch screen-input.
WM_ResizeWindow()	Changes the size of the given window.
WM_Screen2hWin()	Returns the window which lies at the specified position.
WM_Screen2hWinEx()	Returns the window which lies at the specified position using a window handle to stop at.
WM_SelectWindow()	Selects a window to be used for drawing operations.
WM_SendMessage()	Sends a message to a window.
WM_SendMessageNoPara()	Sends a message without parameters to a window.
WM_SendToParent()	Sends the given message to the parent window of the given window.
WM_SetCallback()	Sets the callback routine for a window.
WM_SetCapture()	Routes all PID-messages to the given window.
WM_SetCaptureMove()	Moves a window according to the current PID state.
WM_SetCreateFlags()	Sets the flags to be used by default when creating new windows.
WM_SetDesktopColor()	Sets desktop window color.
WM_SetDesktopColorEx()	Sets desktop window color of the given desktop.
WM_SetFocus()	Sets input focus to a specified window.
WM_SetHasTrans()	Sets the `has transparency` flag.
WM_SetId()	Sends a `WM_SET_ID` message to the given window.
WM_SetModalLayer()	Sets the layer to be used as modal layer.
WM_SetpfPollPID()	Sets a function to be called by the WM for polling the PID.
WM_SetSize()	Sets the new size of a window.
WM_SetWindowPos()	Sets size and position of a window.
WM_SetXSize()	Sets the new X-size of a window.
WM_SetYSize()	Sets the new Y-size of a window.
WM_SetStayOnTop()	Sets the `stay on top` flag.
WM_SetTransState()	Sets or clears the `WM_CF_HASTRANS` and `WM_CF_CONST_OUTLINE` flags.
WM_SetUserClipRect()	Reduces the clipping area temporarily.
WM_SetUserData()	Sets the user data of the given window.

Table 18.15: Window Manager API list

Routine	Description
WM_ShowWindow()	Makes a window visible.
WM_Update()	Draws the invalid part of the given window.
WM_UpdateWindowAndDescs()	Draws the invalid part of a given window and the invalid part of all descendant windows.
WM_ValidateRect()	Validates parts of a window.
WM_ValidateWindow()	Validates a window.
Motion support	
WM_MOTION_Enable()	Enables motion support of the WM.
WM_MOTION_SetDeceleration()	Sets the deceleration for the current movement.
WM_MOTION_SetDefaultPeriod()	Sets the default period for movements.
WM_MOTION_SetMotion()	Sets speed and deceleration for the desired movement.
WM_MOTION_SetMoveable()	Sets movability flags for the given window.
WM_MOTION_SetMovement()	Sets speed and distance for the desired movement.
WM_MOTION_SetSpeed()	Sets the speed for the desired movement.
ToolTip related functions	
WM_TOOLTIP_AddTool()	Adds a tool to an existing ToolTip object.
WM_TOOLTIP_Create()	Creates a ToolTip.
WM_TOOLTIP_Delete()	Deletes the given ToolTip.
WM_TOOLTIP_SetDefaultFont()	Sets the default font to be used for drawing ToolTip windows.
WM_TOOLTIP_SetDefaultColor()	Sets the default colors to be used for drawing ToolTip windows.
WM_TOOLTIP_SetDefaultPeriod()	Sets the default timing periods to be used for ToolTips.
Multiple Buffering support	
WM_MULTIBUF_Enable()	Enables or disables the automatic use of multiple buffers by the Window Manager.
Memory Device support (optional)	
WM_DisableMemdev()	Disables usage of Memory Devices for redrawing.
WM_EnableMemdev()	Enables usage of Memory Devices for redrawing.
Timer related	
WM_CreateTimer()	Creates a timer which sends a WM_TIMER message to a window.
WM_DeleteTimer()	Deletes a timer.
WM_GetTimerId()	Gets the Id of the given timer.
WM_RestartTimer()	Restarts a timer.
Widget related functions	
WM_GetClientWindow()	Returns the handle of the client window.
WM_GetId()	Returns the ID of a widget.
WM_GetInsideRect()	Returns the size of the active window less the border.
WM_GetInsideRectEx()	Returns the size of a window less the border.
WM_GetScrollBarH()	Returns the handle of an attached horizontal scroll bar.
WM_GetScrollBarV()	Returns the handle of an attached vertical scroll bar.
WM_GetScrollPosH()	Returns the horizontal scroll position of a window.
WM_GetScrollPosV()	Returns the vertical scroll position of a window.
WM_GetScrollState()	Gets the state of a SCROLLBAR widget.
WM_SetScrollPosH()	Sets the horizontal scroll position of a window.
WM_SetScrollPosV()	Sets the vertical scroll position of a window.
WM_SetScrollState()	Sets the state of a SCROLLBAR widget.

Table 18.15: Window Manager API list

18.7.1 Using the WM API functions

Many of the WM functions have window handles as parameters. Observe the following rules when using handles:

- Window handles can be 0. In this case functions usually return immediately. Functions which do not follow this rule are described accordingly.
- If a window handle is != 0, it should be a valid handle. The WM does not check if the given handle is valid. If an invalid handle is given to a function it fails or may even cause the application to crash.

18.7.2 WM API: Basic functions

WM_Activate()

Description

Activates the Window Manager.

Prototype

```
void WM_Activate(void);
```

Additional information

The WM is activated by default after initialization. This function only needs to be called if there has been a previous call of `WM_Deactivate()`.

WM_AttachWindow()

Description

The given window will be detached from its parent window and attached to the new parent window. The new origin in window coordinates of the new parent window will be the same as the old origin in window coordinates of the old parent window.

Prototype

```
void WM_AttachWindow(WM_HWIN hWin, WM_HWIN hParent);
```

Parameter	Description
hWin	Window handle.
hWinParent	Window handle of the new parent.

Table 18.16: WM_AttachWindow() parameter list

Additional information

If the given window handle is 0 or both handles are the same the function returns immediately.
If only the given parent window handle is 0 the function detaches the given window and returns; the window remains unattached.

WM_AttachWindowAt()

Description

The given window will be detached from its parent window and attached to the new parent window. The given position will be used to set the origin of the window in window coordinates of the parent window.

Prototype

```
void WM_AttachWindowAt(WM_HWIN hWin, WM_HWIN hParent, int x, int y);
```

Parameter	Description
hWin	Window handle.
hWinParent	Window handle of the new parent.
x	X position of the window in window coordinates of the parent window.
y	Y position of the window in window coordinates of the parent window.

Table 18.17: WM_AttachWindowAt() parameter list

Additional information

If the given window handle is 0 or both handles are the same the function returns immediately.
If only the given parent window handle is 0 the function detaches the given window, moves it to the new position and returns; the window remains unattached.

WM_BringToBottom()

Description

Places a specified window underneath its siblings.

Prototype

```
void WM_BringToBottom(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.18: WM_BringToBottom() parameter list

Additional information

The window will be placed underneath all other sibling windows, but will remain in front of its parent.

WM_BringToTop()

Description

Places a specified window on top of its siblings.

Prototype

```
void WM_BringToTop(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.19: WM_BringToTop() parameter list

Additional information

The window will be placed on top of all other sibling windows and its parent.

WM_BroadcastMessage()

Description

Sends the given message to all existing windows.

Prototype

```
int WM_BroadcastMessage(WM_MESSAGE * pMsg);
```

Parameter	Description
pMsg	Pointer to the message structure to be sent.

Table 18.20: WM_BroadcastMessage() parameter list

Additional information

A window should not delete itself or a parent window in reaction of a broadcasted message.

WM_ClrHasTrans()

Description

Clears the `has transparency` flag (sets it to 0).

Prototype

```
void WM_ClrHasTrans(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.21: WM_ClrHasTrans() parameter list

Additional information

When set, this flag tells the Window Manager that a window contains sections which are not redrawn and will therefore be transparent. The WM then knows that the background needs to be redrawn prior to redrawing the window in order to make sure the transparent sections are restored correctly.
When the flag is cleared with `WM_ClrHasTrans()`, the WM will not automatically redraw the background before redrawing the window.

WM_CreateWindow()

Description

Creates a window of a specified size at a specified location.

Prototype

```
WM_HWIN WM_CreateWindow(int x0,             int           y0,
                        int width,          int           height,
                        U32 Style,          WM_CALLBACK * cb,
                        int NumExtraBytes);
```

Parameter	Description
x0	Upper left X-position in desktop coordinates.
y0	Upper left Y-position in desktop coordinates.
width	X-size of window.
height	Y-size of window.
Style	Window create flags, listed below.
cb	Pointer to callback routine, or NULL if no callback used.
NumExtraBytes	Number of extra bytes to be allocated, normally 0.

Table 18.22: WM_CreateWindow() parameter list

Permitted values for parameter Style (OR-combinable)	
WM_CF_ANCHOR_BOTTOM	Anchors the bottom edge of the new window relative to the bottom edge of the parent window. If the position of the parent windows bottom edge will be adjusted due to a size change, the position of new window will also be adjusted.
WM_CF_ANCHOR_LEFT	Anchors the left edge of the new window relative to the left edge of the parent window (default). If the position of the parent windows left edge will be adjusted due to a size change, the position of new window will also be adjusted.
WM_CF_ANCHOR_RIGHT	Anchors the right edge of the new window relative to the right edge of the parent window. If the position of the parent windows right edge will be adjusted due to a size change, the position of new window will also be adjusted.
WM_CF_ANCHOR_TOP	Anchors the top edge of the new window relative to the top edge of the parent window (default). If the position of the parent windows top edge will be adjusted due to a size change, the position of new window will also be adjusted.
WM_CF_BGND	Put window in background after creation.

Permitted values for parameter `Style` (`OR`-combinable)	
`WM_CF_CONST_OUTLINE`	This flag is an optimization for transparent windows. It gives the Window Manager a chance to optimize redrawing and invalidation of transparent windows. A transparent window is normally redrawn as part of the background, which is less efficient than redrawing the window separately. However, this flag may NOT be used if the window has semi transparency (alpha blending / antialiasing with background) or the outline (the shape) changes with the window's states. To set this flag after the window has been created the function `WM_SetTransState()` should be used.
`WM_CF_FGND`	Put window in foreground after creation (default).
`WM_CF_HASTRANS`	Has transparency flag. Must be defined for windows whose client area is not entirely filled. To set this flag after the window has been created the function `WM_SetTransState()` should be used.
`WM_CF_HIDE`	Hide window after creation (default).
`WM_CF_LATE_CLIP`	This flag can be used to tell the WM that the clipping should be done in the drawing routines (late clipping). The default behavior of the WM is early clipping. That means that the clipping rectangle will be calculated before a `WM_PAINT` message will be sent to a window. In dependence of other existing windows it might be necessary to send more than one `WM_PAINT` message to a window. If using `WM_CF_LATE_CLIP` the WM makes sure only one message will be sent to an invalid window and the clipping will be done by the drawing routines. The `Sample` folder of emWin contains the example `WM_LateClipping.c` to show the effect.
`WM_CF_MEMDEV`	Automatically use a Memory Device for drawing. This will avoid flickering and also improve the output speed in most cases, as clipping is simplified. The Window Manager creates a Memory Device for the current window according to the configured color depth and window size. The Memory Device is deleted immediately after the drawing process was finished. In order to draw images into a remaining Memory Device the IMAGE widget can be used with the creation flag `IMAGE_CF_MEMDEV`. Details can be found in the section "IMAGE: Image widget" on page 618. Note that the Memory Device package is required (and needs to be enabled in the configuration) in order to be able to use this flag. If Memory Devices are not enabled, this flag is ignored.
`WM_CF_MEMDEV_ON_REDRAW`	Equals `WM_CF_MEMDEV` with the difference that the according window is drawn the first time without using a Memory Device. The WM will automatically use a Memory Device for redrawing. This flag can be used as a replacement of `WM_CF_MEMDEV`. It typically accelerates the initial rendering of the window, but maintains the advantage of flicker free updates.
`WM_CF_SHOW`	Show window after creation.
`WM_CF_STAYONTOP`	Make sure window stays on top of all siblings created without this flag.

Return value

Handle to the created window.

Additional information

Several create flags can be combined by using the (`OR`) operator.
Negative-position coordinates may be used.

Examples

Creates a window with callback:

```
hWin2 = WM_CreateWindow(100, 10, 180, 100, WM_CF_SHOW, &_cbWin, 0);
```

Creates a window without callback:

```
hWin2 = WM_CreateWindow(100, 10, 180, 100,WM_CF_SHOW, NULL, 0);
```

WM_CreateWindowAsChild()

Description

Creates a window as a child window.

Prototype

```
WM_HWIN WM_CreateWindowAsChild(int           x0,          int y0,
                               int           width,       int height,
                               WM_HWIN       hWinParent,  U8  Style,
                               WM_CALLBACK * cb,
                               int           NumExtraBytes);
```

Parameter	Description
x0	Upper left X-position in window coordinates of the parent window.
y0	Upper left Y-position in window coordinates of the parent window.
width	X-size of window. If 0, X-size of client area of parent window.
height	Y-size of window. If 0, Y-size of client area of parent window.
hWinParent	Handle of parent window.
Style	Window create flags (see `WM_CreateWindow()`).
cb	Pointer to callback routine, or NULL if no callback used.
NumExtraBytes	Number of extra bytes to be allocated, normally 0.

Table 18.23: WM_CreateWindowAsChild() parameter list

Return value

Handle to the created window.

Additional information

If the `hWinParent` parameter is set to 0, the background window is used as parent.
A child window is placed on top of its parent and any previous siblings by default, so that if their Z-positions are not changed, the "youngest" window will always be topmost.
The Z-positions of siblings may be changed, although they will always remain on top of their parent regardless of their order.

WM_Deactivate()

Description

Deactivates the Window Manager.

Prototype

```
void WM_Deactivate(void);
```

Additional information

After calling this function, the clip area is set to the complete LCD area and the WM will not execute window callback functions.

WM_DefaultProc()

Description

Default message handler.

Prototype

```
void WM_DefaultProc(WM_MESSAGE * pMsg);
```

Parameter	Description
pMsg	Pointer to message.

Table 18.24: WM_DefaultProc() parameter list

Additional information

Use this function to handle unprocessed messages as in the following example:

```
static WM_RESULT cbBackgroundWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_Clear();
  default:
    WM_DefaultProc(pMsg);
  }
}
```

WM_DeleteWindow()

Description

Deletes a specified window.

Prototype

```
void WM_DeleteWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.25: WM_DeleteWindow() parameter list

Additional information

Before the window is deleted, it receives a `WM_DELETE` message. This message is typically used to delete any objects (widgets) it uses and to free memory dynamically allocated by the window.
If the specified window has any existing child windows, these are automatically deleted before the window itself is deleted. Child windows therefore do not need to be separately deleted.
Before the window will be deleted it sends a WM_NOTIFICATION_CHILD_DELETED message to its parent window.

WM_DetachWindow()

Description

Detaches a window from its parent window. Detached windows will not be redrawn by the Window Manager.

Prototype

```
void WM_DetachWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.26: WM_DetachWindow() parameter list

WM_DisableWindow()

Description

Disables the specified window. The WM does not pass user input messages (touch, mouse, joystick, ...) to a disabled window.

Prototype

```
void WM_DisableWindow(WM_Handle hObj);
```

Parameter	Description
hObj	Handle of widget.

Table 18.27: WM_DisableWindow() parameter list

Additional information

A widget that is disabled will typically appear gray, and will not accept input from the user. However, the actual appearance may vary (depends on widget/configuration settings, etc.).
A disabled window will not receive the following messages: `WM_TOUCH`, `WM_TOUCH_CHILD`, `WM_PID_STATE_CHANGED` and `WM_MOUSEOVER`.

WM_EnableWindow()

Description

Sets the specified window to enabled state. An enabled window receives pointer input device (PID) messages (touch, mouse, joystick, ...) from the WM.

Prototype

```
void WM_EnableWindow(WM_Handle hObj);
```

Parameter	Description
hObj	Handle of window.

Table 18.28: WM_EnableWindow() parameter list

Additional information

This is the default setting for any widget.

WM_Exec()

Description

Redraws invalid windows by executing callback functions (all jobs).

Prototype

```
int WM_Exec(void);
```

Return value

0 if there were no jobs performed.
1 if a job was performed.

Additional information

This function will automatically call `WM_Exec1()` repeatedly until it has completed all jobs -- essentially until a 0 value is returned.
It is recommended to call the function `GUI_Exec()` instead.
Normally this function does not need to be called by the user application. It is called automatically by `GUI_Delay()`. If you are using a multitasking system, we recommend executing this function by a separate task as seen below:

```
void ExecIdleTask(void) {
  while(1) {
    WM_Exec();
```



```
    }
}
```

WM_Exec1()

Description

Redraws an invalid window by executing one callback function (one job only).

Prototype

```
int WM_Exec1(void);
```

Return value

0 if there were no jobs performed.
1 if a job was performed.

Additional information

This routine may be called repeatedly until 0 is returned, which means all jobs have been completed.
It is recommended to call the function `GUI_Exec1()` instead.
This function is called automatically by `WM_Exec()`.

WM_ForEachDesc()

Description

Iterates over all descendants of the given window. A descendant of a window is a child window or a grand child window or a child of a grand child or ....

Prototype

```
void WM_ForEachDesc(WM_HWIN hWin, WM_tfForEach * pcb, void * pData);
```

Parameter	Description
hWin	Window handle.
pcb	Pointer to callback function to be called by `WM_ForEachDesc`.
pData	User data to be passed to the callback function.

Table 18.29: WM_ForEachDesc() parameter list

Additional information

This function calls the callback function given by the pointer `pcb` for each descendant of the given window. The parameter `pData` will be passed to the user function and can be used to point to user defined data.

Prototype of callback function

```
void CallbackFunction(WM_HWIN hWin, void * pData);
```

Example

The following example shows how the function can be used. It creates 3 windows, the first as a child window of the desktop, the second as a child window of the first window and the third as a child window of the second window. After creating the window it uses `WM_ForEachDesc()` to move each window within its parent window:

```
static void _cbWin(WM_MESSAGE * pMsg) {
  GUI_COLOR Color;
  switch (pMsg->MsgId) {
  case WM_PAINT:
    WM_GetUserData(pMsg->hWin, &Color, 4);
    GUI_SetBkColor(Color);
    GUI_Clear();
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}
```



```
static void _cbDoSomething(WM_HWIN hWin, void * p) {
  int Value = *(int *)p;
  WM_MoveWindow(hWin, Value, Value);
}

void MainTask(void) {
  WM_HWIN hWin_1, hWin_2, hWin_3;
  int Value = 10;
  GUI_COLOR aColor[] = {GUI_RED, GUI_GREEN, GUI_BLUE};
  GUI_Init();
  WM_SetDesktopColor(GUI_BLACK);
  hWin_1 = WM_CreateWindow( 10, 10, 100, 100, WM_CF_SHOW, _cbWin, 4);
  hWin_2 = WM_CreateWindowAsChild(10, 10, 80, 80, hWin_1, WM_CF_SHOW, _cbWin, 4);
  hWin_3 = WM_CreateWindowAsChild(10, 10, 60, 60, hWin_2, WM_CF_SHOW, _cbWin, 4);
  WM_SetUserData(hWin_1, &aColor[0], 4);
  WM_SetUserData(hWin_2, &aColor[1], 4);
  WM_SetUserData(hWin_3, &aColor[2], 4);
  while(1) {
    WM_ForEachDesc(WM_HBKWIN, _cbDoSomething, (void *)&Value);
    Value *= -1;
    GUI_Delay(500);
  }
}
```

WM_GetCallback()

Description

Returns a pointer to the callback function of the given window

Prototype

`WM_CALLBACK * WM_GetCallback(WM_HWIN hWin);`

Parameter	Description
hWin	Window handle.

Table 18.30: WM_GetCallback() parameter list

Return value

Pointer of type `WM_CALLBACK` which points to the callback function of the given window. If the window has no callback function, `NULL` is returned.

WM_GetActiveWindow()

Description

Returns the handle of the active window used for drawing operations.

Prototype

`WM_HWIN WM_GetActiveWindow(void);`

Additional information

This function should be used only when the message `WM_PAINT` is processed in a window callback function.

Return value

The handle of the active window.

WM_GetClientRect()

Description

Returns the coordinates of the client area in the active window in window coordinates. That means x0 and y0 of the GUI_RECT structure will be 0, x1 and y1 corresponds to the size - 1.

Prototype

```
void WM_GetClientRect(GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to a GUI_RECT structure.

Table 18.31: WM_GetClientRect() parameter list

WM_GetClientRectEx()

Description

Returns the coordinates of the client area of a window in window coordinates. That means x0 and y0 of the GUI_RECT structure will be 0, x1 and y1 corresponds to the size - 1.

Prototype

```
void WM_GetClientRectEx(WM_HWIN hWin, GUI_RECT * pRect);
```

Parameter	Description
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure.

Table 18.32: WM_GetClientRectEx() parameter list

WM_GetDesktopWindow()

Description

Returns the handle of the desktop window.

Prototype

```
WM_HWIN WM_GetDesktopWindow(void);
```

Return value

The handle of the desktop window.

Additional information

The desktop window is always the bottommost window and any further created windows are its descendants.

WM_GetDesktopWindowEx()

Description

Returns the handle of the specified desktop window when working in a multi layer environment.

Prototype

```
WM_HWIN WM_GetDesktopWindowEx(unsigned int LayerIndex);
```

Parameter	Description
LayerIndex	Index of layer

Table 18.33: WM_GetDesktopWindowEx() parameter list

Return value

The handle of the specified desktop window.

WM_GetDialogItem()

Description

Returns the window handle of a dialog box item (widget).

Prototype

```
WM_HWIN WM_GetDialogItem(WM_HWIN hDialog, int Id);
```

Parameter	Description
hDialog	Handle of dialog box.
Id	Window Id of the widget.

Table 18.34: WM_GetDialogItem() parameter list

Return value

The window handle of the widget.

Additional information

This function is always used when creating dialog boxes, since the window Id of a widget used in a dialog must be converted to its handle before it can be used.

WM_GetFirstChild()

Description

Returns the handle of a specified window's first child window.

Prototype

```
void WM_GetFirstChild(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.35: WM_GetFirstChild() parameter list

Return value

Handle of the window's first child window; 0 if no child window exists.

Additional information

A window's first child window is the first child created to that particular parent. If the Z-positions of the windows have not been changed, it will be the window directly on top of the specified parent.

WM_GetFocussedWindow()

Description

Returns the handle of the window with the input focus.

Prototype

```
WM_HWIN WM_GetFocussedWindow(void);
```

Return value

Handle of the window with the input focus or 0 if no window has the input focus.

WM_GetHasTrans()

Description

Returns the current value of the `has transparency` flag.

Prototype

```
U8 WM_GetHasTrans(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.36: WM_GetHasTrans() parameter list

Return value

0: no transparency
1: window has transparency

Additional information

When set, this flag tells the Window Manager that a window contains sections which are not redrawn and will therefore be transparent. The WM then knows that the background needs to be redrawn prior to redrawing the window in order to make sure the transparent sections are restored correctly.

WM_GetInvalidRect()

Description

Returns the invalid rectangle of a window in desktop coordinates.

Prototype

```
int WM_GetInvalidRect(WM_HWIN hWin, GUI_RECT * pRect);
```

Parameter	Description
hWin	Window handle.
pRect	Pointer to a GUI_RECT-structure for storing the invalid rectangle.

Table 18.37: WM_GetInvalidRect() parameter list

Return value

0 if nothing is invalid, otherwise 1.

WM_GetModalLayer()

Description

Returns the current modal layer. Per default there does not exist a modal layer. In that case the function returns -1.

Prototype

```
int WM_GetModalLayer(void);
```

Return value

>= 0: Index of current modal layer.
-1: No modal layer is used.

Additional Information

Additional information can be found in the description of "WM_SetModalLayer()" on page 428.

WM_GetNextSibling()

Description

Returns the handle of a specified window's next sibling.

Prototype

```
void WM_GetNextSibling(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.38: WM_GetNextSibling() parameter list

Return value

Handle of the window's next sibling; 0 if none exists.

Additional information

A window's next sibling is the next child window that was created relative to the same parent. If the Z-positions of the windows have not been changed, it will be the window directly on top of the one specified.

WM_GetOrgX()

WM_GetOrgY()

Description

Returns the X- or Y-position (respectively) of the origin of the active window in desktop coordinates.

Prototypes

```
int WM_GetOrgX(void);
int WM_GetOrgY(void);
```

Return value

X- or Y-position of the origin of the active window in desktop coordinates.

WM_GetParent()

Description

Returns the handle of a specified window's parent window.

Prototype

```
void WM_GetParent(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.39: WM_GetParent() parameter list

Return value

Handle of the window's parent window; 0 if none exists.

Additional information

The only case in which no parent window exists is if the handle of the desktop window is used as parameter.

WM_GetPrevSibling()

Description

Returns the handle of a specified window's previous sibling.

Prototype

```
void WM_GetPrevSibling(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.40: WM_GetPrevSibling() parameter list

Return value

Handle of the window's previous sibling; 0 if none exists.

Additional information

A window's previous sibling is the previous child window that was created relative to the same parent. If the Z-positions of the windows have not been changed, it will be the window directly below of the one specified.

WM_GetStayOnTop()

Description

Returns the current value of the `stay on top` flag.

Prototype

```
int WM_GetStayOnTop(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.41: WM_GetStayOnTop() parameter list

Return value

0: stay on top flag not set
1: stay on top flag set

WM_GetUserData()

Description

Retrieves the data set with `WM_SetUserData()`.

Prototype

```
int WM_GetUserData(WM_HWIN hWin, void * pDest, int SizeOfBuffer);
```

Parameter	Description
hWin	Window handle.
pDest	Pointer to buffer.
SizeOfBuffer	SIze of buffer.

Table 18.42: WM_GetUserData() parameter list

Return value

Number of bytes retrieved.

Additional information

The maximum number of bytes returned by this function is the number of `ExtraBytes` specified when creating the window.

WM_GetWindowOrgX()

WM_GetWindowOrgY()

Description

Returns the X- or Y-position (respectively) of the origin of the specified window in desktop coordinates.

Prototypes

```
int WM_GetWindowOrgX(WM_HWIN hWin);
int WM_GetWindowOrgY(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.43: WM_GetWindowOrgX / WM_GetWindowOrgY() parameter list

Return value

X- or Y-position of the client area in pixels.

WM_GetWindowRect()

Description

Returns the coordinates of the active window in desktop coordinates.

Prototype

```
void WM_GetWindowRect(GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to a GUI_RECT structure.

Table 18.44: WM_GetWindowRect() parameter list

WM_GetWindowRectEx()

Description

Returns the coordinates of a window in desktop coordinates.

Prototype

```
void WM_GetWindowRectEx(WM_HWIN hWin, GUI_RECT * pRect);
```

Parameter	Description
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure.

Table 18.45: WM_GetWindowRectEx() parameter list

Additional information

If the given window handle is 0 or the given pointer to the GUI_RECT structure is NULL the function returns immediately.

WM_GetWindowSizeX()

WM_GetWindowSizeY()

Description

Return the X- or Y-size (respectively) of a specified window.

Prototypes

```
int WM_GetWindowSizeX(WM_HWIN hWin);
int WM_GetWindowSizeY(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.46: WM_GetWindowSizeX / WM_GetWindowSizeY() parameter list

Return value

X- or Y-size of the window in pixels.
Defined as x1-x0+1 in horizontal direction, y1-y0+1 in vertical direction, where x0, x1, y0, y1 are the leftmost/rightmost/topmost/bottommost positions of the window.
If the given window handle is 0 the function returns the size of the desktop window.

WM_HasCaptured()

Description

Checks if the given window has captured PID input.

Prototype

```
int WM_HasCaptured(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.47: WM_HasCaptured() parameter list

Return value

1 if the given window has captured mouse- and touchscreen-input, 0 if not.

Additional information

If the given window handle is invalid or 0 the function returns a wrong result.

WM_HasFocus()

Description

Checks if the given window has the input focus.

Prototype

```
int WM_HasFocus(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.48: WM_HasFocus() parameter list

Return value

1 if the given window has the input focus, otherwise 0.

Additional information

If the given window handle is invalid or 0 the function returns a wrong result.

WM_HideWindow()

Description

Makes a specified window invisible.

Prototype

```
void WM_HideWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.49: WM_HideWindow() parameter list

Additional information

The window will not immediately appear "invisible" after calling this function. The invalid areas of other windows (areas which appear to lie "behind" the window which should be hidden) will be redrawn when executing `WM_Exec()`. If you need to hide (draw over) a window immediately, you should call `WM_Paint()` to redraw the other windows.

WM_InvalidateArea()

Description

Invalidates a specified, rectangular area of the display.

Prototype

```
void WM_InvalidateArea(GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to a GUI_RECT structure with desktop coordinates.

Table 18.50: WM_InvalidateArea() parameter list

Additional information

Calling this function will tell the WM that the specified area is not updated.
This function can be used to invalidate any windows or parts of windows that overlap or intersect the area. The coordinates of the GUI_RECT structure have to be in desktop coordinates.

WM_InvalidateRect()

Description

Invalidates a specfied, rectangular area of a window.

Prototype

```
void WM_InvalidateRect(WM_HWIN hWin, GUI_RECT * pRect);
```

Parameter	Description
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure with window coordinates of the parent window.

Table 18.51: WM_InvalidateRect() parameter list

Additional information

Calling this function will tell the WM that the specified area is not updated.
The next time the Window Manager is executed so the window is redrawn, the area will be redrawn as well. The `GUI_RECT` structure has to be filled with window coordinates.

WM_InvalidateWindow()

Description

Invalidates a specified window.

Prototype

```
void WM_InvalidateWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.52: WM_InvalidateWindow() parameter list

Additional information

Calling this function tells the WM that the specified window is not updated.

WM_IsCompletelyCovered()

Description

Checks if the given window is completely covered or not.

Prototype

```
char WM_IsCompletelyCovered(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.53: WM_IsCompletelyCovered() parameter list

Return value

1 if the given window is completely covered, otherwise 0.

Additional information

If the given window handle is invalid or 0 the function returns a wrong result.

WM_IsCompletelyVisible()

Description

Checks if the given window is completely visible or not.

Prototype

```
char WM_IsCompletelyVisible(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.54: WM_IsCompletelyVisible() parameter list

Return value

1 if the given window is completely visible, otherwise 0.

Additional information

If the given window handle is invalid or 0 the function returns a wrong result.

WM_IsEnabled()

Description

This function returns if a window is enabled or not.

Prototype

```
int WM_IsEnabled(WM_HWIN hObj);
```

Parameter	Description
hObj	Handle of window.

Table 18.55: WM_IsEnabled() parameter list

Return value

1 if the window is enabled, 0 if not.

Additional information

A widget that is disabled will typically appear gray, and will not accept input from the user. However, the actual appearance may vary (depends on widget/configuration settings, etc.)

WM_IsVisible()

Description

Determines whether or not a specified window is visible.

Prototype

```
int WM_IsVisible(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.56: WM_IsVisible() parameter list

Return value

0: Window is not visible
1: Window is visible

WM_IsWindow()

Description

Determines whether or not a specified handle is a valid window handle.

Prototype

```
void WM_IsWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.57: WM_IsWindow() parameter list

Return value

0: handle is not a valid window handle
1: handle is a valid window handle

Additional information

This function should be used only if absolutely necessary. The more windows exist the more time will be used to evaluate, if the given handle is a window.

WM_MakeModal()

Description

This function makes the window work in 'modal' mode. This means pointer device input will only be sent to the 'modal' window or a child window of it if the input position is within the rectangle of the modal window.

Prototype

```
void WM_MakeModal(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle. 0 removes the modal state from the current modal window.

Table 18.58: WM_MakeModal() parameter list

WM_MoveChildTo()

Description

Moves a specified window to a certain position.

Prototype

```
void WM_MoveChildTo(WM_HWIN hWin, int x, int y);
```

Parameter	Description
hWin	Window handle.
x	New X-position in window coordinates of the parent window.
y	New Y-position in window coordinates of the parent window.

Table 18.59: WM_MoveChildTo() parameter list

WM_MoveTo()

Description

Moves a specified window to a certain position.

Prototype

```
void WM_MoveTo(WM_HWIN hWin, int x, int y);
```

Parameter	Description
hWin	Window handle.
x	New X-position in desktop coordinates.
y	New Y-position in desktop coordinates.

Table 18.60: WM_MoveTo() parameter list

WM_MoveWindow()

Description

Moves a specified window by a certain distance.

Prototype

```
void WM_MoveWindow(WM_HWIN hWin, int dx, int dy);
```

Parameter	Description
hWin	Window handle.
dx	Horizontal distance to move.
dy	Vertical distance to move.

Table 18.61: WM_MoveWindow() parameter list

WM_NotifyParent()

Description

Sends a `WM_NOTIFY_PARENT` message to the given window.

Prototype

```
void WM_NotifyParent(WM_HWIN hWin, int Notification);
```

Parameter	Description
hWin	Window handle.
Notification	Value to send to the parent window.

Table 18.62: WM_NotifyParent() parameter list

Additional information

The `Notification`-parameter will be sent in the Data.v element of the message. The macro `WM_NOTIFICATION_USER` can be used for defining application defined messages:

```
#define NOTIFICATION_1 (WM_NOTIFICATION_USER + 0)
#define NOTIFICATION_2 (WM_NOTIFICATION_USER + 1)
```


WM_Paint()

Description

Draws or redraws a specified window immediately.

Prototype

```
void WM_Paint(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.63: WM_Paint() parameter list

Additional information

The window is redrawn reflecting all updates made since the last time it was drawn.

WM_PaintWindowAndDescs()

Description

Paints the given window and all its descendants.

Prototype

```
void WM_PaintWindowAndDescs(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.64: WM_PaintWindowAndDescs() parameter list

Additional information

The function draws the complete window regions by invalidating them before drawing.

WM_ReleaseCapture()

Description

Releases capturing of mouse- and touchscreen-input.

Prototype

```
void WM_ReleaseCapture(void);
```

Additional information

Use `WM_SetCapture()` to send all mouse- and touchscreen-input to a specific window.

WM_ResizeWindow()

Description

Changes the size of a specified window by adding (or subtracting) the given differences.

Prototype

```
void WM_ResizeWindow(WM_HWIN hWin, int XSize, int YSize);
```

Parameter	Description
hWin	Window handle.
dx	Difference in X.
dy	Difference in Y.

Table 18.65: WM_ResizeWindow() parameter list

WM_Screen2hWin()

Description

Returns the window which lies at the specified position.

Prototype

```
WM_HWIN WM_Screen2hWin(int x, int y);
```

Parameter	Description
x	x-coordinate
y	y-coordinate

Table 18.66: WM_Screen2hWin() parameter list

Return value

Handle to the found window.

WM_Screen2hWinEx()

Description

Returns the window which lies at the specified position.

Prototype

```
WM_HWIN WM_Screen2hWinEx(WM_HWIN hStop, int x, int y);
```

Parameter	Description
hStop	Handle of a descendant (low-level window) to stop at.
x	x-coordinate
y	y-coordinate

Table 18.67: WM_Screen2hWinEx() parameter list

Return value

Handle to the found window. If `hStop` was found the handle to it's parent window is returned.

WM_SelectWindow()

Description

Selects a window to be used for drawing operations. The selected window is also called the active window.

Prototype

```
WM_HWIN WM_SelectWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.68: WM_SelectWindow() parameter list

Return value

The previously selected window.

Additional information

This function should not be called within a paint function called by the Window Manager. If the Window Manager sends a WM_PAINT message the target window already has been selected.
When working with a multi layer configuration the function switches also to the layer of the top level parent window of the given window.

If the given window handle is 0 the function selects the first created window, normally the first desktop window.

Example

Sets a window with handle hWin2 to the active window, sets the background color, and then clears the window:

```
WM_SelectWindow(hWin2);
GUI_SetBkColor(0xFF00);
GUI_Clear();
```

WM_SendMessage()

Description

Sends a message to a specified window.

Prototype

```
void WM_SendMessage(WM_HWIN hWin, WM_MESSAGE * pMsg)
```

Parameter	Description
hWin	Window handle.
pMsg	Pointer to a WM_MESSAGE structure. See “Elements of structure WM_MESSAGE” on page 388.

Table 18.69: WM_SendMessage() parameter list

Additional information

This function can be also used to send custom messages as described in the section “Application-defined messages” on page 397.

WM_SendMessageNoPara()

Description

Sends a message without parameters to a specified window.

Prototype

```
void WM_SendMessageNoPara(WM_HWIN hWin, int MsgId)
```

Parameter	Description
hWin	Window handle.
MsgId	Id of message to be sent.

Table 18.70: WM_SendMessageNoPara() parameter list

Additional information

If only a message Id should be sent to a window this should be done with this function, because it does not need a pointer to a WM_MESSAGE structure. Note that the receiving window gets no further information except the message Id.
This function can be used to send application-defined messages. Details about application-defined messages can be found on page 397.

WM_SendToParent()

Description

Sends the given message to the parent window of the given window.

Prototype

```
void WM_SendToParent(WM_HWIN hWin, WM_MESSAGE * pMsg);
```

Parameter	Description
hWin	Window handle.
pMsg	Pointer to WM_MESSAGE-structure.

Table 18.71: WM_SendToParent() parameter list

WM_SetCallback()

Description

Sets a callback routine to be executed by the Window Manager.

Prototype

```
WM_CALLBACK * WM_SetCallback(WM_HWIN hWin, WM_CALLBACK * cb);
```

Parameter	Description
hWin	Window handle.
cb	Pointer to callback routine.

Table 18.72: WM_SetCallback() parameter list

Return value

Pointer to the previous callback routine.

Additional information

The given window will be invalidated. This makes sure the window will be redrawn.

WM_SetCapture()

Description

Routes all PID-messages to the given window.

Prototype

```
void WM_SetCapture(WM_HWIN hObj, int AutoRelease);
```

Parameter	Description
hWin	Window handle.
AutoRelease	1 if capturing should end when the user releases the touch.

Table 18.73: WM_SetCapture() parameter list

WM_SetCaptureMove()

Description

Moves a window according to the given PID state. This function is intended to be used in a window callback function. It should react to the message `WM_TOUCH` if the PID is in pressed state.

Prototype

```
void WM_SetCaptureMove(WM_HWIN hWin,      GUI_PID_STATE * pState,
                       int MinVisibility, int LimitTop);
```

Parameter	Description
hWin	Handle of the window which should be moved.

Table 18.74: WM_SetCaptureMove() parameter list

Parameter	Description
pState	Pointer to the PID state.
MinVisibility	Defines the minimum visibility of the parent window in pixels. The window will not be moved farther than the parent window reduced by the minimum visibility.
LimitTop	Defines a number of top pixel lines which can not be moved outside the parent rectangle. The bottom pixel lines which are excluded are allowed to be moved outside the parent rectangle.

Table 18.74: WM_SetCaptureMove() parameter list

Example

The following example application shows a callback function of a window which is moved using `WM_SetCaptureMove()`:

```
static void _cbWin(WM_MESSAGE * pMsg) {
  const GUI_PID_STATE * pState;
  WM_HWIN hWin;

  hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
  case WM_TOUCH:
    pState = (const GUI_PID_STATE *)pMsg->Data.p;
    if (pState) {
      if (pState->Pressed) {
        WM_SetCaptureMove(hWin, pState, 0, 0);
      }
    }
    break;
  case WM_PAINT:
    GUI_SetBkColor(GUI_DARKBLUE);
    GUI_Clear();
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  WM_SetDesktopColor(GUI_DARKGREEN);
  hWin = WM_CreateWindow(10, 10, 200, 100, WM_CF_SHOW, _cbWin, 0);
  while (1) {
    GUI_Delay(1);
  }
}
```

WM_SetCreateFlags()

Description

Sets the flags to be used as default when creating a new window.

Prototype

```
U8 WM_SetCreateFlags(U8 Flags);
```

Parameter	Description
Flags	Window create flags (see `WM_CreateWindow()`).

Table 18.75: WM_SetCreateFlags() parameter list

Return value

Former value of this parameter.

Additional information

The flags specified here are binary `ORed` with the flags specified in the `WM_CreateWindow()` and `WM_CreateWindowAsChild()` routines.
The flag `WM_CF_MEMDEV` is frequently used to enable Memory Devices on all windows.
Setting create flags is permitted before GUI_Init() is called. This causes the background window to be also affected by the create flags.

Example

```
WM_SetCreateFlags(WM_CF_MEMDEV);  // Auto. use Memory Devices on all windows
```

WM_SetDesktopColor()

Description

Sets the color for the desktop window.

Prototype

```
GUI_COLOR WM_SetDesktopColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color for desktop window, 24-bit RGB value.

Table 18.76: WM_SetDesktopColor() parameter list

Return value

The previously selected desktop window color.

Additional information

The default setting for the desktop window is not to repaint itself. If this function is not called, the desktop window will not be redrawn at all; therefore other windows will remain visible even after they are deleted.
Once a color is specified with this function, the desktop window will repaint itself. In order to restore the default, call this function and specify `GUI_INVALID_COLOR`.

WM_SetDesktopColorEx()

Description

Sets the color for the desktop window in a multi layer environment.

Prototype

```
GUI_COLOR WM_SetDesktopColorEx(GUI_COLOR Color, unsigned int LayerIndex);
```

Parameter	Description
Color	Color for desktop window, 24-bit RGB value.
LayerIndex	Index of the layer.

Table 18.77: WM_SetDesktopColorEx() parameter list

Return value

The previously selected desktop window color.

Additional information

(see WM_SetDesktopColor).

WM_SetFocus()

Description

Sets the input focus to a specified window.

Prototype

```
void WM_SetFocus(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.78: WM_SetFocus() parameter list

Return value

0 if window accepted focus; value other than 0 if it could not.

Additional information

The window receives a WM_SET_FOCUS message which gives it the input focus. If for some reason the window could not accept the focus, nothing happens.

WM_SetHasTrans()

Description

Enables transparency for the given window.

Prototype

```
void WM_SetHasTrans(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.79: WM_SetHasTrans() parameter list

Additional information

Using this function causes the Window Manager to redraw the background of the given window in order to have the transparent parts updated before the actual window is drawn.

WM_SetId()

Description

This function sends a WM_SET_ID message to the given window.

Prototype

```
void WM_SetId(WM_HWIN hObj, int Id);
```

Parameter	Description
hObj	Window handle.
Id	Id to be sent to the window.

Table 18.80: WM_SetId() parameter list

Additional information

This function can be used to change the Id of a widget. It works with every widget. When using this function with an application defined window, the callback function of the window should handle the message. Otherwise it will be ignored.

WM_SetModalLayer()

Description

emWin supports one modal window on each layer per default. But sometimes it could make sense to have only one modal window. To be able to achieve that function could be used. Once a modal layer has been set only windows of that layer will receive input.

Prototype

```
int WM_SetModalLayer(int LayerIndex);
```

Parameter	Description
LayerIndex	Layer to be set as modal layer, -1 means no modal layer

Table 18.81: WM_SetModalLayer() parameter list

Return value

>= 0: Index of previous modal layer.
-1: No modal layer was used previously.
-2: Error.

WM_SetpfPollPID()

Description

Sets a function which will be called by the Window Manager in order to poll the pointer input device (touch-screen or mouse).

Prototype

```
WM_tfPollPID * WM_SetpfPollPID(WM_tfPollPID * pf);
```

Parameter	Description
pf	Pointer to a function of type WM_tfPollPID.

Table 18.82: WM_SetpfPollPID() parameter list

Additional information

The function type is defined as follows:

```
typedef void WM_tfPollPID(void);
```

Example

Example of a touch-screen handled as a device:

```
void ReadTouch(void) {
  // ...read touchscreen
}

WM_SetpfPollPID(ReadTouch);
```

WM_SetSize()

Description

Sets the new size of a window.

Prototype

```
void WM_SetSize(WM_HWIN hWin, int XSize, int YSize);
```

Parameter	Description
hWin	Window handle.
XSize	New size in X.
YSize	New size in Y.

Table 18.83: WM_SetSize() parameter list

WM_SetWindowPos()

Description

Sets the size and the position of a window.

Prototype

```
void WM_SetWindowPos(WM_HWIN hWin,
                     int     xPos,   int yPos,
```



```
                    int     xSize, int ySize);
```

Parameter	Description
hWin	Window handle.
xPos	New position in X in desktop coordinates.
yPos	New position in Y in desktop coordinates.
xSize	New size in X.
ySize	New size in Y.

Table 18.84: WM_SetWindowPos() parameter list

WM_SetXSize()

Description

Sets the new X-size of a window.

Prototype

```
void WM_SetXSize(WM_HWIN hWin, int XSize);
```

Parameter	Description
hWin	Window handle.
XSize	New size in X.

Table 18.85: WM_SetXSize() parameter list

WM_SetYSize()

Description

Sets the new Y-size of a window.

Prototype

```
void WM_SetYSize(WM_HWIN hWin, int YSize);
```

Parameter	Description
hWin	Window handle.
YSize	New size in Y.

Table 18.86: WM_SetYSize() parameter list

WM_SetStayOnTop()

Description

Sets the `stay on top` flag.

Prototype

```
void WM_SetStayOnTop(WM_HWIN hWin, int OnOff);
```

Parameter	Description
hWin	Window handle.
OnOff	See table below.

Table 18.87: WM_SetStayOnTop() parameter list

Permitted values for parameter OnOff	
0	`Stay on top` flag would be cleared.
1	`Stay on top` flag would be set.

WM_SetTransState()

Description

This function sets or clears the flags WM_CF_HASTRANS and WM_CF_CONST_OUTLINE of the given window.

Prototype

```
void WM_SetTransState(WM_HWIN hWin, unsigned State);
```

Parameter	Description
hWin	Window handle.
State	Combination of the flags WM_CF_HASTRANS and WM_CF_CONST_OUTLINE.

Table 18.88: WM_SetTransState() parameter list

Additional information

Details about the flags WM_CF_CONST_OUTLINE and WM_CF_HASTRANS can be found in the function description of "WM_CreateWindow()" on page 404.

WM_SetUserClipRect()

Description

Temporarily reduces the clip area of the current window to a specified rectangle.

Prototype

```
const GUI_RECT * WM_SetUserClipRect(const GUI_RECT * pRect);
```

Parameter	Description
pRect	Pointer to a GUI_RECT structure defining the clipping region in desktop coordinates.

Table 18.89: WM_SetUserClipRect() parameter list

Return value

Pointer to the previous clip rectangle.

Additional information

A NULL pointer can be passed in order to restore the default settings. The clip rectangle will automatically be reset by the WM when callbacks are used.
The specified rectangle must be relative to the current window. You cannot enlarge the clip rectangle beyond the current window rectangle.
Your application must ensure that the specified rectangle retains its value until it is no longer needed; that is, until a different clip rectangle is specified or until a NULL pointer is passed. This means that the rectangle structure passed as parameter should not be an auto variable (usually located on the stack) if the clip rectangle remains active until after the return. In this case, a static variable should be used.

Example

This example is taken from the drawing routine of a progress indicator. The progress indicator must write text on top of the progress bar, where the text color has to be different on the left and right parts of the bar. This means that half of a digit could be in one color, while the other half could be in a different color. The best way to do this is to temporarily reduce the clip area when drawing each part of the bar as shown below:


```
/* Draw left part of the bar */
    r.x0=0; r.x1=x1-1; r.y0=0; r.y1 = GUI_YMAX;
    WM_SetUserClipRect(&r);
    GUI_SetBkColor(pThis->ColorBar[0]);
    GUI_SetColor(pThis->ColorText[0]);
    GUI_Clear();
    GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
/* Draw right part of the bar */
    r.x0=r.x1; r.x1=GUI_XMAX;
    WM_SetUserClipRect(&r);
    GUI_SetBkColor(pThis->ColorBar[1]);
    GUI_SetColor(pThis->ColorText[1]);
    GUI_Clear();
    GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
```

Screenshot of progress bar

WM_SetUserData()

Description

Sets the extra data of a window. Memory for extra data is reserved with the parameter `NumExtraBytes` when creating a window.

Prototype

```
int WM_SetUserData(WM_HWIN hWin, void * pSrc, int NumBytes);
```

Parameter	Description
hWin	Window handle.
pSrc	Pointer to buffer.
NumBytes	Size of buffer.

Table 18.90: WM_SetUserData() parameter list

Return value

Number of bytes written.

Additional information

The maximum number of bytes used to store user data is the number of ExtraBytes specified when creating a window.

WM_ShowWindow()

Description

Makes a specified window visible.

Prototype

```
void WM_ShowWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.91: WM_ShowWindow() parameter list

Additional information

The window will not immediately be visible after calling this function. It will be redrawn when executing `WM_Exec()`. If you need to show (draw) the window immediately, you should call `WM_Paint()`.

WM_Update()

Description

Draws the invalid part of the specified window immediately.

Prototype

```
void WM_Update(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.92: WM_Update() parameter list

Additional information

After updating a window its complete region is marked as valid.

WM_UpdateWindowAndDescs()

Description

Paints the invalid part of the given window and the invalid part of all its descendants.

Prototype

```
void WM_UpdateWindowAndDescs(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.93: WM_UpdateWindowAndDescs() parameter list

Additional information

The function only draws the invalid window regions.

WM_ValidateRect()

Description

Validates a specfied, rectangular area of a window.

Prototype

```
void WM_ValidateRect(WM_HWIN hWin, GUI_RECT * pRect);
```

Parameter	Description
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure with window coordinates of the parent window.

Table 18.94: WM_ValidateRect() parameter list

Additional information

Calling this function will tell the WM that the specified area is updated.
Normally this function is called internally and does not need to be called by the user application. The coordinates of the GUI_RECT structure have to be in desktop coordinates.

WM_ValidateWindow()

Description

Validates a specified window.

Prototype

```
void WM_ValidateWindow(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.95: WM_ValidateWindow() parameter list

Additional information

Calling this function will tell the WM that the specified window is updated.
Normally this function is called internally and does not need to be called by the user application.

18.7.3 WM API: Motion support

WM_MOTION_Enable()

Description

Enables motion support for the WM. Needs to be called once at the beginning of the program.

Prototype

```
void WM_MOTION_Enable(int OnOff);
```

Parameter	Description
OnOff	1 for enabling motion support, 0 for disabling it.

Table 18.96: WM_MOTION_Enable() parameter list

WM_MOTION_SetDeceleration()

Description

Can be used to set the deceleration of the current moving operation.

Prototype

```
void WM_MOTION_SetDeceleration(WM_HWIN hWin, int Axis, I32 Deceleration);
```

Parameter	Description
hWin	Window handle.
Axis	See table below.
Deceleration	Deceleration in pixel / (s * s)

Table 18.97: WM_MOTION_SetDeceleration() parameter list

Permitted values for parameter Axis	
GUI_COORD_X	X axis should be used.
GUI_COORD_Y	Y axis should be used.

Additional information

Makes only sense if the given window is already moving.

WM_MOTION_SetDefaultPeriod()

Description

Sets the default value to be used for the duration of the deceleration phase after the PID has been released. If the window is already moving the window decelerates its motion until it stops. If the window is not moving but snapping is used the window moves within that period to the next raster position. If the window is already moving and snapping is used the window decelerates its motion until it stops to the nearest raster position given by the current speed.

Prototype

```
unsigned WM_MOTION_SetDefaultPeriod(unsigned Period);
```

Parameter	Description
Period	Period to be used.

Table 18.98: WM_MOTION_SetDefaultPeriod() parameter list

Return value

Previous default value of the period.

WM_MOTION_SetMotion()

Description

Starts a moving operation with the given speed and deceleration.

Prototype

```
void WM_MOTION_SetMotion(WM_HWIN hWin, int Axis, I32 Speed,
                         I32 Deceleration);
```

Parameter	Description
hWin	Window handle.
Axis	See table below.
Speed	Speed to be used.
Deceleration	Deceleration to be used.

Table 18.99: WM_MOTION_SetMotion() parameter list

Permitted values for parameter Axis	
GUI_COORD_X	X axis should be used.
GUI_COORD_Y	Y axis should be used.

Additional information

The moving operation then can be affected by further motion functions.

WM_MOTION_SetMoveable()

Description

Enables movability of the given window.

Prototype

```
void WM_MOTION_SetMoveable(WM_HWIN hWin, U32 Flags, int OnOff);
```

Parameter	Description
hWin	Window handle.
Flags	See table below.
OnOff	1 for enabling, 0 for disabling.

Table 18.100: WM_MOTION_SetMoveable() parameter list

Permitted values for parameter Flags	
WM_CF_MOTION_X	Enables / disables movability for the X axis.
WM_CF_MOTION_Y	Enables / disables movability for the Y axis.

Additional information

Motion support of a window can also be set with creation flags when creating the window or within the callback routine of the window. Details can be found in the section “Motion support” on page 383.

WM_MOTION_SetMovement()

Description

Starts a moving operation with the given speed for the given distance.

Prototype

```
void WM_MOTION_SetMovement(WM_HWIN hWin, int Axis, I32 Speed, I32 Dist);
```

Parameter	Description
hWin	Window handle.
Axis	See table below.
Speed	Speed in pixels / s to be used. Positive and negative values are supported.
Dist	Distance to be used. Needs to be a positive value.

Table 18.101: WM_MOTION_SetMovement() parameter list

Permitted values for parameter Axis	
GUI_COORD_X	X axis should be used.
GUI_COORD_Y	Y axis should be used.

Additional information

The moving operation stops automatically if the given distance is reached.

WM_MOTION_SetSpeed()

Description

Starts moving the given window with the given speed.

Prototype

```
void WM_MOTION_SetSpeed(WM_HWIN hWin, int Axis, I32 Speed);
```

Parameter	Description
hWin	Window handle.
Axis	See table below.
Speed	Speed in pixel / s to be used.

Table 18.102: WM_MOTION_SetSpeed() parameter list

Permitted values for parameter Axis	
GUI_COORD_X	X axis should be used.
GUI_COORD_Y	Y axis should be used.

18.7.4 WM API: ToolTip related functions

In addition to the introduction at the beginning of the chapter the following contains the detailed descriptions of the ToolTip related functions.

WM_TOOLTIP_AddTool()

Description

Adds a tool to an existing ToolTip object.

Prototype

```
int WM_TOOLTIP_AddTool(WM_TOOLTIP_HANDLE hToolTip, WM_HWIN hTool,
```



```
const char * pText);
```

Parameter	Description
hToolTip	Handle of ToolTip object.
hTool	Handle of tool window.
pText	Pointer to a string.

Table 18.103: WM_TOOLTIP_AddTool() parameter list

Return value

0 on success, !=0 on error.

Additional information

This function can be used for adding tools by passing the window Id and a string pointer. The given string is copied into the dynamic memory of emWin and does not need to remain valid.

WM_TOOLTIP_Create()

Description

Creates a ToolTip object for the given dialog.

Prototype

```
WM_TOOLTIP_HANDLE WM_TOOLTIP_Create(WM_HWIN hDlg,
                                    const TOOLTIP_INFO * pInfo,
                                    unsigned NumItems);
```

Parameter	Description
hDlg	Handle of the dialog containing the tools as child- or grand child windows.
pInfo	Pointer to an array of TOOLTIP_INFO structures. Can be NULL.
NumItems	Number of tools to be added.

Table 18.104: WM_TOOLTIP_Create() parameter list

Return value

Handle to the ToolTip object on success, 0 on failure.

Additional information

If one of the parameters pInfo or NumItems is 0 the function only creates the ToolTip object. It is the responsibility of the application to delete the object if it is no longer used.

WM_TOOLTIP_Delete()

Description

Deletes the given ToolTip object.

Prototype

```
void WM_TOOLTIP_Delete(WM_TOOLTIP_HANDLE hToolTip);
```

Parameter	Description
hToolTip	Handle of ToolTip object to be deleted.

Table 18.105: WM_TOOLTIP_Delete() parameter list

WM_TOOLTIP_SetDefaultColor()

Description

Sets the default colors to be used for drawing ToolTips.

Prototype

```
GUI_COLOR WM_TOOLTIP_SetDefaultColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Description
Index	See table below.
Color	Default color to be used.

Table 18.106: WM_TOOLTIP_SetDefaultColor() parameter list

Permitted values for parameter Index	
WM_TOOLTIP_CI_BK	Color to be used for the background.
WM_TOOLTIP_CI_FRAME	Color to be used for the thin frame.
WM_TOOLTIP_CI_TEXT	Color to be used for the text.

Return value

Previous used color.

WM_TOOLTIP_SetDefaultFont()

Description

Sets the font to be used for displaying the text of ToolTips.

Prototype

```
const GUI_FONT * WM_TOOLTIP_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Font to be used.

Table 18.107: WM_TOOLTIP_SetDefaultFont() parameter list

Return value

Previous default font used for ToolTips.

WM_TOOLTIP_SetDefaultPeriod()

Description

Sets the default periods to be used for showing ToolTips.

Prototype

```
unsigned WM_TOOLTIP_SetDefaultPeriod(unsigned Index, unsigned Period);
```

Parameter	Description
Index	See table below.
Period	Period to be used.

Table 18.108: WM_TOOLTIP_SetDefaultPeriod() parameter list

Permitted values for parameter Index	
WM_TOOLTIP_PI_FIRST	Period to be used the first time the PID is hovered over a tool. The ToolTip appears after the PID has not moved for at least this period. Default is 1000 ms.
WM_TOOLTIP_PI_SHOW	Period to be used for showing the ToolTip. The ToolTip disappears after the PID remains for at least this period without moving. Default is 5000 ms.
WM_TOOLTIP_PI_NEXT	Period to be used if the PID hovers over a tool of the same parent as before. The ToolTip appears after the PID is not moved for at least this period. Default is 50 ms.

Return value

Previous used value.

18.7.5 WM API: Multiple Buffering support

WM_MULTIBUF_Enable()

Description

This functions enables or disables the automatic use of Multiple Buffering by the Window Manager.

Prototype

```
int WM_MULTIBUF_Enable(int OnOff);
```

Parameter	Description
OnOff	1 to enable the automatic use of multiple buffers. 0 to disable the automatic use of multiple buffers.

Table 18.109: WM_MULTIBUF_Enable() parameter list

Additional information

Detailed information on how to use Multiple Buffering can be found in the chapter "Multiple Buffering" on page 935.

Return value

Previous state.

18.7.6 WM API: Memory Device support (optional)

When a Memory Device is used for redrawing a window, all drawing operations are automatically routed to a Memory Device context and are executed in memory. Only after all drawing operations have been carried out is the window redrawn on the LCD, reflecting all updates at once. The advantage of using Memory Devices is that any flickering effects (which normally occur when the screen is continuously updated as drawing operations are executed) are eliminated.
For more information on how Memory Devices operate, see the chapter "Memory Devices" on page 319.

WM_DisableMemdev()

Description

Disables the use of Memory Devices for redrawing a window.

Prototype

```
void WM_DisableMemdev(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.110: WM_DisableMemdev() parameter list

WM_EnableMemdev()

Description

Enables the use of Memory Devices for redrawing a window.

Prototype

```
void WM_EnableMemdev(WM_HWIN hWin);
```

Parameter	Description
hWin	Window handle.

Table 18.111: WM_EnableMemdev() parameter list

18.7.7 WM API: Timer related functions

WM_CreateTimer()

Description

Creates a timer which sends a message to the given window after the given time period has expired. The timer is associated to the given window.

Prototype

```
WM_HTIMER WM_CreateTimer(WM_HWIN hWin, int UserId, int Period, int Mode);
```

Parameter	Description
hWin	Handle of window to be informed.
UserId	User defined Id. Can be set to 0 if not using multiple timers for the same window.
Period	Time period after which the given window should receive a message.
Mode	(reserved for future use, should be 0)

Table 18.112: WM_CreateTimer() parameter list

Return value

Handle of the timer.

Additional information

The function creates a 'one shot timer' which sends a `WM_TIMER` message to the given window. After the timer period has expired the timer object remains valid and can be restarted using the function `WM_RestartTimer()` or deleted with `WM_DeleteTimer()`. Once a window is deleted the Window Manager automatically deletes all timers associated to the window.

Example

```
static void _cbWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_TIMER:
    /*
      ... do something ...
    */
    WM_RestartTimer(pMsg->Data.v, 1000);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

static void _DemoTimer(void) {
  WM_HWIN hWin;
  WM_HTIMER hTimer;
  hWin   = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, _cbWin, 0);
  hTimer = WM_CreateTimer(hWin, 0, 1000, 0);
  while (1) {
    GUI_Exec();
  }
}
```

WM_DeleteTimer()

Description

Deletes the given timer.

Prototype

```
void WM_DeleteTimer(WM_HTIMER hTimer);
```

Parameter	Description
hTimer	Handle of the timer to be deleted.

Table 18.113: WM_DeleteTimer() parameter list

Additional information

After a timer has expired the timer object remains valid and will not be deleted automatically. If it is not used anymore it should be deleted using this function.
Once a window is deleted the Window Manager automatically deletes all timers associated to the window.

WM_GetTimerId()

Description

Gets the Id of the given timer.

Prototype

```
int WM_GetTimerId(WM_HTIMER hTimer);
```

Parameter	Description
hTimer	Handle of the timer to be deleted.

Table 18.114: WM_GetTimerId() parameter list

Return value

The Id of the timer which was previously set within the function `WM_CreateTimer()`.

WM_RestartTimer()

Description

Restarts the given timer with the given period.

Prototype

```
void WM_RestartTimer(WM_HTIMER hTimer, int Period);
```

Parameter	Description
hTimer	Handle of the timer to be restarted.
Period	New period to be used.

Table 18.115: WM_RestartTimer() parameter list

Additional information

After the period has expired a `WM_TIMER` message will be sent to the window assigned to the timer. For details, refer to "WM_CreateTimer()" on page 440.

18.7.8 WM API: Widget related functions

WM_GetClientWindow()

Description

Returns the handle of the client window. The function sends a message to the active window to retrieve the handle of the client window. If the window does not handle the message the handle of the current window will be returned.

Prototype

`WM_HWIN WM_GetClientWindow(WM_HWIN hObj);`

Parameter	Description
`hWin`	Handle of widget.

Table 18.116: WM_GetClientWindow() parameter list

Return value

Handle of the client window.

Additional information

Use this function to retrieve the client window handle of a FRAMEWIN widget.

WM_GetId()

Description

Returns the ID of a specified widget window.

Prototype

`int WM_GetId(WM_HWIN hObj);`

Parameter	Description
`hObj`	Handle of widget.

Table 18.117: WM_GetId() parameter list

Return value

>0, ID of the widget which was specified at creation or set using `WM_SetId()`.
0 will be returned if the specified window is not a widget.

WM_GetInsideRect()

Description

Returns the coordinates of the client area of the active widget less the border size. The function sends a message to the active window to retrieve the inside rectangle. If the widget does not handle the message (that means the widget has no border) WM_GetClientRect will be used to calculate the rectangle. The result is given in window coordinates. That means x0 and y0 of the GUI_RECT structure corresponds to the border size in x and y, x1 and y1 corresponds to the size of the window less the border size - 1.

Prototype

`void WM_GetInsideRect(GUI_RECT * pRect);`

Parameter	Description
`pRect`	Pointer to a `GUI_RECT` structure.

Table 18.118: WM_GetInsideRect() parameter list

WM_GetInsideRectEx()

Description

Returns the coordinates of a window less the border size. For details, refer to “WM_GetInsideRect()” on page 442.

Prototype

```
void WM_GetInsideRectEx(WM_HWIN hObj, GUI_RECT * pRect);
```

Parameter	Description
hObj	Handle of widget.
pRect	Pointer to a GUI_RECT structure.

Table 18.119: WM_GetInsideRectEx() parameter list

WM_GetScrollBarH()

Description

If the given window has a horizontal scroll bar attached the function returns the handle of that scrollbar.

Prototype

```
WM_HWIN WM_GetScrollbarH(WM_HWIN hWin);
```

Parameter	Description
hWin	Handle of a window which has a horizontal SCROLLBAR attached.

Table 18.120: WM_GetScrollBar() parameter list

Return value

Handle of the horizontal SCROLLBAR widget
0, if no horizontal SCROLLBAR widget is attached.

Additional information

Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

WM_GetScrollBarV()

Description

If the given window has a vertical scroll bar attached the function returns the handle of that scrollbar.

Prototype

```
WM_HWIN WM_GetScrollbarV(WM_HWIN hWin);
```

Parameter	Description
hWin	Handle of a window which has a vertical SCROLLBAR attached.

Table 18.121: WM_GetScrollBar() parameter list

Return value

Handle of the horizontal SCROLLBAR widget
0, if no horizontal SCROLLBAR widget is attached.

Additional information

Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

WM_GetScrollPosH()

Description

Returns the horizontal scrolling position of a window.

Prototype

```
int WM_GetScrollPosH(WM_HWIN hWin);
```

Parameter	Description
hWin	Handle of a window which has a horizontal SCROLLBAR attached.

Table 18.122: WM_GetScrollPosH() parameter list

Return value

Position of the horizontal SCROLLBAR widget (0 < n)
0, if no horizontal SCROLLBAR widget is attached.

Additional information

Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

WM_GetScrollPosV()

Description

Returns the vertical scrolling position of a window.

Prototype

```
int WM_GetScrollPosV(WM_HWIN hWin);
```

Parameter	Description
hWin	Handle of a window which has a vertical SCROLLBAR attached.

Table 18.123: WM_GetScrollPosV() parameter list

Return value

Position of the horizontal SCROLLBAR widget (0 < n)
0, if no horizontal SCROLLBAR widget is attached.

Additional information

Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

WM_GetScrollState()

Description

Fills a data structure with information of the current state of a specified SCROLLBAR widget.

Prototype

```
void WM_GetScrollState(WM_HWIN hObj, WM_SCROLL_STATE * pScrollState);
```

Parameter	Description
hObj	Handle of scroll bar widget.
pScrollState	Pointer to a data structure of type WM_SCROLL_STATE.

Table 18.124: WM_GetScrollState() parameter list

Additional information

This function does not return since the state of a scroll bar is defined by more than one value.
It has no effect on other types of widgets or windows.
Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

Elements of structure WM_SCROLL_STATE

Data type	Element	Description
int	NumItems	Number of items.
int	v	Current value.
int	PageSize	Number of items visible on one page.

Table 18.125: WM_SCROLL_STATE element list

WM_SetScrollPosH()

Description

Sets the horizontal scrolling position of a window.

Prototype

```
void WM_SetScrollPosH(WM_HWIN hWin, unsigned ScrollPos);
```

Parameter	Description
hWin	Handle of a window which has a horizontal SCROLLBAR attached.
ScrollPos	New scroll position of the scroll bar.

Table 18.126: WM_SetScrollPosH() parameter list

Additional information

Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

WM_SetScrollPosV()

Description

Sets the vertical scrolling position of a window.

Prototype

```
void WM_SetScrollPosV(WM_HWIN hWin, unsigned ScrollPos);
```

Parameter	Description
hWin	Handle of a window which has a vertical SCROLLBAR attached.
ScrollPos	New scroll position of the scroll bar.

Table 18.127: WM_SetScrollPosV() parameter list

Additional information

Additional information can be found in “SCROLLBAR: Scroll bar widget” on page 765.

WM_SetScrollState()

Description

Sets the state of a specified SCROLLBAR widget.

Prototype

```
void WM_SetScrollState(WM_HWIN hObj, const WM_SCROLL_STATE * pState);
```

Parameter	Description
hObj	Handle of scroll bar widget.

Table 18.128: WM_SetScrollState() parameter list

18.8 Example

The following example illustrates the difference between using a callback routine for redrawing the background and not having one. It also shows how to set your own callback function. The example is available as `WM_Redraw.c` in the examples shipped with emWin:

```
/*********************************************************************
*                SEGGER MICROCONTROLLER SYSTEME GmbH                 *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                      emWin example code                            *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        : WM_Redraw.c
Purpose     : Demonstrates the redrawing mechanism of the Window Manager
----------------------------------------------------------------------
*/

#include "GUI.H"

/*******************************************************************
*
*             Callback routine for background window
*
********************************************************************
*/

static void cbBackgroundWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_Clear();
  default:
    WM_DefaultProc(pMsg);
  }
}

/*******************************************************************
*
*             Callback routine for foreground window
*
********************************************************************
*/

static void cbForegroundWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(GUI_GREEN);
    GUI_Clear();
    GUI_DispString("Foreground window");
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

/*******************************************************************
*
*          Demonstrates the redraw mechanism of emWin
*
********************************************************************
*/

static void DemoRedraw(void) {
  GUI_HWIN hWnd;
  while(1) {
    /* Create foreground window */
    hWnd = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, cbForegroundWin, 0);
    /* Show foreground window */
    GUI_Delay(1000);
    /* Delete foreground window */
    WM_DeleteWindow(hWnd);
    GUI_DispStringAt("Background of window has not been redrawn", 10, 10);
    /* Wait a while, background will not be redrawn */
    GUI_Delay(1000);
    GUI_Clear();
    /* Set callback for Background window */
```



```
    WM_SetCallback(WM_HBKWIN, cbBackgroundWin);
    /* Create foreground window */
    hWnd = WM_CreateWindow(10, 10, 100, 100,WM_CF_SHOW, cbForegroundWin, 0);
    /* Show foreground window */
    GUI_Delay(1000);
    /* Delete foreground window */
    WM_DeleteWindow(hWnd);
    /* Wait a while, background will be redrawn */
    GUI_Delay(1000);
    /* Delete callback for Background window */
    WM_SetCallback(WM_HBKWIN, 0);
 }}

/*******************************************************************
*
*                 main
*
********************************************************************
*/

void main(void) {
  GUI_Init();
  DemoRedraw();
}
```

Chapter 19

Widgets (window objects)

Widgets are windows with object-type properties. They are called controls in the Windows environments and make up the elements of the user interface. They can react automatically to certain events. For example, a button can appear in a different state if it is pressed. Widgets have properties which may be changed at any time during their existence. They are typically deleted as soon as they are not used any longer. Similar to windows, widgets are referenced by handles which are returned by the respective create function.
Widgets require the Window Manager. Once a widget is created, it is treated just like any other window. The WM ensures that it is properly displayed (and redrawn) whenever necessary. The use of widgets is not mandatory for applications or user interfaces, but they decrease development time.

19.1 Some basics

19.1.1 Available widgets

The following table shows the appearance of the currently available widgets. Some of the widgets support skinning. This method of changing the appearance is explained in detail in chapter “Skinning” on page 871. The second screenshot shows the appearance when skinning is enabled for the widget:

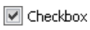
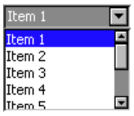
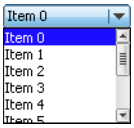
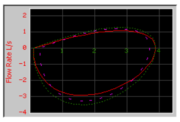
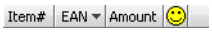
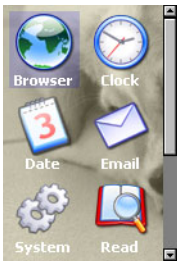
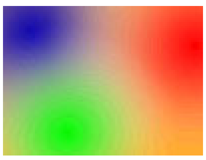
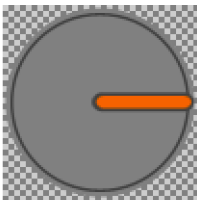
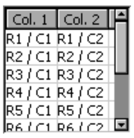
Name	Screenshot (classic)	Screenshot (skinned)	Description
BUTTON			Button which can be pressed. Text or bitmaps may be displayed on a button.
CHECKBOX			Check box which may be checked or unchecked.
DROPDOWN			Dropdown listbox, opens a listbox when pressed.
EDIT			Single-line edit field which prompts the user to type a number or text.
FRAMEWIN			Frame window. Creates the typical GUI look.
GRAPH			Graph widget, used to show curves or measured values.
HEADER			Header control, used to manage columns.
ICONVIEW			Icon view widget. Useful for icon based platforms as found in common hand held devices.
IMAGE			Image widget. Displays several image formats automatically.
KNOB			Knob widget which can be used to adjust uncountable values.
LISTBOX			Listbox which highlights items as they are selected by the user.
LISTVIEW			Listview widgets are used to creates tables.

Table 19.1: Available widgets

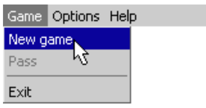
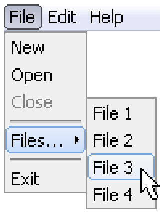
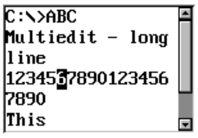
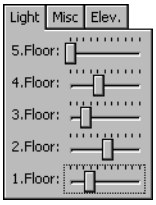
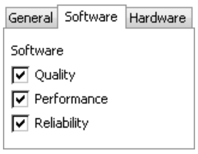
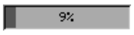
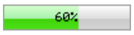
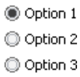
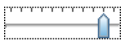
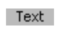
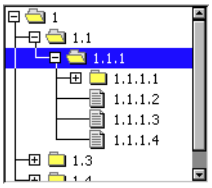
Name	Screenshot (classic)	Screenshot (skinned)	Description
LISTWHEEL			Listwheel widget. The data can be moved and accelerated via pointer input device.
MENU			Menu widgets are used to create horizontal and vertical menus.
MULTIEDIT			Multiedit widgets are used to edit multiple lines of text.
MULTIPAGE			Multipage widgets are used to create dialogs with multiple pages.
PROGBAR			Progress bar used for visualization.
RADIO			Radio button which may be selected. Only one button may be selected at a time.
SCROLLBAR			Scrollbar which may be horizontal or vertical.
SLIDER			Slider bar used for changing values.
SPINBOX			Spinning box to display and adjust a specific value.
TEXT			Static text controls typically used in dialogs.
TREEVIEW			Treeview widget for managing hierarchical lists.

Table 19.1: Available widgets

19.1.2 Custom widget types

emWin users have the possibility to create custom types of widgets. This can be done using a custom callback function for an existing widget in order to preserve certain functionality. In case it is required to implement a new type of widget it is recommended to use a simple window as starting point and follow the instructions in `AN03002_Custom_Widget_Type.pdf` which can also be found in the Doc-folder.

19.1.3 Understanding the redrawing mechanism

A widget draws itself according to its properties. This is done when `WM_Exec()`, `GUI_Exec()` or `GUI_Delay()` is called. In a multitasking environment, a background task is normally used to call `WM_Exec()` and update the widgets (and all other windows with callback functions).

When a property of a widget is changed, the window of the widget (or part of it) is marked as invalid, but it is not immediately redrawn. Therefore, the section of code executes very fast. The redrawing is done by the WM at a later time or it can be forced by calling `WM_Paint()` for the widget (or `WM_Exec()` until all windows are redrawn).

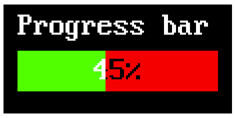
19.1.4 How to use widgets

Suppose we would like to display a progress bar. All that is needed is the following code:

```
PROGBAR_Handle hProgBar;
GUI_DispStringAt("Progress bar", 100, 20);
hProgBar = PROGBAR_Create(100, 40, 100, 20, WM_CF_SHOW);
```

The first line reserves memory for the handle of the widget. The last line actually creates the widget. The widget will then automatically be drawn by the Window Manager once `WM_Exec()` is called the next time, what may happen in a separate task.
Member functions are available for each type of widget which allow modifications to their appearance. Once the widget has been created, its properties can be changed by calling its member functions. These functions take the handle of the widget as their first argument. In order to make the progress bar created above show 45% and to change the bar colors from their defaults (dark gray/light gray) to green/red, the following section of code may be used:

```
PROGBAR_SetBarColor(hProgBar, 0, GUI_GREEN);
PROGBAR_SetBarColor(hProgBar, 1, GUI_RED);
PROGBAR_SetValue(hProgBar, 45);
```



Default configuration

All widgets also have one or more configuration macros which define various default settings such as fonts and colors used. The available configuration options are listed for each widget in their respective sections later in the chapter.

How widgets communicate

Widgets are often created as child windows. The parent window may be any type of window, even another widget. A parent window usually needs to be informed whenever something occurs with one of its children in order to ensure synchronization. Child window widgets communicate with their parent window by sending a `WM_NOTIFY_PARENT` message whenever an event occurs. The notification code sent as part of the message depends on the event.
Most widgets have one or more notification codes defining different types of events. The available notification codes for each widget (if any) are listed under their respective sections.

Skinning

The appearance of a widget can be modified by using the member functions of the respective widget. Some of the widgets support skinning. If skinning is used for a widget the 'skin' determines the appearance of the widget and some of the member functions have no effect. Details can be found in the chapter "Skinning" on page 871.

Dynamic memory usage for widgets

In embedded applications it is usually not very desirable to use dynamic memory at all because of fragmentation effects. There are a number of different strategies that can be used to avoid this, but they all work in a limited way whenever memory areas are referenced by a pointer in the application program. For this reason, emWin uses a different approach: all objects (and all data stored at run-time) are stored in memory areas referenced by a handle. This makes it possible to relocate the allocated memory areas at run-time, thus avoiding the long-term allocation problems which occur when using pointers. All widgets are thus referenced by handles.

Determine the type of a widget

The type of a widget can be determined by comparing the callback function of a specific widget with the public callback functions of the widget API. The following shows a short example how to determine the type of a widget. In case of overwritten callback functions the method should be adapted:

```
WM_CALLBACK * pCb;

pCb = WM_GetCallback(hWidget);
if        (pCb == BUTTON_Callback) {
  // Widget is a button
} else if (pCb == DROPDOWN_Callback) {
  // Widget is a dropdown
} else if (pCb == LISTBOX_Callback) {
  // Widget is a listbox
} else if (...) {
  ...
}
```

This code works only if callback function have not been overwritten. As custom callback functions are used, the code above needs to be adapted.

19.2 Configuration options

Type	Macro	Default	Description
B	WIDGET_USE_PARENT_EFFECT	0	If set to 1, each 'child widget' of a widget, has the same effect as the parent widget. If for example a listbox needs to create a scroll bar, the new scroll bar has the same effect as the listbox.
B	WIDGET_USE_SCHEME_LARGE	0	If set to 1, the default appearance of the widgets is large sized. That means that all widgets which show text are configured to use large sized default fonts.
B	WIDGET_USE_SCHEME_MEDIUM	0	If set to 1, the default appearance of the widgets is medium sized. That means that all widgets which show text are configured to use medium sized default fonts.
B	WIDGET_USE_SCHEME_SMALL	1	If set to 1, the default appearance of the widgets is small sized. That means that all widgets which show text are configured to use small sized default fonts.
B	WIDGET_USE_FLEX_SKIN	0	If set to 1, widgets are drawn using the Flex Skin by default. Detailed information about Skinning can be found in the chapter "Skinning" on page 871.

Table 19.2: Configuration options

WIDGET_USE_SCHEME...

The table below shows the default appearance of the widget schemes:

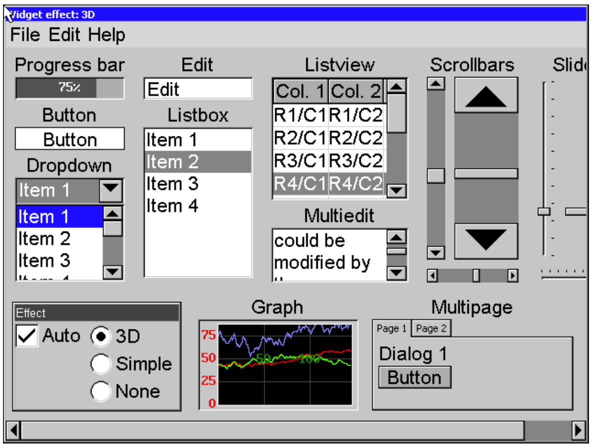
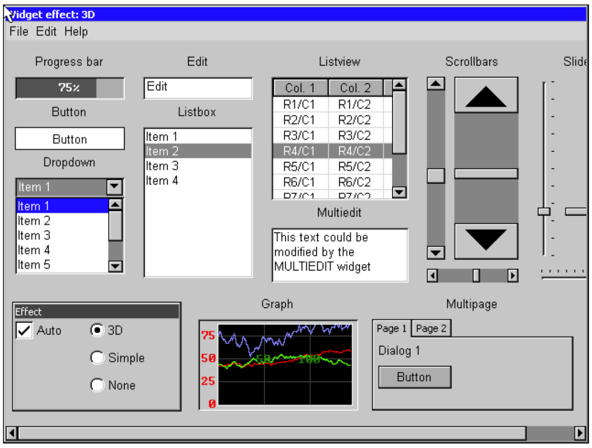
WIDGET_USE_SCHEME_LARGE	WIDGET_USE_SCHEME_MEDIUM
	

Table 19.3: Widget schemes

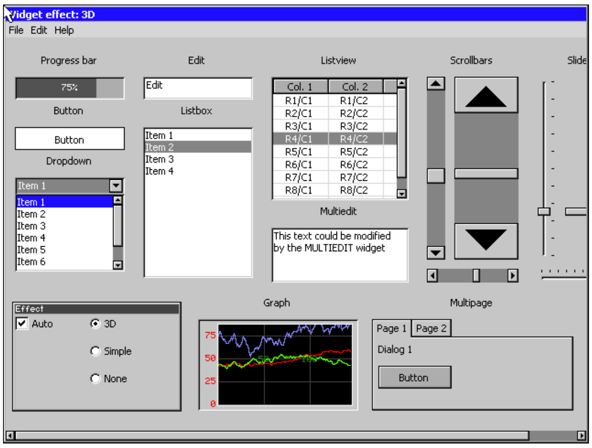
WIDGET_USE_SCHEME_SMALL


Table 19.4: Widget schemes

19.3 Widget IDs

In order to be able to separate all widgets from each other IDs can be assigned. This is usually done by using the according parameter of the <WIDGET>_Create...()-functions. To make sure that every widget has its unique Id, predefined symbols may be used. The predefined symbols are listed in the subsections of the according widgets. If the predefined symbols do not match ones requirements, custom unique IDs may be defined as follows:

```
#define MY_WIDGET_ID_0 (GUI_ID_USER + 0)
#define MY_WIDGET_ID_1 (GUI_ID_USER + 1)
#define MY_WIDGET_ID_2 (GUI_ID_USER + 2)
#define MY_WIDGET_ID_3 (GUI_ID_USER + 3)
  .
  .
  .
```

19.4 General widget API

19.4.1 WM routines used for widgets

Since widgets are essentially windows, they are compatible with any of the Window Manager API routines. The handle of the widget is used as the `hWin` parameter and the widget is treated like any other window. The WM functions most commonly used with widgets are listed as follows:

Routine	Description
`WM_DeleteWindow()`	Deletes a window.
`WM_DisableWindow()`	Disables usage of Memory Devices for redrawing.
`WM_EnableMemdev()`	Enables usage of Memory Devices for redrawing.
`WM_InvalidateWindow()`	Invalidates a window.
`WM_Paint()`	Draws or redraws a window immediately.

Table 19.5: WM routines used for widgets

The complete list of WM-related functions can be found in the chapter “The Window Manager (WM)” on page 375.

19.4.2 Common routines

The table below lists available widget-related routines in alphabetical order. These functions are common to all widgets, and are listed here in order to avoid repetition. Detailed descriptions of the routines follow. The additional member functions available for each widget may be found in later sections.

Routine	Description
`<WIDGET>_Callback()`	Default callback function.
`<WIDGET>_CreateIndirect()`	Used for automatic creation in dialog boxes.
`<WIDGET>_CreateUser()`	Creates a widget using extra bytes as user data.
`<WIDGET>_GetUserData()`	Retrieves the data set with <WIDGET>_SetUserData.
`<WIDGET>_SetUserData()`	Sets the extra data of a widget.
`WIDGET_GetDefaultEffect()`	Returns the default effect used for widgets.
`WIDGET_SetDefaultEffect()`	Sets the default effect used for widgets.
`WIDGET_SetEffect()`	Sets the effect used for a given widget.

Table 19.6: Common routines

<WIDGET>_Callback()

Description

Default callback function of the widgets to be used from within overwritten callback function.

Prototype

```
void <WIDGET>_Callback(WM_MESSAGE * pMsg);
```

Parameter	Description
`pMsg`	Pointer to a data structure of type WM_MESSAGE.

Table 19.7: <WIDGET>_Callback() parameter list

Additional information

A default callback function of a widget should not be called directly. It is only to be used from within an overwritten callback function.
For details about the `WM_MESSAGE` data structure, refer to “Messages” on page 388.

<WIDGET>_CreateIndirect()

Description

Creates a widget to be used in dialog boxes.

Prototype

```
<WIDGET>_Handle <WIDGET>_CreateIndirect(
  const GUI_WIDGET_CREATE_INFO * pCreateInfo,
  WM_HWIN                        hParent,
  int                            x0,
  int                            y0,
  WM_CALLBACK                  * cb
);
```

Parameter	Description
`pCreateInfo`	Pointer to a `GUI_WIDGET_CREATE_INFO` structure (see below).
`hParent`	Handle of parent window.
`x0`	Leftmost pixel of the widget (in parent coordinates).
`y0`	Topmost pixel of the widget (in parent coordinates).
`cb`	Pointer to a callback function.

Table 19.8: <WIDGET>_CreateIndirect() parameter list

Additional information

Any widget may be created indirectly by using the appropriate prefix. For example:
`BUTTON_CreateIndirect()` to indirectly create a button widget,
`CHECKBOX_CreateIndirect()` to indirectly create a check box widget, and so on.

A widget only needs to be created indirectly if it is to be included in a dialog box. Otherwise, it may be created directly by using the `<WIDGET>_Create()` functions. See the chapter “Dialogs” on page 831 for more information about dialog boxes.

Resource table

The `GUI_WIDGET_CREATE_INFO` data structure is defined in the dialog resource table as follows:

```
typedef struct {
  GUI_WIDGET_CREATE_FUNC * pfCreateIndirect; // Create function
  const char * pName;                       // Text (not used for all widgets)
  I16 Id;                                   // Window ID of the widget
  I16 x0, y0, xSize, ySize;                 // Size and position of the widget
  I16 Flags;                                // Widget-specific flags (or 0)
  I32 Para;                                 // Widget-specific parameter (or 0)
  U32 NumExtraBytes;                        // Number of extra bytes usable
                                            // with <WIDGET>_SetUserData &
                                            //      <WIDGET>_GetUserData
} GUI_WIDGET_CREATE_INFO;
```

Widget flags and parameters are optional, and vary depending on the type of widget. The available flags and parameters for each widget (if any) will be listed under the appropriate section later in this chapter.

<WIDGET>_CreateUser()

Description

Creates a widget using extra bytes as user data. This function is similar to the `<WIDGET>_CreateEx()`-function of the appropriate widget in every case except the additional parameter NumExtraBytes.

Prototype

```
<WIDGET>_Handle <WIDGET>_CreateUser(int x0, int y0, ..., int Id,
                                    int NumExtraBytes);
```

Parameter	Description
NumBytes	Number of extra bytes to be allocated

Table 19.9: <WIDGET>_CreateUser() parameter list

Return value

Handle of the created widget; 0 if the function fails.

Additional information

For more information about the other parameters the appropriate `<WIDGET>_CreateEx()`-functions can be referred to.

<WIDGET>_GetUserData()

Description

Retrieves the data set with `<WIDGET>_SetUserData`.

Prototype

```
int <WIDGET>_GetUserData(<WIDGET>_Handle hObj,      void * pDest,
                         int                NumBytes);
```

Parameter	Description
hObj	Handle of the widget
pDest	Pointer to buffer
NumBytes	Number of bytes to read

Table 19.10: <WIDGET>_GetUserData() parameter list

Return value

Number of bytes read

Additional information

The maximum number of bytes returned by this function is the number of extra bytes specified when creating the widget using `<WIDGET>_CreateUser()` or `<WIDGET>_CreateIndirect()`.

<WIDGET>_SetUserData()

Description

Sets the extra data of a widget.

Prototype

```
int <WIDGET>_GetUser(<WIDGET>_Handle hObj,      void * pDest,
                     int                NumBytes);
```

Parameter	Description
hObj	Handle of the widget
pDest	Pointer to buffer
NumBytes	Number of bytes to write

Table 19.11: <WIDGET>_SetUserData() parameter list

Return value

Number of bytes written

Additional information

The maximum number of bytes used to store user data is the number of extra bytes specified when creating the widget using `<WIDGET>_CreateUser()` or `<WIDGET>_CreateIndirect()`.

WIDGET_GetDefaultEffect()

Description

Returns the default effect used for widgets.

Prototype

```
const WIDGET_EFFECT * WIDGET_GetDefaultEffect(void);
```

Return value

The result of the function is a pointer to a `WIDGET_EFFECT` structure.

Additional information

For more information, refer to “WIDGET_SetDefaultEffect()” on page 459.

WIDGET_SetDefaultEffect()

Description

Sets the default effect used for widgets.

Prototype

```
const WIDGET_EFFECT * WIDGET_SetDefaultEffect(
                                        const WIDGET_EFFECT * pEffect);
```

Parameter	Description
pEffect	Pointer to a WIDGET_EFFECT structure. See table below.

Table 19.12: WIDGET_SetDefaultEffect() parameter list

Permitted values for element pEffect	
&WIDGET_Effect_3D	Sets the default effect to '3D'.
&WIDGET_Effect_None	Sets the default effect to 'None'.
&WIDGET_Effect_Simple	Sets the default effect to 'Simple'.

Return value

Previous used default effect.

Additional information

The following table shows the appearance of some widgets in dependence of the used effect:

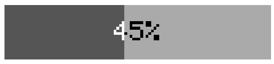
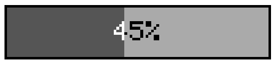
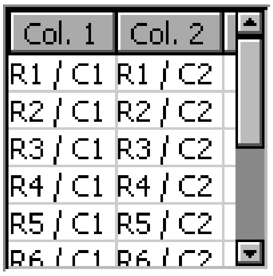
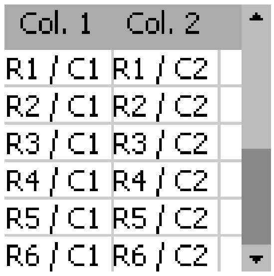
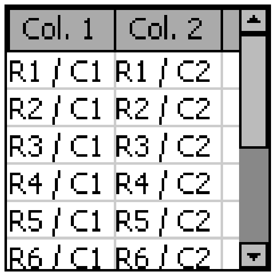
'3D'	'None'	'Simple'
		
		
		

Table 19.13: Appearance of different effects

WIDGET_SetEffect()

Description

Sets the effect for the given widget.

Prototype

```
void WIDGET_SetEffect(WM_HWIN hObj, const WIDGET_EFFECT * pEffect);
```

Parameter	Description
hObj	Handle of widget.
pEffect	Pointer to a WIDGET_EFFECT structure. For details, refer to "WIDGET_SetDefaultEffect()" on page 459.

Table 19.14: WIDGET_SetEffect() parameter list

19.4.3 User drawn widgets

Some widgets supports owner drawing, for example the LISTBOX widget. If the user draw mode of a widget has been activated a application-defined function of type WIDGET_DRAW_ITEM_FUNC will be called to draw the widget (item). The prototype of an application-defined owner draw function should be defined as follows:

Prototype

```
int WIDGET_DRAW_ITEM_FUNC(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Description
pDrawItemInfo	Pointer to a WIDGET_ITEM_DRAW_INFO structure.

Table 19.15: WIDGET_DRAW_ITEM_FUNC parameter list

Elements of structure WIDGET_ITEM_DRAW_INFO

Data type	Element	Description
WM_HWIN	hWin	Handle to the widget.
int	Cmd	See table below.
int	ItemIndex	Zero based index of item to be drawn.
int	Col	Zero based column index of item to be drawn.
int	x0	x0 window coordinate which is used to draw the item.
int	y0	y0 window coordinate which is used to draw the item.
int	x1	x1 window coordinate which is used to draw the item.
int	y1	y1 window coordinate which is used to draw the item.

Table 19.16: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element Cmd	
WIDGET_ITEM_GET_XSIZE	The function returns the x-size (width) in pixels of the given item.
WIDGET_ITEM_GET_YSIZE	The function returns the y-size (height) in pixels of the given item.
WIDGET_ITEM_DRAW	The function draws the given item at the given position.
WIDGET_DRAW_BACKGROUND	The background of the widget should be drawn.
WIDGET_DRAW_OVERLAY	This command is sent after all other drawing operations have been finished and enables the possibility to draw some overlaying items above the widget.

Return value

Depends on the given command.

Reaction to commands

The function has to react to the command given in the WIDGET_ITEM_DRAW_INFO structure. This can be done in one of 2 ways:

- By calling the appropriate default function supplied by the particular widget (for example, LISTBOX_OwnerDraw())
- By supplying code that reacts accordingly.

Commands

The commands listed below are supported and should be reacted to by the function. As explained above, the default owner draw function should be called for all not handled functions. This can save code size (for example if the height is the same as the default height) and makes sure that your code stays compatible if additional commands are introduced in future versions of the software.

WIDGET_ITEM_GET_XSIZE

The X-size in pixels of the given item has to be returned.

WIDGET_ITEM_GET_YSIZE

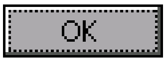
The Y-size (height) in pixels of the given item has to be returned.

WIDGET_ITEM_DRAW

The given item has to be drawn. x0 and y0 of the `WIDGET_ITEM_DRAW_INFO` structure specify the position of the item in window coordinates. The item has to fill its entire rectangle; the rectangle is defined by the starting position x0, y0 supplied to the function and the sizes returned by the function as reaction to the commands `WIDGET_ITEM_GET_YSIZE`, `WIDGET_ITEM_GET_XSIZE`. It may NOT leave a part of this rectangular area unpainted. It can not paint outside of this rectangular area because the clip rectangle has been set before the function call.

19.5 BUTTON: Button widget

BUTTON widgets are commonly used as the primary user interface element for touch-screens. If the button has the input focus, it also reacts on the keys GUI_KEY_SPACE and GUI_KEY_ENTER. Buttons may be displayed with text, as shown below, or with a bitmap.



All BUTTON-related routines are located in the file(s) `BUTTON*.c`, `BUTTON.h`. All identifiers are prefixed BUTTON.

Skinning...

...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter “Skinning” on page 871.

19.5.1 Configuration options

Type	Macro	Default	Description
N	`BUTTON_3D_MOVE_X`	1	Number of pixels that text/bitmap moves in horizontal direction in pressed state.
N	`BUTTON_3D_MOVE_Y`	1	Number of pixels that text/bitmap moves in vertical direction in pressed state.
N	`BUTTON_ALIGN_DEFAULT`	GUI_TA_HCENTER | GUI_TA_VCENTER	Alignment used to display the button text.
N	`BUTTON_BKCOLOR0_DEFAULT`	0xAAAAAA	Background color, unpressed state.
N	`BUTTON_BKCOLOR1_DEFAULT`	GUI_WHITE	Background color, pressed state.
N	`BUTTON_FOCUSCOLOR_DEFAULT`	GUI_BLACK	Default color for rendering the focus rectangle.
S	`BUTTON_FONT_DEFAULT`	&GUI_Font13_1	Font used for button text.
B	`BUTTON_REACT_ON_LEVEL`	0	See description below.
N	`BUTTON_TEXTCOLOR0_DEFAULT`	GUI_BLACK	Text color, unpressed state.
N	`BUTTON_TEXTCOLOR1_DEFAULT`	GUI_BLACK	Text color, pressed state.

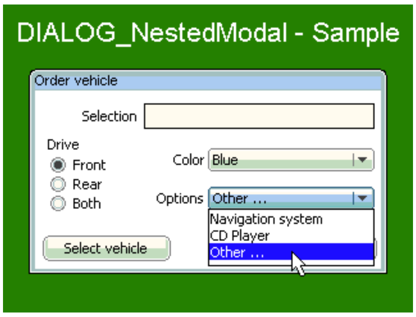
Table 19.17: Configuration options

BUTTON_REACT_ON_LEVEL

There are 2 ways for a BUTTON widget to handle PID events.
The default way is recognizing and processing all PID events which happen on the BUTTON area. This includes PIDs moved in the BUTTON area in pressed state. The BUTTON widget would "accept" the pressed state and change its state accordingly. This behavior can be useful in combination with a touch panel, but the disadvantage is that BUTTONs may be mistakenly clicked.
The other way of handling PID events for the BUTTON widget would be reacting on level changes, only. This would make BUTTON widgets react, only if the PID state changes on the BUTTON. This logic can be enabled either by defining `BUTTON_REACT_ON_LEVEL` with 1 or by calling the function `BUTTON_SetReactOnLevel()`.

Example for an unwanted BUTTON click

This example shows how a BUTTON widget may be mistakenly clicked without the BUTTON being configured to react on level.



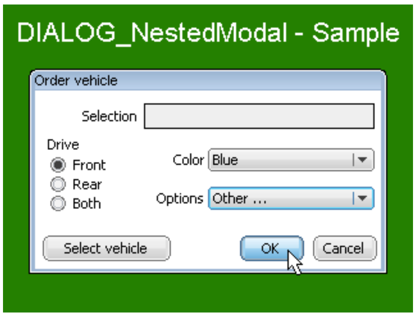


Table 19.18: BUTTON_REACT_ON_LEVEL example screenshots

BUTTON_BKCOLOR0_DEFAULT, BUTTON_BKCOLOR1_DEFAULT

The default for the BUTTON widget is to use a white background in the pressed state. This has been done purposely because it makes it very obvious that the button is pressed, on any kind of display. If you want the background color of the BUTTON widget to be the same in both its pressed and unpressed states, change `BUTTON_BKCOLOR1_DEFAULT` to `BUTTON_BKCOLOR0_DEFAULT`.

19.5.2 Predefined IDs

The following symbols define IDs which may be used to make BUTTON widgets distinguishable from creation: `GUI_ID_BUTTON0` - `GUI_ID_BUTTON9`

19.5.3 Notification codes

The following events are sent from a BUTTON widget to its parent window as part of a `WM_NOTIFY_PARENT` message:

Message	Description
`WM_NOTIFICATION_CLICKED`	BUTTON has been clicked.
`WM_NOTIFICATION_RELEASED`	BUTTON has been released.
`WM_NOTIFICATION_MOVED_OUT`	BUTTON has been clicked and pointer has been moved out of the BUTTON widget without releasing.

Table 19.19: Notification codes

19.5.4 Keyboard reaction

The BUTTON widget reacts to the following keys if it has the input focus:

Key	Reaction
`GUI_KEY_ENTER`	If the keys is pressed, the BUTTON reacts as it has been pressed and immediately released.
`GUI_KEY_SPACE`	If the keys is pressed, the BUTTON state changes to pressed. If the keys is released, the BUTTON state changes to unpressed.

Table 19.20: Keyboard reaction

19.5.5 BUTTON API

The table below lists the available emWin BUTTON-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
BUTTON_Create()	Creates a BUTTON widget. (Obsolete)
BUTTON_CreateAsChild()	Creates a BUTTON widget as a child window. (Obsolete)
BUTTON_CreateEx()	Creates a BUTTON widget.
BUTTON_CreateIndirect()	Creates a BUTTON widget from a resource table entry.
BUTTON_CreateUser()	Creates a BUTTON widget using extra bytes as user data.
BUTTON_GetBitmap()	Returns the pointer to the BUTTON bitmap.
BUTTON_GetBkColor()	Returns the background color of the BUTTON
BUTTON_GetDefaultBkColor()	Returns the default background color for BUTTON widgets.
BUTTON_GetDefaultFont()	Returns the default font for BUTTON widgets.
BUTTON_GetDefaultTextAlign()	Returns the default text alignment for BUTTON widgets.
BUTTON_GetDefaultTextColor()	Returns the default text color for BUTTON widgets.
BUTTON_GetFont()	Returns the pointer to the font of the BUTTON widget.
BUTTON_GetText()	Retrieves the text of a specified BUTTON.
BUTTON_GetTextAlign()	Returns the alignment of the BUTTON text.
BUTTON_GetTextColor()	Returns the text color of the specified BUTTON.
BUTTON_GetUserData()	Retrieves the data set with BUTTON_SetUserData().
BUTTON_IsPressed()	Returns if the BUTTON is pressed or not.
BUTTON_SetBitmap()	Sets the bitmap used when displaying the BUTTON.
BUTTON_SetBitmapEx()	Sets the bitmap used when displaying the BUTTON.
BUTTON_SetBkColor()	Sets the background color of the BUTTON widget.
BUTTON_SetBMP()	Sets a bitmap to be displayed on a BUTTON widget.
BUTTON_SetBMPEx()	Sets a bitmap from external memory to be displayed on a BUTTON widget.
BUTTON_SetDefaultBkColor()	Sets the default background color for BUTTON widgets.
BUTTON_SetDefaultFocusColor()	Sets the default focus color for the BUTTON widget.
BUTTON_SetDefaultFont()	Sets the default font for BUTTON widgets.
BUTTON_SetDefaultTextAlign()	Sets the default text alignment for BUTTON widgets.
BUTTON_SetDefaultTextColor()	Sets the default text color for BUTTON widgets.
BUTTON_SetFocusColor()	Sets the focus color of the BUTTON widget.
BUTTON_SetFocussable()	Sets the ability to receive the input focus.
BUTTON_SetFont()	Selects the font for the text.
BUTTON_SetPressed()	Sets the state of the BUTTON to pressed or unpressed.
BUTTON_SetReactOnLevel()	Sets all BUTTON widgets to react on level.
BUTTON_SetReactOnTouch()	Sets all BUTTON widgets to react on touch.
BUTTON_SetStreamedBitmap()	Sets the bitmap used when displaying the BUTTON widget.
BUTTON_SetStreamedBitmapEx()	Sets the bitmap used when displaying the BUTTON widget.
BUTTON_SetText()	Sets the text.
BUTTON_SetTextAlign()	Sets the alignment of the BUTTON text.
BUTTON_SetTextColor()	Set the color(s) for the text.
BUTTON_SetTextOffset()	Adjusts the position of the BUTTON text considering the current text alignment setting.
BUTTON_SetUserData()	Sets the extra data of a BUTTON widget.

Table 19.21: BUTTON API list

BUTTON_Create()

(Obsolete, `BUTTON_CreateEx()` should be used instead)

Description

Creates a BUTTON widget of a specified size at a specified location.

Prototype

```
BUTTON_Handle BUTTON_Create(int x0,     int y0,
                            int xSize, int ySize,
                            int Id,     int Flags);
```

Parameter	Description
x0	Leftmost pixel of the button (in parent coordinates).
y0	Topmost pixel of the button (in parent coordinates).
xSize	Horizontal size of the button (in pixels).
ySize	Vertical size of the button (in pixels).
Id	ID to be returned when button is pressed.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).

Table 19.22: BUTTON_Create() parameter list

Return value

Handle of the created BUTTON widget; 0 if the function fails.

BUTTON_CreateAsChild()

(Obsolete, BUTTON_CreateEx should be used instead)

Description

Creates a BUTTON widget as a child window.

Prototype

```
BUTTON_Handle BUTTON_CreateAsChild(int      x0,       int y0,
                                   int      xSize,    int ySize,
                                   WM_HWIN hParent, int Id,
                                   int      Flags);
```

Parameter	Description
x0	X-position of the button relative to the parent window.
y0	Y-position of the button relative to the parent window.
xSize	Horizontal size of the button (in pixels).
ySize	Vertical size of the button (in pixels).
hParent	Handle of parent window. If 0, the BUTTON widget will be a child of the desktop (top-level window).
Id	ID to be returned when button is pressed.
Flags	Window create flags (see BUTTON_Create()).

Table 19.23: BUTTON_CreateAsChild() parameter list

Return value

Handle of the created BUTTON widget; 0 if the function fails.

BUTTON_CreateEx()

Description

Creates a BUTTON widget of a specified size at a specified location.

Prototype

```
BUTTON_Handle BUTTON_CreateEx(int      x0,       int y0,
                              int      xSize,    int ySize,
                              WM_HWIN hParent, int WinFlags,
```

```
                                    int     ExFlags, int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the BUTTON widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	Not used yet, reserved for future use.
Id	Window ID of the widget.

Table 19.24: BUTTON_CreateEx() parameter list

Return value

Handle of the created BUTTON widget; 0 if the function fails.

Additional information

If the possibility of storing user data is a matter the function BUTTON_CreateUser() should be used instead.

BUTTON_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The elements Flags and Para of the according GUI_WIDGET_CREATE_INFO structure are not used.

BUTTON_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function BUTTON_CreateEx() can be referred to.

BUTTON_GetBitmap()

Description

Returns a pointer to the optional BUTTON bitmap.

Prototype

```
const GUI_BITMAP * BUTTON_GetBitmap(BUTTON_Handle hObj,
                                    unsigned int  Index);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index of desired bitmap. See table below.

Table 19.25: BUTTON_GetBitmap() parameter list

Permitted values for parameter Index	
BUTTON_BI_DISABLED	Bitmap for disabled state.
BUTTON_BI_PRESSED	Bitmap for pressed state.
BUTTON_BI_UNPRESSED	Bitmap for unpressed state.

Return value

Pointer to the bitmap, 0 if no bitmap exist.

Additional information

For details about how to set a button bitmap, refer to "BUTTON_SetBitmap()" on page 470 and "BUTTON_SetBitmapEx()" on page 471.

BUTTON_GetBkColor()

Description

Returns the background color of the given BUTTON widget.

Prototype

```
GUI_COLOR BUTTON_GetBkColor(BUTTON_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Color index. See table below.

Table 19.26: BUTTON_GetBkColor() parameter list

Permitted values for parameter Index	
BUTTON_CI_DISABLED	Color for disabled state.
BUTTON_CI_PRESSED	Color for pressed state.
BUTTON_CI_UNPRESSED	Color for unpressed state.

Return value

Background color of the given BUTTON widget

BUTTON_GetDefaultBkColor()

Description

Returns the default background color for BUTTON widgets.

Prototype

```
GUI_COLOR BUTTON_GetDefaultBkColor(unsigned Index);
```

Parameter	Description
Index	Index for color. See table below.

Table 19.27: BUTTON_GetDefaultBkColor() parameter list

Permitted values for parameter Index	
BUTTON_CI_DISABLED	Color for disabled state.
BUTTON_CI_PRESSED	Color for pressed state.
BUTTON_CI_UNPRESSED	Color for unpressed state.

Return value

Default background color for BUTTON widgets

BUTTON_GetDefaultFont()

Description

Returns the pointer to the font used to display the text of BUTTON widgets.

Prototype

```
const GUI_FONT * BUTTON_GetDefaultFont(void);
```

Return value

Pointer to the font used to display the text of BUTTON widgets.

BUTTON_GetDefaultTextAlign()

Description

Returns the default text alignment used to display the text of BUTTON widgets.

Prototype

```
int BUTTON_GetDefaultTextAlign(void);
```

Return value

Default text alignment used to display the text of BUTTON widgets.

BUTTON_GetDefaultTextColor()

Description

Returns the default text color used to display the text of BUTTON widgets.

Prototype

```
GUI_COLOR BUTTON_GetDefaultTextColor(unsigned Index);
```

Parameter	Description
`Index`	Index for color. See table below.

Table 19.28: BUTTON_GetDefaultTextColor() parameter list

Permitted values for parameter `Index`	
`BUTTON_CI_DISABLED`	Color for disabled state.
`BUTTON_CI_PRESSED`	Color for pressed state.
`BUTTON_CI_UNPRESSED`	Color for unpressed state.

Return value

Default text color used to display the text of BUTTON widgets.

BUTTON_GetFont()

Description

Returns a pointer to the font used to display the text of the given BUTTON widget

Prototype

```
const GUI_FONT * BUTTON_GetFont(BUTTON_Handle hObj);
```

Parameter	Description
`hObj`	Handle of BUTTON widget.

Table 19.29: BUTTON_GetFont() parameter list

Return value

Pointer to the font used to display the text of the given BUTTON widget.

BUTTON_GetText()

Description

Retrieves the text of the specified BUTTON widget.

Prototype

```
void BUTTON_GetText(BUTTON_Handle hObj, char * pBuffer, int MaxLen);
```

Parameter	Description
hObj	Handle of BUTTON widget.
pBuffer	Pointer to buffer.
MaxLen	Size of buffer.

Table 19.30: BUTTON_GetText() parameter list

BUTTON_GetTextAlign()

Description

Returns the alignment of the BUTTON text.

Prototype

```
int BUTTON_GetTextAlign(BUTTON_Handle hObj);
```

Parameter	Description
hObj	Handle of the BUTTON widget.

Table 19.31: BUTTON_GetTextAlign() parameter list

Return value

`Alignment of the BUTTON text.`

BUTTON_GetTextColor()

Description

Returns the text color of the given BUTTON widget.

Prototype

```
GUI_COLOR BUTTON_GetTextColor(BUTTON_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for color. See table below.

Table 19.32: BUTTON_GetTextColor() parameter list

Permitted values for parameter Index	
BUTTON_CI_DISABLED	Color for disabled state.
BUTTON_CI_PRESSED	Color for pressed state.
BUTTON_CI_UNPRESSED	Color for unpressed state.

Return value

Text color of the given BUTTON widget.

BUTTON_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

BUTTON_IsPressed()

Description

Returns if the BUTTON is pressed or not.

Prototype

```
unsigned BUTTON_IsPressed(BUTTON_Handle hObj);
```

Parameter	Description
hObj	Handle of widget.

Table 19.33: BUTTON_IsPressed() parameter list

Return value

1 if the button is pressed, 0 if not.

BUTTON_SetBitmap()

Description

Sets the bitmap(s) to be used when displaying a specified BUTTON widget.

Prototype

```
void BUTTON_SetBitmap(BUTTON_Handle      hObj,
                      unsigned int       Index,
                      const GUI_BITMAP * pBitmap);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for bitmap. See table below.
pBitmap	Pointer to the bitmap structure.

Table 19.34: BUTTON_SetBitmap() parameter list

Permitted values for parameter Index	
BUTTON_BI_DISABLED	Bitmap for disabled state.
BUTTON_BI_PRESSED	Bitmap for pressed state.
BUTTON_BI_UNPRESSED	Bitmap for unpressed state.

Additional information

If only a bitmap for the unpressed state is set the button will show it also when it is pressed or disabled. To deactivate a previously set bitmap, `NULL` has to be passed as `pBitmap`.

BUTTON_SetBitmapEx()

Description

Sets the bitmap(s) to be used when displaying a specified BUTTON widget.

Prototype

```
void BUTTON_SetBitmapEx(BUTTON_Handle       hObj,
                        unsigned int        Index,
                        const GUI_BITMAP *  pBitmap,
                        int                 x,
                        int                 y);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for bitmap (see `BUTTON_SetBitmap()`).
pBitmap	Pointer to the bitmap structure.
x	X-position for the bitmap relative to the button.
y	Y-position for the bitmap relative to the button.

Table 19.35: BUTTON_SetBitmapEx() parameter list

Additional information

If only a bitmap for the unpressed state is set the BUTTON widget will show it also when it is pressed or disabled.

BUTTON_SetBkColor()

Description

Sets the background color of a BUTTON widget.

Prototype

```
void BUTTON_SetBkColor(BUTTON_Handle hObj,   unsigned int Index,
                       GUI_COLOR     Color);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for color. See table below.
Color	Background color to be set.

Table 19.36: BUTTON_SetBkColor() parameter list

Permitted values for parameter Index	
BUTTON_CI_DISABLED	Sets the color to be used when button is disabled.
BUTTON_CI_PRESSED	Sets the color to be used when button is pressed.
BUTTON_CI_UNPRESSED	Sets the color to be used when button is unpressed.

BUTTON_SetBMP()

Description

Sets the bitmap to be displayed on the specified BUTTON widget.

Prototype

```
void BUTTON_SetBMP(BUTTON_Handle hObj, unsigned int Index,
                   const void * pBitmap);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for bitmap. See table below.
pBitmap	Pointer to bitmap file data

Table 19.37: BUTTON_SetBMP() parameter list

Permitted values for parameter Index	
BUTTON_BI_DISABLED	Sets the bitmap to be used when button is disabled.
BUTTON_BI_PRESSED	Sets the bitmap to be used when button is pressed.
BUTTON_BI_UNPRESSED	Sets the bitmap to be used when button is unpressed.

Additional information

If a bitmap was set only for the unpressed state, it will be also displayed in pressed or disabled state. For additional information regarding bitmap files, refer to “BMP file support” on page 162.

BUTTON_SetBMPEx()

Description

Sets the bitmap to be displayed at the specified position on the given BUTTON widget.

Prototype

```
void BUTTON_SetBMPEx(BUTTON_Handle   hObj,     unsigned int Index,
                     const void    * pBitmap,  int          x,
                     int             y);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for bitmap (see BUTTON_SetBitmap()).
pBitmap	Pointer to bitmap file data
x	X-position for the bitmap relative to the button.
y	Y-position for the bitmap relative to the button.

Table 19.38: BUTTON_SetBMPEx() parameter list

Additional information

If only a bitmap for the unpressed state is set the BUTTON widget will show it also when it is pressed or disabled.
For additional information regarding bitmap files, refer to “BMP file support” on page 162.

BUTTON_SetDefaultBkColor()

Description

Sets the default background color used for BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Description
`Color`	Color to be used.
`Index`	Index for color. See table below.

Table 19.39: BUTTON_SetDefaultBkColor() parameter list

Permitted values for parameter `Index`	
`BUTTON_CI_DISABLED`	Color for disabled state.
`BUTTON_CI_PRESSED`	Color for pressed state.
`BUTTON_CI_UNPRESSED`	Color for unpressed state.

BUTTON_SetDefaultFocusColor()

Description

Sets the default focus rectangle color for BUTTON widgets.

Prototype

```
GUI_COLOR BUTTON_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Description
`Color`	Default color to be used for BUTTON widgets.

Table 19.40: BUTTON_SetDefaultFocusColor() parameter list

Return value

Previous default focus rectangle color.

Additional information

For more information, refer to “BUTTON_SetFocusColor()” on page 474.

BUTTON_SetDefaultFont()

Description

Sets a pointer to a GUI_FONT structure used to display the text of BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
`pFont`	Pointer to GUI_FONT structure to be used.

Table 19.41: BUTTON_SetDefaultFont() parameter list

BUTTON_SetDefaultTextAlign()

Description

Sets the default text alignment used to display the text of BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultTextAlign(int Align);
```

Parameter	Description
`Align`	Text alignment to be used. For details, refer to “GUI_SetTextAlign()” on page 105.

Table 19.42: BUTTON_SetDefaultTextAlign() parameter list

BUTTON_SetDefaultTextColor()

Description

Sets the default text color used to display the text of BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultTextColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Description
Color	Default text color to be used.
Index	Index for color. See table below.

Table 19.43: BUTTON_SetDefaultTextColor() parameter list

Permitted values for parameter Index	
BUTTON_CI_DISABLED	Color for disabled state.
BUTTON_CI_PRESSED	Color for pressed state.
BUTTON_CI_UNPRESSED	Color for unpressed state.

BUTTON_SetFocusColor()

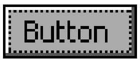
Before	After
	

Table 19.44: Button with and without focus color

Description

Sets the color used to render the focus rectangle of the BUTTON widget.

Prototype

```
GUI_COLOR BUTTON_SetFocusColor(BUTTON_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Color	Color to be used for the focus rectangle.

Table 19.45: BUTTON_SetFocusColor() parameter list

Return value

Previous color of the focus rectangle.

Additional information

The focus rectangle is only visible if the widget has the input focus.

BUTTON_SetFocussable()

Description

Sets the ability to receive the input focus.

Prototype

```
void BUTTON_SetFocussable(BUTTON_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of BUTTON widget.
State	see table below

Table 19.46: BUTTON_SetFocussable() parameter list

Permitted values for parameter State	
1	Button can receive the input focus
0	Button can't receive the input focus

BUTTON_SetFont()

Description

Sets the font of the BUTTON widget.

Prototype

```
void BUTTON_SetFont(BUTTON_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of BUTTON widget.
pFont	Pointer to the font.

Table 19.47: BUTTON_SetFont() parameter list

Additional information

If no font is selected, `BUTTON_FONT_DEF` will be used.

BUTTON_SetPressed()

Description

Sets the state of the button to pressed or unpressed.

Prototype

```
void BUTTON_SetPressed(BUTTON_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of BUTTON widget.
State	State, 1 for pressed, 0 for unpressed

Table 19.48: BUTTON_SetPressed() parameter list

BUTTON_SetReactOnLevel()

Description

Sets all BUTTON widgets to react on level changes of the PID.

Prototype

```
void BUTTON_SetReactOnLevel(void);
```

Additional Information

Alternatively to this function the configuration option `BUTTON_REACT_ON_LEVEL` can be used.

BUTTON_SetReactOnTouch()

Description

Sets all BUTTON widgets to react on touch events.

Prototype

```
void BUTTON_SetReactOnTouch(void);
```

Additional Information

The default behavior of BUTTON widgets is reacting on touch events.

BUTTON_SetStreamedBitmap()

Description

Sets the streamed bitmap(s) to be used when displaying a specified BUTTON widget.

Prototype

```
void BUTTON_SetStreamedBitmap(BUTTON_Handle              hObj,
                              unsigned int               Index,
                              const GUI_BITMAP_STREAM *  pBitmap);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for bitmap (see BUTTON_SetBitmap()).
pBitmap	Pointer to a bitmap stream.

Table 19.49: BUTTON_SetStreamedBitmap() parameter list

Additional information

For details about streamed bitmaps, refer to "Drawing streamed bitmaps" on page 135.

Example

```
BUTTON_SetStreamedBitmap(hButton, BUTTON_CI_UNPRESSED, (const GUI_BITMAP_STREAM
*)acImage);
```

BUTTON_SetStreamedBitmapEx()

Description

Sets the streamed bitmap(s) to be used when displaying the specified BUTTON widget.

Prototype

```
void BUTTON_SetStreamedBitmapEx(BUTTON_Handle              hObj,
                                unsigned int               Index,
                                const GUI_BITMAP_STREAM *  pBitmap,
                                int                        x,
                                int                        y);
```

Parameter	Description
hObj	Handle of BUTTON widget.
Index	Index for bitmap (see BUTTON_SetBitmap()).
pBitmap	Pointer to a bitmap stream.
x	X-position for the bitmap relative to the button.
y	Y-position for the bitmap relative to the button.

Table 19.50: BUTTON_SetStreamedBitmapEx() parameter list

Additional information

For details about streamed bitmaps, refer to "Drawing streamed bitmaps" on page 135().

BUTTON_SetText()

Description

Sets the text to be displayed on the BUTTON widget.

Prototype

```
int BUTTON_SetText(BUTTON_Handle hObj, const char * s);
```

Parameter	Description
hObj	Handle of the BUTTON widget.
s	Text to display.

Table 19.51: BUTTON_SetText() parameter list

Return value

0 on success, 1 on error.

BUTTON_SetTextAlign()

Description

Sets the text alignment of the BUTTON widget.

Prototype

```
void BUTTON_SetTextAlign(BUTTON_Handle hObj, int Align);
```

Parameter	Description
hObj	Handle of the BUTTON widget.
Align	Text alignment to be set (see "GUI_SetTextAlign()" on page 105)

Table 19.52: BUTTON_SetTextAlign() parameter list

Additional information

The default value of the text alignment is `GUI_TA_HCENTER | GUI_TA_VCENTER`.

BUTTON_SetTextColor()

Description

Sets the text color of the BUTTON widget.

Prototype

```
void BUTTON_SetTextColor(BUTTON_Handle hObj, unsigned int Index,
                                                GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of the BUTTON widget.
Index	Color index. See table below.
Color	Text color to be set.

Table 19.53: BUTTON_SetTextColor() parameter list

Permitted values for parameter Index	
BUTTON_CI_DISABLED	Sets the color to be used when button is disabled.
BUTTON_CI_PRESSED	Sets the color to be used when button is pressed.
BUTTON_CI_UNPRESSED	Sets the color to be used when button is unpressed.

BUTTON_SetTextOffset()

Description

Adjusts the position of the BUTTON text considering the current text alignment setting.

Prototype

```
void BUTTON_SetTextOffset(BUTTON_Handle hObj, int xPos, int yPos);
```

Parameter	Description
hObj	Handle of the BUTTON widget.
xPos	Offset to be used for the x-axis. Default is 0.
yPos	Offset to be used for the y-axis. Default is 0.

Table 19.54: BUTTON_SetTextOffset() parameter list

BUTTON_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

19.5.6 Examples

The `Sample` folder contains the following examples which show how the widget can be used:

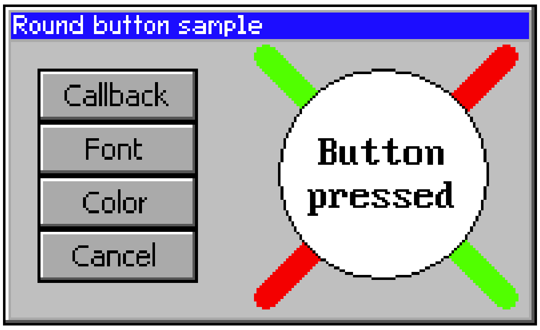
- `WIDGET_ButtonSimple.c`
- `WIDGET_ButtonPhone.c`
- `WIDGET_ButtonRound.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_ButtonSimple.c:

Screenshot of WIDGET_ButtonPhone.c:

Screenshot of WIDGET_ButtonRound.c:



19.6 CHECKBOX: Checkbox widget

One of the most familiar widgets for selecting various choices is the check box. A check box may be checked or unchecked by the user, and any number of boxes may be checked at one time. If using a keyboard interface the state of a focused check box can be toggled by the <SPACE> key. A box will appear gray if it is disabled, as seen in the table below where each of the possible check box appearances are illustrated:

	Unchecked	Checked	Third state
Enabled			
Disabled			

Table 19.55: Checkbox examples

All CHECKBOX-related routines are located in the file(s) `CHECKBOX*.c`, `CHECKBOX.h`. All identifiers are prefixed CHECKBOX.

Skinning...

...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter “Skinning” on page 871.

19.6.1 Configuration options

Type	Macro	Default	Description
N	`CHECKBOX_BKCOLOR_DEFAULT`	0xC0C0C0	Default background color.
N	`CHECKBOX_BKCOLOR0_DEFAULT`	0x808080	Background color of the default image, disabled state.
N	`CHECKBOX_BKCOLOR1_DEFAULT`	GUI_WHITE	Background color of the default image, enabled state.
N	`CHECKBOX_FGCOLOR0_DEFAULT`	0x101010	Foreground color of the default image, disabled state.
N	`CHECKBOX_FGCOLOR1_DEFAULT`	GUI_BLACK	Foreground color of the default image, enabled state.
N	`CHECKBOX_FOCUSCOLOR_DEFAULT`	GUI_BLACK	Color used to render the focus rectangle.
S	`CHECKBOX_FONT_DEFAULT`	&GUI_Font13_1	Default font used to display the optional CHECKBOX text.
S	`CHECKBOX_IMAGE0_DEFAULT`	(see table above)	Pointer to bitmap used to draw the widget if checked, disabled state.
S	`CHECKBOX_IMAGE1_DEFAULT`	(see table above)	Pointer to bitmap used to draw the widget if checked, enabled state.
N	`CHECKBOX_SPACING_DEFAULT`	4	Spacing used to display the optional CHECKBOX text beside the box.
N	`CHECKBOX_TEXTALIGN_DEFAULT`	GUI_TA_LEFT | GUI_TA_VCENTER	Default alignment of the optional CHECKBOX text.
N	`CHECKBOX_TEXTCOLOR_DEFAULT`	GUI_BLACK	Default color used to display the optional CHECKBOX text.

Table 19.56: Configuration options

19.6.2 Predefined IDs

The following symbols define IDs which may be used to make CHECKBOX widgets distinguishable from creation: GUI_ID_CHECK0 - GUI_ID_CHECK9

19.6.3 Notification codes

The following events are sent from a check box widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	CHECKBOX has been clicked.
WM_NOTIFICATION_RELEASED	CHECKBOX has been released.
WM_NOTIFICATION_MOVED_OUT	CHECKBOX has been clicked and pointer has been moved out of the box without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Status of CHECKBOX has been changed.

Table 19.57: Notification codes

19.6.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_SPACE	Toggles the checked state of the CHECKBOX widget.

Table 19.58: Keyboard reaction

19.6.5 CHECKBOX API

The table below lists the available emWin CHECKBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
CHECKBOX_Check()	Set the state of CHECKBOX to checked. (Obsolete)
CHECKBOX_Create()	Creates a CHECKBOX widget. (Obsolete)
CHECKBOX_CreateEx()	Creates a CHECKBOX widget.
CHECKBOX_CreateIndirect()	Creates a CHECKBOX widget from resource table entry.
CHECKBOX_CreateUser()	Creates a CHECKBOX widget using extra bytes as user data.
CHECKBOX_GetDefaultBkColor()	Returns the default background color for CHECKBOX widgets.
CHECKBOX_GetDefaultFont()	Returns the default font used to display the text of CHECKBOX widgets.
CHECKBOX_GetDefaultSpacing()	Returns the default spacing between the box and the text of CHECKBOX widgets.
CHECKBOX_GetDefaultTextAlign()	Returns the default alignment used to display the text of CHECKBOX widgets.
CHECKBOX_GetDefaultTextColor()	Returns the default text color used to display the text of CHECKBOX widgets.
CHECKBOX_GetState()	Returns the current state of the CHECKBOX widget.
CHECKBOX_GetText()	Returns the text of the CHECKBOX widget.
CHECKBOX_GetUserData()	Retrieves the data set with CHECKBOX_SetUserData().
CHECKBOX_IsChecked()	Return the current state (checked or not checked) of the CHECKBOX widget.
CHECKBOX_SetBkColor()	Sets the background color of the given CHECKBOX widget.
CHECKBOX_SetBoxBkColor()	Sets the background color of the box area.
CHECKBOX_SetDefaultBkColor()	Sets the default background color for CHECKBOX widget.

Table 19.59: CHECKBOX API list

Routine	Description
CHECKBOX_SetDefaultFocusColor()	Sets the default focus rectangle color for CHECKBOX widgets.
CHECKBOX_SetDefaultFont()	Sets the default font used to display the text of CHECKBOX widgets.
CHECKBOX_SetDefaultImage()	Sets the default image to be shown when a box has been checked.
CHECKBOX_SetDefaultSpacing()	Sets the default spacing between the box and the text of CHECKBOX widgets.
CHECKBOX_SetDefaultTextAlign()	Sets the default alignment used to display the text of CHECKBOX widgets.
CHECKBOX_SetDefaultTextColor()	Sets the default text color used to display the text of CHECKBOX widgets.
CHECKBOX_SetFocusColor()	Sets the color of the focus rectangle.
CHECKBOX_SetFont()	Sets the font of the CHECKBOX widget.
CHECKBOX_SetImage()	Sets the image to be shown when CHECKBOX widget been checked.
CHECKBOX_SetNumStates()	Sets the number of possible states of the CHECKBOX widget (2 or 3).
CHECKBOX_SetSpacing()	Sets the spacing between the box and the check box text.
CHECKBOX_SetState()	Sets the state of the CHECKBOX widget.
CHECKBOX_SetText()	Sets the text of the CHECKBOX widget.
CHECKBOX_SetTextAlign()	Sets the alignment used to display the text of the CHECKBOX widget.
CHECKBOX_SetTextColor()	Sets the color used to display the text of the CHECK-BOX widget.
CHECKBOX_SetUserData()	Sets the extra data of a CHECKBOX widget.
CHECKBOX_Uncheck()	Set the state of CHECKBOX to unchecked. (Obsolete)

Table 19.59: CHECKBOX API list

CHECKBOX_Check()

(Obsolete, `CHECKBOX_SetState()` should be used instead)

Before	After
	

Table 19.60: CHECKBOX_Check() before after screenshots

Description

Sets the state of a specified CHECKBOX widget to checked.

Prototype

```
void CHECKBOX_Check(CHECKBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.

Table 19.61: CHECKBOX_Check() parameter list

CHECKBOX_Create()

(Obsolete, `CHECKBOX_CreateEx` should be used instead)

Description

Creates a CHECKBOX widget of a specified size at a specified location.

Prototype

```
CHECKBOX_Handle CHECKBOX_Create(int     x0,      int y0,
                                int     xSize,   int ySize,
                                WM_HWIN hParent, int Id,
                                int     Flags);
```

Parameter	Description
x0	Leftmost pixel of the check box (in parent coordinates).
y0	Topmost pixel of the check box (in parent coordinates).
xSize	Horizontal size of the check box (in pixels).
ySize	Vertical size of the check box (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).

Table 19.62: CHECKBOX_Create() parameter list

Return value

Handle of the created CHECKBOX widget; 0 if the function fails.

Additional information

If the parameters xSize or ySize are 0 the size of the bitmap will be used as default size of the check box.

CHECKBOX_CreateEx()

Description

Creates a CHECKBOX widget of a specified size at a specified location.

Prototype

```
CHECKBOX_Handle CHECKBOX_CreateEx(int     x0,       int y0,
                                  int     xSize,    int ySize,
                                  WM_HWIN hParent,  int WinFlags,
                                  int     ExFlags,  int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new CHECKBOX widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	Not used yet, reserved for future use.
Id	Window ID of the widget.

Table 19.63: CHECKBOX_CreateEx() parameter list

Return value

Handle of the created CHECKBOX widget; 0 if the function fails.

Additional information

If the parameters xSize or ySize are 0 the size of the default check mark bitmap (11 x 11 pixels) plus the effect size will be used as default size of the check box. If the desired size of the check box is different to the default size it can be useful to set a user defined check mark image using the function `CHECKBOX_SetImage()`.
If check box text should be shown with the widget the size should be large enough to show the box + text + spacing between box and text.

CHECKBOX_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The elements `Flags` and `Para` of the according `GUI_WIDGET_CREATE_INFO` structure are not used.

CHECKBOX_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `CHECKBOX_CreateEx()` can be referred to.

CHECKBOX_GetDefaultBkColor()

Description

Returns the default background color of new CHECKBOX widgets.

Prototype

```
GUI_COLOR CHECKBOX_GetDefaultBkColor(void);
```

Return value

Default background color of new CHECKBOX widgets.

Additional information

The background color returned by this function is not the background color shown in the box, but the background color of the rest of the widget.
For more information, refer to “CHECKBOX_SetBoxBkColor()” on page 486.

CHECKBOX_GetDefaultFont()

Description

Returns a pointer to a `GUI_FONT` structure used to display the text of new CHECKBOX widgets.

Prototype

```
const GUI_FONT * CHECKBOX_GetDefaultFont(void);
```

Return value

Pointer to a `GUI_FONT` structure used to display the text of new CHECKBOX widgets.

Additional information

For more information, refer to “CHECKBOX_SetFont()” on page 489.

CHECKBOX_GetDefaultSpacing()

Description

Returns the default spacing between box and text used to display the text of new CHECKBOX widgets.

Prototype

```
int CHECKBOX_GetDefaultSpacing(void);
```


Return value

Default spacing between box and text used to display the text of new CHECKBOX widgets.

Additional information

For more information, refer to “CHECKBOX_SetSpacing()” on page 491.

CHECKBOX_GetDefaultTextAlign()

Description

Returns the default alignment used to display the text of new CHECKBOX widgets.

Prototype

```
int CHECKBOX_GetDefaultAlign(void);
```

Return value

Default alignment used to display the text of new CHECKBOX widgets.

Additional information

For more information, refer to “CHECKBOX_SetTextAlign()” on page 492.

CHECKBOX_GetDefaultTextColor()

Description

Returns the default text color used to display the text of new CHECKBOX widgets.

Prototype

```
GUI_COLOR CHECKBOX_GetDefaultTextColor(void);
```

Return value

Default text color used to display the text of new check box widgets.

Additional information

For more information, refer to “CHECKBOX_SetTextColor()” on page 493.

CHECKBOX_GetState()

Description

Returns the current state of the given CHECKBOX widget.

Prototype

```
int CHECKBOX_GetState(CHECKBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.

Table 19.64: CHECKBOX_GetState() parameter list

Return value

Current state of the given CHECKBOX widget.

Additional information

Per default a check box can have 2 states, checked (1) and unchecked (0). With the function `CHECKBOX_SetNumStates()` the number of possible states can be increased to 3. If the check box is in the third state the function returns 2.
For more information, refer to “CHECKBOX_SetNumStates()” on page 490.

CHECKBOX_GetText()

Description

Returns the optional text of the given CHECKBOX widget.

Prototype

```
int CHECKBOX_GetText(CHECKBOX_Handle hObj, char * pBuffer, int MaxLen);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
pBuffer	Pointer to buffer to which the text will be copied.
MaxLen	Buffer size in bytes.

Table 19.65: CHECKBOX_GetText() parameter list

Return value

Length of the text copied into the buffer.

Additional information

If the CHECKBOX widget contains no text the function returns 0 and the buffer remains unchanged.

CHECKBOX_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

CHECKBOX_IsChecked()

Description

Returns the current state (checked or not checked) of a specified CHECKBOX widget.

Prototype

```
int CHECKBOX_IsChecked(CHECKBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.

Table 19.66: CHECKBOX_IsChecked() parameter list

Return value

0: not checked
1: checked

CHECKBOX_SetBkColor()

Before	After
	

Table 19.67: CHECKBOX_SetBkColor() before after screenshots

Description

Sets the background color used to display the background of the CHECKBOX widget.

Prototype

```
void CHECKBOX_SetBkColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
Color	Color to be used to draw the background or GUI_INVALID_COLOR to work in transparent mode.

Table 19.68: CHECKBOX_SetBkColor() parameter list

Additional information

If the check box should work in transparent mode GUI_INVALID_COLOR should be used.

CHECKBOX_SetBoxBkColor()

Before	After
	

Table 19.69: CHECKBOX_SetBoxBkColor() before after screenshots

Description

Sets the background color of the box area.

Prototype

```
GUI_COLOR CHECKBOX_SetBoxBkColor(CHECKBOX_Handle hObj,
                                 GUI_COLOR       Color,
                                 int             Index);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
Color	Color to be used.
Index	See table below.

Table 19.70: CHECKBOX_SetBoxBkColor() parameter list

Permitted values for parameter Index	
CHECKBOX_CI_DISABLED	Background color used for disabled state.
CHECKBOX_CI_ENABLED	Background color used for enabled state.

Return value

Previous background color.

Additional information

The color set by this function will only be visible, if the images used by the widget are transparent or no image is used. The default images of this widget are transparent.

CHECKBOX_SetDefaultBkColor()

Description

Sets the default background color used for new CHECKBOX widgets.

Prototype

```
void CHECKBOX_SetDefaultBkColor(GUI_COLOR Color);
```

Parameter	Description
`Color`	Color to be used, `GUI_INVALID_COLOR` for transparency.

Table 19.71: CHECKBOX_SetDefaultBkColor() parameter list

Additional information

For more information, refer to "CHECKBOX_SetBkColor()" on page 485.

CHECKBOX_SetDefaultFocusColor()

Description

Sets the color used to render the focus rectangle of new CHECKBOX widgets.

Prototype

```
GUI_COLOR CHECKBOX_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Description
`Color`	Color to be used.

Table 19.72: CHECKBOX_SetDefaultFocusColor() parameter list

Return value

Previous color used to render the focus rectangle.

Additional information

For mode information, refer to "CHECKBOX_SetFocusColor()" on page 489.

CHECKBOX_SetDefaultFont()

Description

Sets a pointer to a `GUI_FONT` structure used to display the text of new CHECKBOX widgets.

Prototype

```
void CHECKBOX_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
`pFont`	Pointer to GUI_FONT structure to be used.

Table 19.73: CHECKBOX_SetDefaultFont() parameter list

Additional information

For more information, refer to "CHECKBOX_SetFont()" on page 489.

CHECKBOX_SetDefaultImage()

Description

Sets the images used for new CHECKBOX widgets to be shown if they has been checked.

Prototype

```
void CHECKBOX_SetDefaultImage(const GUI_BITMAP * pBitmap,
                              unsigned int       Index);
```

Parameter	Description
pBitmap	Pointer to bitmap.
Index	See table below.

Table 19.74: CHECKBOX_SetDefaultImage() parameter list

Permitted values for parameter Index	
CHECKBOX_BI_INACTIV_UNCHECKED	Sets the bitmap displayed when the check box is unchecked and disabled.
CHECKBOX_BI_ACTIV_UNCHECKED	Sets the bitmap displayed when the CHECKBOX is unchecked and enabled.
CHECKBOX_BI_INACTIV_CHECKED	Sets the bitmap displayed when the CHECKBOX is checked and disabled.
CHECKBOX_BI_ACTIV_CHECKED	Sets the bitmap displayed when the CHECKBOX is checked and enabled.
CHECKBOX_BI_INACTIV_3STATE	Sets the bitmap displayed when the CHECKBOX is in the third state and disabled.
CHECKBOX_BI_ACTIV_3STATE	Sets the bitmap displayed when the CHECKBOX is in the third state and enabled.

Additional information

The image has to fill the complete inner area of the CHECKBOX widget.

CHECKBOX_SetDefaultSpacing()

Description

Sets the default spacing between box and text used to display the text of new CHECKBOX widgets.

Prototype

```
void CHECKBOX_SetDefaultSpacing(int Spacing);
```

Parameter	Description
Spacing	Number of pixels between box and text used for new CHECKBOX widgets.

Table 19.75: CHECKBOX_SetDefaultSpacing() parameter list

Additional information

For more information, refer to “CHECKBOX_SetSpacing()” on page 491.

CHECKBOX_SetDefaultTextAlign()

Description

Sets the default alignment used to display the text of new CHECKBOX widgets.

Prototype

```
void CHECKBOX_SetDefaultTextAlign(int Align);
```

Parameter	Description
Align	Text alignment used to display the text of new CHECKBOX widgets.

Table 19.76: CHECKBOX_SetDefaultTextAlign() parameter list

Additional information

For more information, refer to “CHECKBOX_SetTextAlign()” on page 492.

CHECKBOX_SetDefaultTextColor()

Description

Sets the default text color used to display the text of new CHECKBOX widgets.

Prototype

```
void CHECKBOX_SetDefaultTextColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color to be used.

Table 19.77: CHECKBOX_SetDefaultTextColor() parameter list

Additional information

For more information, refer to “CHECKBOX_SetTextColor()” on page 493.

CHECKBOX_SetFocusColor()

Before	After
	

Table 19.78: CHECKBOX_SetFocusColor() before after screenshots

Description

Sets the color used to render the focus rectangle.

Prototype

```
GUI_COLOR CHECKBOX_SetFocusColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.

Table 19.79: CHECKBOX_SetFocusColor() parameter list

Return value

Previous color of the focus rectangle.

Additional information

The focus rectangle is only visible if the widget has the input focus.

CHECKBOX_SetFont()

Description

Sets the font of the CHECKBOX widget.

Prototype

```
void CHECKBOX_SetFont(CHECKBOX_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
pFont	Pointer to the font.

Table 19.80: CHECKBOX_SetFont() parameter list

CHECKBOX_SetImage()

Before	After
	
	

Table 19.81: CHECKBOX_SetImage() before after screenshots

Description

Sets the images to be shown if the CHECKBOX widget has been checked.

Prototype

```
void CHECKBOX_SetImage(CHECKBOX_Handle    hObj,
                       const GUI_BITMAP * pBitmap,
                       unsigned int       Index);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
pBitmap	Pointer to bitmap.
Index	(see table shown under CHECKBOX_SetDefaultImage)

Table 19.82: CHECKBOX_SetImage() parameter list

Additional information

The image has to fill the complete inner area of the check box. If using this function make sure, the size of the check box used to create the widget is large enough to show the bitmap and (optional) the text.

CHECKBOX_SetNumStates()

Description

This function sets the number of possible states of the given CHECKBOX widget.

Prototype

```
void CHECKBOX_SetNumStates(CHECKBOX_Handle hObj, unsigned NumStates);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
NumStates	Number of possible states of the given check box. Currently supported are 2 or 3 states.

Table 19.83: CHECKBOX_SetNumStates() parameter list

Additional information

Per default a check box supports 2 states: checked (1) and unchecked (0). If the check box should support a third state the number of possible states can be increased to 3.

CHECKBOX_SetSpacing()

Before	After
	

Table 19.84: CHECKBOX_SetSpacing() before after screenshots

Description

Sets the number of pixels between box and text of a given CHECKBOX widget.

Prototype

```
void CHECKBOX_SetSpacing(CHECKBOX_Handle hObj, unsigned Spacing);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
Spacing	Number of pixels between box and text to be used.

Table 19.85: CHECKBOX_SetSpacing() parameter list

Additional information

The default spacing is 4 pixels. The function `CHECKBOX_SetDefaultSpacing()` or the configuration macro `CHECKBOX_SPACING_DEFAULT` can be used to set the default value.

CHECKBOX_SetState()

Before	After
	 

Table 19.86: CHECKBOX_SetState() before after screenshots

Description

Sets the new state of the given CHECKBOX widget.

Prototype

```
void CHECKBOX_SetState(CHECKBOX_Handle hObj, unsigned State);
```

Parameter	Description
hObj	Handle of CHECKBOX widget.
State	Zero based number of new state.

Table 19.87: CHECKBOX_SetState() parameter list

Permitted values for parameter State	
0	Unchecked
1	Checked
2	Third state

Additional information

The passed state should not be greater than the number of possible states set with `CHECKBOX_SetNumStates()` minus 1.

CHECKBOX_SetText()

Before	After
	

Table 19.88: CHECKBOX_SetText() before after screenshots

Description

Sets the optional text shown beside the box.

Prototype

```
void CHECKBOX_SetText(CHECKBOX_Handle hObj, const char * pText);
```

Parameter	Description
`hObj`	Handle of CHECKBOX widget.
`pText`	Pointer to text to be shown beside the box.

Table 19.89: CHECKBOX_SetText() parameter list

Additional information

Clicking on the text beside the box has the same effect as clicking into the box.

CHECKBOX_SetTextAlign()

Before	After
	

Table 19.90: CHECKBOX_SetTextAlign() before after screenshots

Description

Sets the alignment used to display the text beside the box.

Prototype

```
void CHECKBOX_SetTextAlign(CHECKBOX_Handle hObj, int Align);
```

Parameter	Description
`hObj`	Handle of CHECKBOX widget.
`Align`	Desired text alignment.

Table 19.91: CHECKBOX_SetTextAlign() parameter list

Additional information

Per default the text alignment is `GUI_TA_LEFT | GUI_TA_VCENTER`. The function `CHECKBOX_SetDefaultTextAlign()` and the configuration macro `CHECKBOX_TEXTALIGN_DEFAULT` can be used to set a user defined default value.

CHECKBOX_SetTextColor()

Before	After
	

Table 19.92: CHECKBOX_SetTextColor() before after screenshots

Description

Sets the color used to display the text of the CHECKBOX widget.

Prototype

```
void CHECKBOX_SetTextColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
`hObj`	Handle of CHECKBOX widget.
`Color`	Desired color of check box text.

Table 19.93: CHECKBOX_SetTextColor() parameter list

Additional information

Per default the text color of a check box text is `GUI_BLACK`. The function `CHECKBOX_SetDefaultTextColor()` and the configuration macro `CHECKBOX_TEXTCOLOR_DEFAULT` can be used to set a user defined default color.

CHECKBOX_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

CHECKBOX_Uncheck()

(Obsolete, `CHECKBOX_SetState()` should be used instead)

Before	After
	

Table 19.94: CHECKBOX_Uncheck() before after screenshots

Description

Sets the state of a specified CHECKBOX widget to unchecked.

Prototype

```
void CHECKBOX_Uncheck(CHECKBOX_Handle hObj);
```

Parameter	Description
`hObj`	Handle of CHECKBOX widget.

Table 19.95: CHECKBOX_Uncheck() parameter list

Additional information

This is the default setting for CHECKBOX widgets.

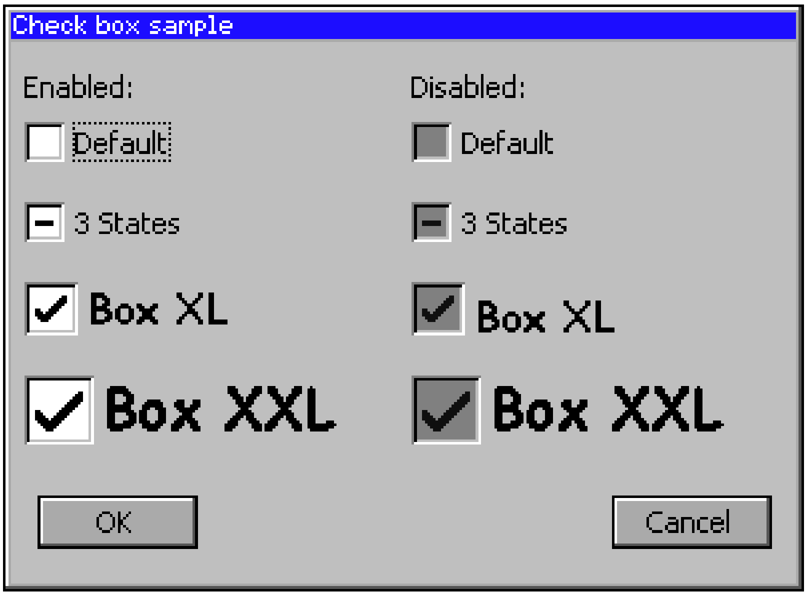
19.6.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_Checkbox.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Checkbox.c:



19.7 DROPDOWN: Dropdown widget

DROPDOWN widgets are used to select one element of a list with several columns. It shows the currently selected item in non open state. If the user opens a DROPDOWN widget a LISTBOX appears to select a new item.

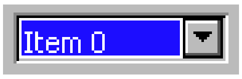
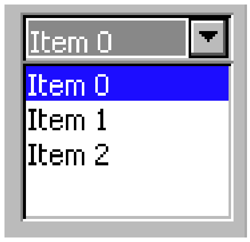
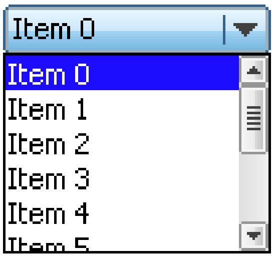
DROPDOWN closed	DROPDOWN opened
	

Table 19.96: Dropdown widget example

If mouse support is enabled, the open list reacts on moving the mouse over it.

Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter "Skinning" on page 871.

19.7.1 Configuration options

Type	Macro	Default	Description
N	DROPDOWN_ALIGN_DEFAULT	GUI_TA_LEFT	Text alignment used to display the dropdown text in closed state.
S	DROPDOWN_FONT_DEFAULT	&GUI_Font13_1	Default font
N	DROPDOWN_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	DROPDOWN_BKCOLOR1_DEFAULT	GUI_GRAY	Background color, selected state without focus.
N	DROPDOWN_BKCOLOR2_DEFAULT	GUI_BLUE	Background color, selected state with focus.
N	DROPDOWN_KEY_EXPAND	GUI_KEY_SPACE	Key which can be used to expand the DROPDOWN list.
N	DROPDOWN_KEY_SELECT	GUI_KEY_ENTER	Key which can be used to select an item from the open dropdown list.
N	DROPDOWN_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	DROPDOWN_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, selected state without focus.
N	DROPDOWN_TEXTCOLOR2_DEFAULT	GUI_WHITE	Enable 3D support.

Table 19.97: Configuration options

19.7.2 Predefined IDs

The following symbols define IDs which may be used to make DROPDOWN widgets distinguishable from creation: GUI_ID_DROPDOWN0 - GUI_ID_DROPDOWN3

19.7.3 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	DROPDOWN has been clicked.
WM_NOTIFICATION_RELEASED	DROPDOWN has been released.
WM_NOTIFICATION_MOVED_OUT	DROPDOWN has been clicked and pointer has been moved out of the widget without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scroll bar of the opened DROPDOWN widget has been changed.
WM_NOTIFICATION_SEL_CHANGED	The selection of the DROPDOWN list has been changed.

Table 19.98: Notification codes

19.7.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_ENTER	Selects an item from the open DROPDOWN list and closes the list.
GUI_KEY_SPACE	Opens the DROPDOWN list.

Table 19.99: Keyboard reaction

19.7.5 DROPDOWN API

The table below lists the available emWin DROPDOWN-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
DROPDOWN_AddString()	Adds an element to the DROPDOWN widget.
DROPDOWN_Collapse()	Closes the DROPDOWN list.
DROPDOWN_Create()	Creates a DROPDOWN widget. (Obsolete)
DROPDOWN_CreateEx()	Creates a DROPDOWN widget.
DROPDOWN_CreateIndirect()	Creates a DROPDOWN widget from a resource table entry.
DROPDOWN_CreateUser()	Creates a DROPDOWN widget using extra bytes as user data.
DROPDOWN_DecSel()	Decrements selection.
DROPDOWN_DecSelExp()	Decrements selection in expanded state.
DROPDOWN_DeleteItem()	Deletes an item of the DROPDOWN list.
DROPDOWN_Expand()	Opens the DROPDOWN list.
DROPDOWN_GetDefaultFont()	Returns the default font used to create DROPDOWN widgets.
DROPDOWN_GetItemDisabled()	Returns the state of the given item.
DROPDOWN_GetItemText()	Returns the text of a specific DROPDOWN item.
DROPDOWN_GetListbox()	Returns the handle of the attached LISTBOX in expanded state.
DROPDOWN_GetNumItems()	Returns the number of items in the DROPDOWN list.
DROPDOWN_GetSel()	Returns the number of the currently selected element.
DROPDOWN_GetSelExp()	Returns the number of the currently selected element in expanded state.

Table 19.100: Dropdown API list

Routine	Description
DROPDOWN_GetUserData()	Retrieves the data set with DROPDOWN_SetUserData().
DROPDOWN_IncSel()	Increments selection.
DROPDOWN_IncSelExp()	Increments selection in expanded state.
DROPDOWN_InsertString()	Inserts a string to the DROPDOWN list.
DROPDOWN_SetAutoScroll()	Enables the automatic use of a scroll bar in the DROPDOWN list.
DROPDOWN_SetBkColor()	Sets the background color.
DROPDOWN_SetColor()	Sets the color of the arrow and the button of the DROPDOWN widget.
DROPDOWN_SetDefaultColor()	Sets the default color for arrow and button of DROPDOWN widgets.
DROPDOWN_SetDefaultFont()	Sets the default font for DROPDOWN widgets.
DROPDOWN_SetDefaultScrollbarColor()	Sets the default colors of the optional scroll bar in the DROPDOWN list.
DROPDOWN_SetFont()	Sets the font of the given DROPDOWN widget
DROPDOWN_SetItemDisabled()	Sets the state of the given item.
DROPDOWN_SetItemSpacing()	Sets the spacing between the items of the DROP-DOWN list.
DROPDOWN_SetListHeight()	Sets the height in pixels used for expanding the dropdown list.
DROPDOWN_SetScrollbarColor()	Sets the colors of the scroll bar in the DROPDOWN list.
DROPDOWN_SetScrollbarWidth()	Sets the scroll bar width of the used by the DROP-DOWN widget.
DROPDOWN_SetSel()	Sets the current selection.
DROPDOWN_SetSelExp()	Sets the current selection in expanded state.
DROPDOWN_SetTextAlign()	Sets the text alignment used to display the DROP-DOWN text in closed state.
DROPDOWN_SetTextColor()	Sets the text color of the given DROPDOWN widget.
DROPDOWN_SetTextHeight()	Sets the height of the rectangle used to display the dropdown text in closed state.
DROPDOWN_SetUpMode()	Enables the up mode for the given widget.
DROPDOWN_SetUserData()	Sets the extra data of a DROPDOWN widget.

Table 19.100: Dropdown API list

DROPDOWN_AddString()

Description

Adds a new element to the DROPDOWN widget.

Prototype

```
void DROPDOWN_AddString(DROPDOWN_Handle hObj, const char * s);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
s	Pointer to string to be added

Table 19.101: DROPDOWN_AddString() parameter list

DROPDOWN_Collapse()

Description

Closes the dropdown list of the DROPDOWN widget.

Prototype

```
void DROPDOWN_Collapse(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.102: DROPDOWN_Collapse() parameter list

DROPDOWN_Create()

(Obsolete, DROPDOWN_CreateEx() should be used instead)

Description

Creates a DROPDOWN widget of a specified size at a specified location.

Prototype

```
DROPDOWN_Handle DROPDOWN_Create(WM_HWIN hWinParent,
                                int     x0,         int y0,
                                int     xSize,      int ySize,
                                int     Flags);
```

Parameter	Description
hWinParent	Handle of parent window
x0	Leftmost pixel of the DROPDOWN widget (in parent coordinates).
y0	Topmost pixel of the DROPDOWN widget (in parent coordinates).
xSize	Horizontal size of the DROPDOWN widget (in pixels).
ySize	Vertical size of the DROPDOWN widget in open state (in pixels).
Flags	Window create flags. Typically, WM_CF_SHOW to make the widget visible immediately (refer to “WM_CreateWindow()” on page 404 for a list of available parameter values).

Table 19.103: DROPDOWN_Create() parameter list

Return value

Handle of the created DROPDOWN widget; 0 if the function fails.

Additional information

The ySize of the widget in closed state depends on the font used to create the widget. You can not set the ySize of a closed DROPDOWN widget.

DROPDOWN_CreateEx()

Description

Creates a DROPDOWN widget of a specified size at a specified location.

Prototype

```
DROPDOWN_Handle DROPDOWN_CreateEx(int     x0,      int y0,
                                  int     xSize,   int ySize,
                                  WM_HWIN hParent, int WinFlags,
                                  int     ExFlags, int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget in open state (in pixels).
hParent	Handle of parent window. If 0, the new DROPDOWN widget will be a child of the desktop (top-level window).

Table 19.104: DROPDOWN_CreateEx() parameter list

Parameter	Description
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	See table below.
Id	Window ID of the widget.

Table 19.104: DROPDOWN_CreateEx() parameter list

Permitted values for parameter ExFlags	
0	No function.
DROPDOWN_CF_AUTOSCROLLBAR	Enable automatic use of a scroll bar. For details, refer to "DROPDOWN_SetAutoScroll()" on page 503.
DROPDOWN_CF_UP	Creates a DROPDOWN widget which opens the dropdown list above the widget. This flag is useful if the space below the widget is not sufficient for the dropdown list.

Return value

Handle of the created DROPDOWN widget; 0 if the function fails.

DROPDOWN_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `DROPDOWN_CreateEx()`.

DROPDOWN_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `DROPDOWN_CreateEx()` can be referred to.

DROPDOWN_DecSel()

Description

Decrement the selection, moves the selection of a specified DROPDOWN widget up by one item.

Prototype

```
void DROPDOWN_DecSel(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.105: DROPDOWN_DecSel() parameter list

DROPDOWN_DecSelExp()

Description

Decrements the selection of the attached LISTBOX in expanded state.

Prototype

```
void DROPDOWN_DecSelExp(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.106: DROPDOWN_DecSelExp() parameter list

DROPDOWN_DeleteItem()

Description
Deletes the given item of the DROPDOWN widget.

Prototype
```
void DROPDOWN_DeleteItem(DROPDOWN_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of DROPDOWN widget.
Index	Zero based index of the item to be deleted.

Table 19.107: DROPDOWN_DeleteItem() parameter list

Additional information
If the index is greater than the number of items < 1 the function returns immediately.

DROPDOWN_Expand()

Description
Opens the list of the DROPDOWN widget.

Prototype
```
void DROPDOWN_Expand(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget.

Table 19.108: DROPDOWN_Expand() parameter list

Additional information
The DROPDOWN list remains open until an element has been selected or the focus has been lost.

DROPDOWN_GetDefaultFont()

Description
Returns the default font of DROPDOWN widgets.

Prototype
```
const GUI_FONT * DROPDOWN_GetDefaultFont(void);
```

Return value
Returns a pointer to the default font used by DROPDOWN widgets.

DROPDOWN_GetItemDisabled()

Description
Returns the state of the given item.

Prototype
```
unsigned DROPDOWN_GetItemDisabled(DROPDOWN_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Index	Zero-based index of the item.

Table 19.109: DROPDOWN_GetItemDisabled() parameter list

Return value

1 if the given item is disabled, 0 if not.

DROPDOWN_GetItemText()

Description

Returns the text of a specific item of the given DROPDOWN widget.

Prototype

```
int DROPDOWN_GetItemText(DROPDOWN_Handle   hObj,    unsigned Index,
                         char            * pBuffer, int      MaxSize);
```

Parameter	Description
hObj	Handle of DROPDOWN widget.
Index	Zero-based index of the item.
pBuffer	Pointer to a char buffer which is filled with the text.
MaxSize	Maximum number of chars which can be stored by pBuffer.

Table 19.110: DROPDOWN_GetItemText() parameter list

Return value

0 on success, 1 on error.

DROPDOWN_GetListbox()

Description

Returns the handle of the attached LISTBOX widget in expanded state.

Prototype

```
LISTBOX_Handle DROPDOWN_GetListbox(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.111: DROPDOWN_GetListbox() parameter list

Return value

Handle of the attached LISTBOX widget in expanded state, 0 if DROPDOWN is in collapsed state.

DROPDOWN_GetNumItems()

Description

Returns the number of items in the given DROPDOWN widget.

Prototype

```
int DROPDOWN_GetNumItems(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.112: DROPDOWN_GetNumItems() parameter list

Return value

Number of items in the given DROPDOWN widget.

DROPDOWN_GetSel()

Description

Returns the number of the currently selected item in a specified DROPDOWN widget.

Prototype

```
int DROPDOWN_GetSel(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.113: DROPDOWN_GetSel() parameter list

Return value

Number of the currently selected item.

DROPDOWN_GetSelExp()

Description

Returns the index of the currently selected item of the attached LISTBOX in expanded state.

Prototype

```
int DROPDOWN_GetSelExp(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.114: DROPDOWN_GetSelExp() parameter list

Return value

Index of the currently selected item.

DROPDOWN_GetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_GetUserData().

DROPDOWN_IncSel()

Description

Increment the selection, moves the selection of a specified DROPDOWN widget down by one item.

Prototype

```
void DROPDOWN_IncSel(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.115: DROPDOWN_IncSel() parameter list

DROPDOWN_IncSelExp()

Description

Increments the selection of the attached LISTBOX in expanded state.

Prototype

```
void DROPDOWN_IncSelExp(DROPDOWN_Handle hObj);
```

Parameter	Description
hObj	Handle of DROPDOWN widget

Table 19.116: DROPDOWN_IncSelExp() parameter list

DROPDOWN_InsertString()

Description

Inserts a string to the DROPDOWN widget at the given position.

Prototype

```
void DROPDOWN_InsertString(DROPDOWN_Handle   hObj,
                           const char      * s,
                           unsigned int      Index);
```

Parameter	Description
hObj	Handle of DROPDOWN widget.
s	Pointer to the string to be inserted.
Index	Zero based index of the position.

Table 19.117: DROPDOWN_InsertString() parameter list

Additional information

If the given index is greater than the number of items the string will be appended to the end of the dropdown list.

DROPDOWN_SetAutoScroll()

Description

Enables the automatic use of a vertical scroll bar in the DROPDOWN widget.

Prototype

```
void DROPDOWN_SetAutoScroll(DROPDOWN_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle of DROPDOWN widget.
OnOff	See table below.

Table 19.118: DROPDOWN_SetAutoScroll() parameter list

Permitted values for parameter OnOff	
0	Disable automatic use of a scroll bar.
1	Enable automatic use of a scroll bar.

Additional information

If enabled the DROPDOWN widget checks if all elements fits into the listbox. If not a vertical scroll bar will be added.

DROPDOWN_SetBkColor()

Before	After
	

Table 19.119: DROPDOWN_SetBkColor() before after screenshots

Description

Sets the background color of the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetBkColor(DROPDOWN_Handle hObj,
                         unsigned int    Index,
                         GUI_COLOR       Color);
```

Parameter	Description
`hObj`	Handle of DROPDOWN widget
`Index`	Index for background color. See table below.
`Color`	Color to be set.

Table 19.120: DROPDOWN_SetBkColor() parameter list

Permitted values for parameter `Index`	
`DROPDOWN_CI_UNSEL`	Unselected element.
`DROPDOWN_CI_SEL`	Selected element, without focus.
`DROPDOWN_CI_SELFOCUS`	Selected element, with focus.

DROPDOWN_SetColor()

Before	After
	

Table 19.121: DROPDOWN_SetColor() before after screenshots

Description

Sets the color of the button or the arrow of the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetColor(DROPDOWN_Handle hObj,
                       unsigned int    Index,
                       GUI_COLOR       Color);
```

Parameter	Description
`hObj`	Handle of DROPDOWN widget
`Index`	Index of desired item. See table below.
`Color`	Color to be used.

Table 19.122: DROPDOWN_SetColor() parameter list

Permitted values for parameter `Index`	
`DROPDOWN_CI_ARROW`	Color of small arrow within the button.
`DROPDOWN_CI_BUTTON`	Button color.

DROPDOWN_SetDefaultColor()

Description

Sets the default colors for the arrow and the button of new DROPDOWN widgets.

Prototype

```
GUI_COLOR DROPDOWN_SetDefaultColor(int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Refer to "DROPDOWN_SetColor()" on page 504.
Color	Color to be used.

Table 19.123: DROPDOWN_SetDefaultColor() parameter list

DROPDOWN_SetDefaultFont()

Description

Sets the default font used for new DROPDOWN widgets.

Prototype

```
void DROPDOWN_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to GUI_FONT structure.

Table 19.124: DROPDOWN_SetDefaultFont() parameter list

DROPDOWN_SetDefaultScrollbarColor()

Description

Sets the default colors used for the optional scroll bar in the DROPDOWN widget.

Prototype

```
GUI_COLOR DROPDOWN_SetDefaultScrollbarColor(int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Refer to "DROPDOWN_SetScrollbarColor()" on page 507.
Color	Color to be used.

Table 19.125: DROPDOWN_SetDefaultScrollbarColor() parameter list

DROPDOWN_SetFont()

Description

Sets the font used to display the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetFont(DROPDOWN_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
pFont	Pointer to the font.

Table 19.126: DROPDOWN_SetFont() parameter list

DROPDOWN_SetItemDisabled()

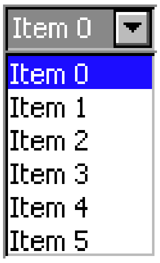
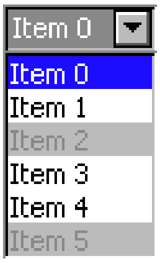
Before	After
	

Table 19.127: DROPDOWN_SetItemDisabled() before after screenshots

Description

Sets the enabled state of the given item.

Prototype

```
void DROPDOWN_SetItemDisabled(DROPDOWN_Handle hObj,
                              unsigned        Index,
                              int             OnOff);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Index	Zero-based index of the item.
OnOff	1 for enabled, 0 for disabled.

Table 19.128: DROPDOWN_SetItemDisabled() parameter list

DROPDOWN_SetListHeight()

Description

Sets the height in pixels to be used for the expanded list of the DROPDOWN widget.

Prototype

```
int DROPDOWN_SetListHeight(DROPDOWN_Handle hObj, unsigned Height);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Height	Height to be used.

Table 19.129: DROPDOWN_SetListHeight() parameter list

Return value

Previously used height for the DROPDOWN list in pixels.

DROPDOWN_SetScrollbarColor()

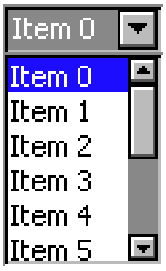
Before	After
	

Table 19.130: DROPDOWN_SetScrollbarColor() before after screenshots

Description

Sets the colors of the optional scroll bar in the DROPDOWN widget.

Prototype

```
void DROPDOWN_SetScrollbarColor(DROPDOWN_Handle hObj,
                                unsigned int    Index,
                                GUI_COLOR       Color);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Index	Index of desired item. See table below.
Color	Color to be used.

Table 19.131: DROPDOWN_SetScrollbarColor() parameter list

Permitted values for parameter Index	
SCROLLBAR_CI_THUMB	Color of thumb area.
SCROLLBAR_CI_SHAFT	Color of shaft.
SCROLLBAR_CI_ARROW	Color of arrows.

DROPDOWN_SetScrollbarWidth()

Description

Sets the width of the scroll bars used by the list of the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetScrollbarWidth(DROPDOWN_Handle hObj, unsigned Width);
```

Parameter	Description
hObj	Handle of DROPDOWN widget.
Width	Width of the scroll bar(s) used by the dropdown list of the given DROPDOWN widget.

Table 19.132: DROPDOWN_SetScrollbarWidth() parameter list

DROPDOWN_SetSel()

Description

Sets the selected item of a specified DROPDOWN widget.

Prototype

```
void DROPDOWN_SetSel(DROPDOWN_Handle hObj, int Sel);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Sel	Element to be selected.

Table 19.133: DROPDOWN_SetSel() parameter list

DROPDOWN_SetSelExp()

Description

Sets the selected item of the attached LISTBOX in expanded state.

Prototype

```
void DROPDOWN_SetSelExp(DROPDOWN_Handle hObj, int Sel);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Sel	Element to be selected.

Table 19.134: DROPDOWN_SetSelExp() parameter list

DROPDOWN_SetItemSpacing()

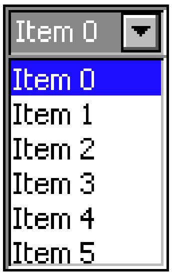
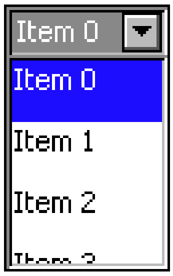
Before	After
	

Table 19.135: DROPDOWN_SetItemSpacing() before after screenshots

Description

Sets an additional spacing below the items of the DROPDOWN widget.

Prototype

```
void DROPDOWN_SetItemSpacing(DROPDOWN_Handle hObj, unsigned Value);
```

Parameter	Description
hObj	Handle of DROPDOWN widget
Value	Number of pixels used as additional space between the items of the DROPDOWN widget.

Table 19.136: DROPDOWN_SetItemSpacing() parameter list

DROPDOWN_SetTextAlign()

Before	After
	

Table 19.137: DROPDOWN_SetTextAlign() before after screenshots

Description

Sets the alignment used to display the text of the DROPDOWN widget in closed state.

Prototype

```
void DROPDOWN_SetTextAlign(DROPDOWN_Handle hObj, int Align);
```

Parameter	Description
`hObj`	Handle of DROPDOWN widget
`Align`	Alignment used to display the dropdown text in closed state.

Table 19.138: DROPDOWN_SetTextAlign() parameter list

DROPDOWN_SetTextColor()

Description

Sets the background color of the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetTextColor(DROPDOWN_Handle hObj,   unsigned int Index,
                           GUI_COLOR       Color);
```

Parameter	Description
`hObj`	Handle of DROPDOWN widget
`Index`	Index for background color. See table below.
`Color`	Color to be set.

Table 19.139: DROPDOWN_SetTextColor() parameter list

Permitted values for parameter `Index`	
`DROPDOWN_CI_UNSEL`	Unselected element.
`DROPDOWN_CI_SEL`	Selected element, without focus.
`DROPDOWN_CI_SELFOCUS`	Selected element, with focus.

DROPDOWN_SetTextHeight()

Before	After
	

Table 19.140: DROPDOWN_SetTextHeight() before after screenshots

Description

Sets the height of the rectangle used to display the text of DROPDOWN widget in closed state.

Prototype

```
void DROPDOWN_SetTextHeight(DROPDOWN_Handle hObj, unsigned TextHeight);
```

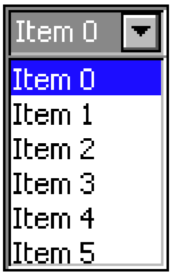
Parameter	Description
`hObj`	Handle of DROPDOWN widget
`TextHeight`	Height of the rectangle used to display the text in closed state.

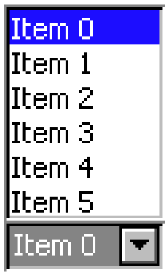
Table 19.141: DROPDOWN_SetTextHeight() parameter list

Additional information

Per default the height of the DROPDOWN widget depends on the used font. Using this function with `TextHeight` > 0 means the given value should be used. `Text Height` = 0 means the default behavior should be used.

DROPDOWN_SetUpMode()

<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>


</td><td>


</td></tr>
</table>

Table 19.142: DROPDOWN_SetUpMode() before after screenshots

Description

Enables opening of the list to the upper side of the DROPDOWN widget.

Prototype

```
int DROPDOWN_SetUpMode(DROPDOWN_Handle hObj, int OnOff);
```

Parameter	Description
`hObj`	Handle of DROPDOWN widget
`OnOff`	1 for enabling, 0 for disabling 'up mode'.

Table 19.143: DROPDOWN_SetUpMode() parameter list

DROPDOWN_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

19.7.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_Dropdown.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Dropdown.c:

19.8 EDIT: Edit widget

EDIT widgets are commonly used as the primary user interface for text input:

Blank EDIT widget	EDIT widget with user input
	

Table 19.144: Edit widget example

You can also use EDIT widgets for entering values in binary, decimal, or hexadecimal modes. A decimal-mode edit field might appear similar to those in the following table. The background color of EDIT widgets by default turns gray if disabled:

EDIT widget with user input (decimal)	Disabled EDIT widget
	

Table 19.145: Edit widget diabled edit field example

All EDIT-related routines are located in the file(s) `EDIT*.c`, `EDIT.h`. All identifiers are prefixed EDIT.

19.8.1 Configuration options

Type	Macro	Default	Description
S	`EDIT_ALIGN_DEFAULT`	GUI_TA_RIGHT | GUI_TA_VCENTER	Text alignment for EDIT widget.
N	`EDIT_BKCOLOR0_DEFAULT`	0xc0c0c0	Background color, disabled state.
N	`EDIT_BKCOLOR1_DEFAULT`	GUI_WHITE	Background color, enabled state.
N	`EDIT_BORDER_DEFAULT`	1	Width of border, in pixels.
S	`EDIT_FONT_DEFAULT`	&GUI_Font13_1	Font used for edit field text.
N	`EDIT_TEXTCOLOR0_DEFAULT`	GUI_BLACK	Text color, disabled state.
N	`EDIT_TEXTCOLOR1_DEFAULT`	GUI_BLACK	Text color, enabled state.
N	`EDIT_XOFF`	2	Distance in X to offset text from left border of EDIT widget.

Table 19.146: Configuration options

Available alignment flags are:
`GUI_TA_LEFT`, `GUI_TA_RIGHT`, `GUI_TA_HCENTER` for horizontal alignment.
`GUI_TA_TOP`, `GUI_TA_BOTTOM`, `GUI_TA_VCENTER` for vertical alignment.

19.8.2 Predefined IDs

The following symbols define IDs which may be used to make EDIT widgets distinguishable from creation: `GUI_ID_EDIT0` - `GUI_ID_EDIT9`

19.8.3 Notification codes

The following events are sent from an EDIT widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Value (content) of the EDIT widget has changed.

Table 19.147: Notification codes

19.8.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_UP	Increases the current character. If for example the current character (the character below the cursor) is a 'A' it changes to 'B'.
GUI_KEY_DOWN	Decreases the current character. If for example the current character is a 'B' it changes to 'A'.
GUI_KEY_RIGHT	Moves the cursor one character to the right.
GUI_KEY_LEFT	Moves the cursor one character to the left.
GUI_KEY_BACKSPACE	If the widget works in text mode, the character before the cursor is deleted.
GUI_KEY_DELETE	If the widget works in text mode, the current is deleted.
GUI_KEY_INSERT	If the widget works in text mode, this key toggles the edit mode between GUI_EDIT_MODE_OVERWRITE and GUI_EDIT_MODE_INSERT. For details, refer to "EDIT_SetInsertMode()" on page 525.

Table 19.148: Keyboard reaction

19.8.5 EDIT API

The table below lists the available emWin EDIT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
EDIT_AddKey()	Key input routine.
EDIT_Create()	Creates an EDIT widget. (Obsolete)
EDIT_CreateAsChild()	Creates an EDIT widget as a child window. (Obsolete)
EDIT_CreateEx()	Creates an EDIT widget.
EDIT_CreateIndirect()	Creates an EDIT widget from resource table entry.
EDIT_CreateUser()	Creates an EDIT widget using extra bytes as user data.
EDIT_EnableBlink()	Enables/disables a blinking cursor
EDIT_GetBkColor()	Returns the background color of the EDIT widget.
EDIT_GetCursorCharPos()	Returns the number of the character at the cursor position.
EDIT_GetCursorPixelPos()	Returns the pixel position of the cursor.
EDIT_GetDefaultBkColor()	Returns the default background color.
EDIT_GetDefaultFont()	Returns the default font.
EDIT_GetDefaultTextAlign()	Returns the default text alignment.
EDIT_GetDefaultTextColor()	Returns the default text color.
EDIT_GetFloatValue()	Returns the current value as floating point value.
EDIT_GetFont()	Returns a pointer to the used font.
EDIT_GetNumChars	Returns the number of characters of the given EDIT widget.

Table 19.149: Edit API list

Routine	Description
EDIT_GetText()	Returns the user input.
EDIT_GetTextColor()	Returns the text color.
EDIT_GetUserData()	Retrieves the data set with EDIT_SetUserData().
EDIT_GetValue()	Returns the current value.
EDIT_SetBinMode()	Enables the binary edit mode.
EDIT_SetBkColor()	Sets the background color of the EDIT widget.
EDIT_SetCursorAtChar()	Sets the EDIT widget cursor to a specified character position.
EDIT_SetCursorAtPixel()	Sets the EDIT widget cursor to a specified pixel position.
EDIT_SetDecMode()	Enables the decimal edit mode.
EDIT_SetDefaultBkColor()	Sets the default background color.
EDIT_SetDefaultFont()	Sets the default font used for EDIT widgets.
EDIT_SetDefaultTextAlign()	Sets the default text alignment for EDIT widgets.
EDIT_SetDefaultTextColor()	Sets the default text color for EDIT widgets.
EDIT_SetFloatMode()	Enables the floating point edit mode.
EDIT_SetFloatValue()	Sets the floating point value if using the floating point edit mode.
EDIT_SetFocussable()	Sets focussability of the EDIT widget.
EDIT_SetFont()	Selects the font to be used.
EDIT_SetHexMode()	Enables the hexadecimal edit mode.
EDIT_SetInsertMode()	Enables or disables the insert mode.
EDIT_SetMaxLen()	Sets the maximum number of characters of the EDIT widget.
EDIT_SetpfAddKeyEx()	Sets a function which is called to add a character.
EDIT_SetSel()	Sets the current selection.
EDIT_SetText()	Sets the text.
EDIT_SetTextAlign()	Sets the text alignment for the EDIT widget.
EDIT_SetTextColor()	Sets the color(s) for the text.
EDIT_SetTextMode()	Sets the edit mode of the EDIT widget back to text mode.
EDIT_SetValue()	Sets the current value.
EDIT_SetUlongMode()	Enables the unsigned long decimal edit mode.
EDIT_SetUserData()	Sets the extra data of an EDIT widget.
GUI_EditBin()	Edits a binary value at the current cursor position.
GUI_EditDec()	Edits a decimal value at the current cursor position.
GUI_EditFloat()	Edits a floating point value at the current cursor position.
GUI_EditHex()	Edits a hexadecimal value at the current cursor position.
GUI_EditString()	Edits a string at the current cursor position.

Table 19.149: Edit API list

EDIT_AddKey()

Description

Adds user input to a specified EDIT widget.

Prototype

```
void EDIT_AddKey(EDIT_Handle hObj, int Key);
```

Parameter	Description
hObj	Handle of EDIT widget.
Key	Character to be added.

Table 19.150: EDIT_AddKey() parameter list

Additional information

The specified character is added to the user input of the EDIT widget. If the last character should be erased, the key GUI_KEY_BACKSPACE can be used. If the maximum count of characters has been reached, another character will not be added.

EDIT_Create()

(Obsolete, `EDIT_CreateEx()` should be used instead)

Description

Creates an EDIT widget of a specified size at a specified location.

Prototype

```
EDIT_Handle EDIT_Create(int x0, int y0,     int xSize,  int ySize,
                        int Id, int MaxLen, int Flags);
```

Parameter	Description
x0	Leftmost pixel of the edit field (in parent coordinates).
y0	Topmost pixel of the edit field (in parent coordinates).
xSize	Horizontal size of the edit field (in pixels).
ySize	Vertical size of the edit field (in pixels.
Id	ID to be returned.
MaxLen	Maximum count of characters.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).

Table 19.151: EDIT_Create() parameter list

Return value

Handle of the created EDIT widget; 0 if the function fails.

EDIT_CreateAsChild()

(Obsolete, `EDIT_CreateEx` should be used instead)

Description

Creates an EDIT widget as a child window.

Prototype

```
EDIT_Handle EDIT_CreateAsChild(int     x0,      int y0,
                               int     xSize,   int ySize,
                               WM_HWIN hParent, int Id,
                               int     Flags,   int MaxLen);
```

Parameter	Description
x0	X-position of the edit field relative to the parent window.
y0	Y-position of the edit field relative to the parent window.
xSize	Horizontal size of the edit field (in pixels).
ySize	Vertical size of the edit field (in pixels).
hParent	Handle of parent window.
Id	ID to be assigned to the EDIT widget.
Flags	Window create flags (see `EDIT_Create()`).
MaxLen	Maximum count of characters.

Table 19.152: EDIT_CreateAsChild() parameter list

Return value

Handle of the created EDIT widget; 0 if the function fails.

EDIT_CreateEx()

Description

Creates an EDIT widget of a specified size at a specified location.

Prototype

```
EDIT_Handle EDIT_CreateEx(int      x0,       int y0,
                          int      xSize,    int ySize,
                          WM_HWIN hParent,  int WinFlags,
                          int      ExFlags,  int Id,
                          int      MaxLen);
```

Parameter	Description
`x0`	Leftmost pixel of the widget (in parent coordinates).
`y0`	Topmost pixel of the widget (in parent coordinates).
`xSize`	Horizontal size of the widget (in pixels).
`ySize`	Vertical size of the widget (in pixels).
`hParent`	Handle of parent window. If 0, the new EDIT widget will be a child of the desktop (top-level window).
`WinFlags`	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
`ExFlags`	Not used, reserved for future use.
`Id`	Window ID of the widget.
`MaxLen`	Maximum count of characters.

Table 19.153: EDIT_CreateEx() parameter list

Return value

Handle of the created EDIT widget; 0 if the function fails.

EDIT_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Flags` is used according to the parameter `Align` of the function `EDIT_SetTextAlign()`. The element `Para` is used according to the parameter `MaxLen` of the function `EDIT_CreateEx()`.

EDIT_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `EDIT_CreateEx()` can be referred to.

EDIT_EnableBlink()

Description

Enables/disables a blinking cursor.

Prototype

```
void EDIT_EnableBlink(EDIT_Handle hObj, int Period, int OnOff);
```

Parameter	Description
`hObj`	Handle of EDIT widget.
`Period`	Blinking period
`OnOff`	1 enables blinking, 0 disables blinking

Table 19.154: EDIT_EnableBlink() parameter list

Additional information

This function calls `GUI_X_GetTime()`.

EDIT_GetBkColor()

Description

Returns the background color of the EDIT widget.

Prototype

```
GUI_COLOR EDIT_GetBkColor(EDIT_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of EDIT widget.
Index	Color index. See table below.

Table 19.155: EDIT_GetBkColor() parameter list

Permitted values for parameter Index	
EDIT_CI_DISABLED	Color index for the disabled state.
EDIT_CI_ENABLED	Color index for the enabled state.

Return value

Background color of the EDIT widget.

EDIT_GetCursorCharPos()

Description

Returns the character related position of the cursor.

Prototype

```
int EDIT_GetCursorCharPos(EDIT_Handle hObj);
```

Parameter	Description
hObj	Handle of EDIT widget.

Table 19.156: EDIT_GetCursorCharPos() parameter list

Return value

Character related position of the cursor.

Additional information

The widget returns the character position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

EDIT_GetCursorPixelPos()

Description

Returns the pixel related position of the cursor in window coordinates.

Prototype

```
void EDIT_GetCursorPixelPos(EDIT_Handle hObj, int * pxPos, int * pyPos);
```

Parameter	Description
hObj	Handle of EDIT widget.
pxPos	Pointer to integer variable for the X-position in window coordinates.
pyPos	Pointer to integer variable for the Y-position in window coordinates.

Table 19.157: EDIT_GetCursorPixelPos() parameter list

Return value

Pixel related position of the cursor in window coordinates.

Additional information

The widget returns the pixel position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

EDIT_GetDefaultBkColor()

Description

Returns the default background color used for EDIT widgets.

Prototype

```
GUI_COLOR EDIT_GetDefaultBkColor(unsigned int Index);
```

Parameter	Description
Index	Color index. See table below.

Table 19.158: EDIT_GetDefaultBkColor() parameter list

Permitted values for parameter Index	
EDIT_CI_DISABLED	Color index for the disabled state.
EDIT_CI_ENABLED	Color index for the enabled state.

Return value

Default background color used for EDIT widgets.

EDIT_GetDefaultFont()

Description

Returns the default font used for EDIT widgets.

Prototype

```
const GUI_FONT * EDIT_GetDefaultFont(void);
```

Return value

Default font used for EDIT widgets.

EDIT_GetDefaultTextAlign()

Description

Returns the default text alignment used for EDIT widgets.

Prototype

```
int EDIT_GetDefaultTextAlign(void);
```

Return value

Default text alignment used for EDIT widgets.

EDIT_GetDefaultTextColor()

Description

Returns the default text color used for EDIT widgets.

Prototype

```
GUI_COLOR EDIT_GetDefaultTextColor(unsigned int Index);
```

Parameter	Description
Index	Has to be 0, reserved for future use.

Table 19.159: EDIT_GetDefaultTextColor() parameter list

Return value

Default text color used for EDIT widgets.

EDIT_GetFloatValue()

Description

Returns the current value of the EDIT widget as floating point value.

Prototype

```
float EDIT_GetFloatValue(EDIT_Handle hObj);
```

Parameter	Description
hObj	Handle of EDIT widget.

Table 19.160: EDIT_GetFloatValue() parameter list

Return value

The current value.

Additional information

The use of this function makes only sense if the edit field is in floating point edit mode.

EDIT_GetFont()

Description

Returns a pointer to the used font.

Prototype

```
const GUI_FONT * EDIT_GetFont(EDIT_Handle hObj);
```

Parameter	Description
hObj	Handle of EDIT widget.

Table 19.161: EDIT_GetFont() parameter list

Return value

Pointer to the used font.

EDIT_GetNumChars

Description

Returns the number of characters of the specified EDIT widget.

Prototype

```
int EDIT_GetNumChars(EDIT_Handle hObj);
```

Parameter	Description
hObj	Handle of EDIT widget.

Table 19.162: EDIT_GetNumChars() parameter list

Return value

Number of characters of the specified EDIT widget.

EDIT_GetText()

Description

Retrieves the user input of a specified EDIT widget.

Prototype

```
void EDIT_GetText(EDIT_Handle hObj, char * sDest, int MaxLen);
```

Parameter	Description
hObj	Handle of EDIT widget.
sDest	Pointer to buffer.
MaxLen	Size of buffer.

Table 19.163: EDIT_GetText() parameter list

EDIT_GetTextColor()

Description

Returns the text color.

Prototype

```
GUI_COLOR EDIT_GetTextColor(EDIT_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of EDIT widget.
Index	Color index. See table below.

Table 19.164: EDIT_GetTextColor() parameter list

Permitted values for parameter Index	
EDIT_CI_DISABLED	Color index for the disabled state.
EDIT_CI_ENABLED	Color index for the enabled state.

EDIT_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

EDIT_GetValue()

Description

Returns the current value of the EDIT widget. The current value is only useful if the EDIT widget is in binary, decimal or hexadecimal mode.

Prototype

```
I32  EDIT_GetValue(EDIT_Handle hObj);
```

Parameter	Description
hObj	Handle of EDIT widget.

Table 19.165: EDIT_GetValue() parameter list

Return value

The current value.

EDIT_SetBinMode()

Description

Enables the binary edit mode of the EDIT widget. The given value can be modified in the given range.

Prototype

```
void EDIT_SetBinMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.

Table 19.166: EDIT_SetBinMode() parameter list

EDIT_SetBkColor()

Description

Sets the background color of the EDIT widget.

Prototype

```
void EDIT_SetBkColor(EDIT_Handle hObj, unsigned int Index, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of EDIT widget.
Index	Color index. See table below.
Color	Color to be set.

Table 19.167: EDIT_SetBkColor() parameter list

Permitted values for parameter Index	
EDIT_CI_DISABLED	Color index for the disabled state.
EDIT_CI_ENABLED	Color index for the enabled state.

EDIT_SetCursorAtChar()

Description

Sets the EDIT widget cursor to a specified character position.

Prototype

```
void EDIT_SetCursorAtChar(EDIT_Handle hObj, int xPos);
```

Parameter	Description
hObj	Handle of EDIT widget.
xPos	Character position to set cursor to.

Table 19.168: EDIT_SetCursorAtChar() parameter list

Additional information

The character position works as follows:
0: left of the first (leftmost) character,
1: between the first and second characters,
2: between the second and third characters,
and so on.

EDIT_SetCursorAtPixel()

Description

Sets the EDIT widget cursor to a specified pixel position.

Prototype

```
void EDIT_SetCursorAtPixel(EDIT_Handle hObj, int Pos);
```

Parameter	Description
hObj	Handle of EDIT widget.
Pos	Pixel position to set cursor to.

Table 19.169: EDIT_SetCursorAtPixel() parameter list

EDIT_SetDecMode()

Description

Enables the decimal edit mode of the EDIT widget. The given value can be modified in the given range.

Prototype

```
void EDIT_SetDecMode(EDIT_Handle hEdit, I32 Value, I32 Min,
                     I32         Max,   int Shift, U8  Flags);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	Value to be set.
Min	Minimum value.
Max	Maximum value.
Shift	If > 0 it specifies the position of the decimal point.
Flags	See table below.

Table 19.170: EDIT_SetDecMode() parameter list

Permitted values for parameter Flags ("OR" combinable)	
GUI_EDIT_NORMAL	Edit in normal mode. A sign is displayed only if the value is negative.
GUI_EDIT_SIGNED	"+" and "-" sign is displayed.

EDIT_SetDefaultBkColor()

Description

Sets the default background color used for EDIT widgets.

Prototype

```
void EDIT_SetDefaultBkColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Color index. See table below.
Color	Color to be used.

Table 19.171: EDIT_SetDefaultBkColor() parameter list

Permitted values for parameter Index	
EDIT_CI_DISABLED	Color index for the disabled state.
EDIT_CI_ENABLED	Color index for the enabled state.

EDIT_SetDefaultFont()

Description

Sets the default font used for EDIT widgets.

Prototype

```
void EDIT_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font to be set as default.

Table 19.172: EDIT_SetDefaultFont() parameter list

EDIT_SetDefaultTextAlign()

Description

Sets the default text alignment for EDIT widgets.

Prototype

```
void EDIT_SetDefaultTextAlign(int Align);
```

Parameter	Description
Align	Default text alignment. For details, refer to "EDIT_SetTextAlign()" on page 527.

Table 19.173: EDIT_SetDefaultTextAlign() parameter list

EDIT_SetDefaultTextColor()

Description

Sets the default text color used for EDIT widgets.

Prototype

```
void EDIT_SetDefaultTextColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Has to be 0, reserved for future use.
Color	Color to be used.

Table 19.174: EDIT_SetDefaultTextColor() parameter list

EDIT_SetFloatMode()

Description

Enables the floating point edit mode of the EDIT widget. The given value can be modified in the given range.

Prototype

```
void  EDIT_SetFloatMode(EDIT_Handle hObj, float Value, float Min,
                        float       Max,  int   Shift, U8    Flags);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Shift	Number of post decimal positions.
Flags	See table below.

Table 19.175: EDIT_SetFloatMode() parameter list

Permitted values for parameter Flags ("OR" combinable)	
GUI_EDIT_NORMAL	Edit in normal mode. A sign is displayed only if the value is negative.
GUI_EDIT_SIGNED	"+" and "-" sign is displayed.
GUI_EDIT_SUPPRESS_LEADING_ZEROES	Does not show leading zeroes.

EDIT_SetFloatValue()

Description

The function can be used to set the floating point value of the EDIT widget if working in floating point mode.

Prototype

```
void EDIT_SetFloatValue(EDIT_Handle hObj, float Value);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	New floating point value of the EDIT widget.

Table 19.176: EDIT_SetFloatValue() parameter list

Additional information

The use of this function makes only sense if the EDIT widget works in floating point mode. If working in text mode the function has no effect. If working in binary, decimal or hexadecimal mode the behavior of the EDIT widget is undefined.

EDIT_SetFocussable()

Description

Sets the focussability of the EDIT widget.

Prototype

```
void EDIT_SetFocussable(EDIT_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of EDIT widget.
State	If State is set to 0, the EDIT widget is set not to be focusable. Otherwise it is set to be focusable.

Table 19.177: EDIT_SetFocussable() parameter list

EDIT_SetFont()

Description

Sets the used font of the EDIT widget.

Prototype

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of EDIT widget.
pFont	Pointer to the font.

Table 19.178: EDIT_SetFont() parameter list

EDIT_SetHexMode()

Description

Enables the hexadecimal edit mode of the EDIT widget. The given value can be modified in the given range.

Prototype

```
void EDIT_SetHexMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.

Table 19.179: EDIT_SetHexMode() parameter list

EDIT_SetInsertMode()

Description

Enables or disables the insert mode of the EDIT widget.

Prototype

```
int EDIT_SetInsertMode(EDIT_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle of EDIT widget.
OnOff	See table below.

Table 19.180: EDIT_SetInsertMode() parameter list

Permitted values for parameter OnOff	
0	Disable insert mode.
1	Enable insert mode.

Return value

Returns the previous insert mode state.

Additional information

The use of this function makes only sense if the EDIT widget operates in text mode or in any user defined mode. If working in hexadecimal, binary, floating point or decimal mode the use of this function has no effect except that it changes the appearance of the cursor.

EDIT_SetMaxLen()

Description

Sets the maximum number of characters to be edited by the given EDIT widget.

Prototype

```
void EDIT_SetMaxLen(EDIT_Handle hObj, int MaxLen);
```

Parameter	Description
hObj	Handle of EDIT widget.
MaxLen	Number of characters.

Table 19.181: EDIT_SetMaxLen() parameter list

EDIT_SetpfAddKeyEx()

Description

Sets the function pointer which is used by the EDIT widget to call the function which is responsible for adding characters.

Prototype

```
void EDIT_SetpfAddKeyEx(EDIT_Handle hObj, tEDIT_AddKeyEx * pfAddKeyEx);
```

Parameter	Description
hObj	Handle of EDIT widget.
pfAddKeyEx	Function pointer to the function to be used to add a character.

Table 19.182: EDIT_SetpfAddKeyEx() parameter list

Additional information

If working in text mode (default) or one of the modes for editing values, the EDIT widget uses its own routines to add a character. The use of this function only makes sense if the default behavior of the EDIT widget needs to be changed. If a function pointer has been set with this function the application program is responsible for the content of the text buffer.

EDIT_SetSel()

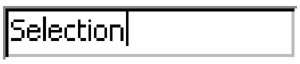
Before	After
	

Table 19.183: EDIT_SetSel() before after screenshots

Description

Used to set the current selection of the EDIT widget.

Prototype

```
void EDIT_SetSel(EDIT_Handle hObj, int FirstChar, int LastChar);
```

Parameter	Description
hObj	Handle of EDIT widget.
FirstChar	Zero based index of the first character to be selected. -1 if no character should be selected.
LastChar	Zero based index of the last character to be selected. -1 if all characters from the first character until the last character should be selected.

Table 19.184: EDIT_SetSel() parameter list

Additional information

Selected characters are usually displayed in reverse. Setting the cursor position deselects all characters.

Example

```
EDIT_SetSel(0, -1);  // Selects all characters of the widget
EDIT_SetSel(-1, 0);  // Deselects all characters
EDIT_SetSel(0, 2);   // Selects the first 3 characters
```

EDIT_SetText()

Description

Sets the text to be displayed in the EDIT widget.

Prototype

```
void EDIT_SetText(EDIT_Handle hObj, const char * s)
```

Parameter	Description
hObj	Handle of EDIT widget.
s	Text to display.

Table 19.185: EDIT_SetText() parameter list

EDIT_SetTextAlign()

Description

Sets the text alignment of the EDIT widget.

Prototype

```
void EDIT_SetTextAlign(EDIT_Handle hObj, int Align);
```

Parameter	Description
hObj	Handle of the EDIT widget.
Align	Or-combination of text alignment flags. See table below.

Table 19.186: EDIT_SetTextAlign() parameter list

Permitted values for parameter Align (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right.
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

EDIT_SetTextColor()

Description

Sets the text color of the EDIT widget.

Prototype

```
void EDIT_SetTextColor(EDIT_Handle hObj,   unsigned int Index,
                                   GUI_COLOR    Color);
```

Parameter	Description
hObj	Handle of EDIT widget.
Index	Index for text color. See table below.
Color	Color to be set.

Table 19.187: EDIT_SetTextColor() parameter list

Permitted values for parameter OnOff	
EDIT_CI_DISABLED	Sets the text color for disabled state.
EDIT_CI_ENABLED	Sets the text color for enabled state.

EDIT_SetTextMode()

Description

Sets the edit mode of the EDIT widget back to text mode.

Prototype

```
void EDIT_SetTextMode(EDIT_Handle hEdit);
```

Parameter	Description
hObj	Handle of EDIT widget.

Table 19.188: EDIT_SetTextMode() parameter list

Additional information

If one of the functions EDIT_SetBinMode(), EDIT_SetDecMode(), EDIT_SetFloatMode() or EDIT_SetHexMode() has been used to set the EDIT widget to one of the numeric edit modes, this function sets the edit mode back to text mode. It also clears the content of the EDIT widget.

EDIT_SetUlongMode()

Description

Enables the unsigned long decimal edit mode of the EDIT widget. The given value can be modified in the given range.

Prototype

```
void EDIT_SetUlongMode(EDIT_Handle hEdit, U32 Value, U32 Min, U32 Max);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.

Table 19.189: EDIT_SetUlongMode() parameter list

EDIT_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

EDIT_SetValue()

Description

Sets the current value of the EDIT widget. Only useful if binary, decimal or hexadecimal edit mode is set.

Prototype

```
void EDIT_SetValue(EDIT_Handle hObj, I32 Value);
```

Parameter	Description
hObj	Handle of EDIT widget.
Value	New value.

Table 19.190: EDIT_SetValue() parameter list

GUI_EditBin()

Description

Edits a binary value at the current cursor position.

Prototype

```
U32 GUI_EditBin(U32 Value, U32 Min, U32 Max, int Len, int xSize);
```

Parameter	Description
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Len	Number of digits to edit.
xSize	Pixel-size in X of the edit field.

Table 19.191: GUI_EditBin() parameter list

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Additional information

The routine returns after pressing <ENTER> or <ESC>. The content of the given text will be modified only if <ENTER> is pressed.

GUI_EditDec()

Description

Edits a decimal value at the current cursor position.

Prototype

```
U32 GUI_EditDec(I32 Value, I32 Min, I32 Max, int Len, int xSize,
                int Shift, U8 Flags);
```

Parameter	Description
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Len	Number of digits to edit.
xSize	Pixel-size in X of edit field.
Shift	If > 0 it specifies the position of the decimal point.
Flags	See `EDIT_SetDecMode()`.

Table 19.192: GUI_EditDec() parameter list

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Additional information

The routine returns after pressing <ENTER> or <ESC>. The content of the given text will be modified only if <ENTER> is pressed.

GUI_EditFloat()

Description

Edits a floating point value at the current cursor position.

Prototype

```
float GUI_EditFloat(float Value, float Min,   float Max,    int Len,
```



```
                      int   xSize, int   Shift, U8   Flags);
```

Parameter	Description
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Len	Number of digits to edit.
xSize	Pixel-size in X of the EDIT widget.
Shift	Specifies the position of the decimal point, if > 0.
Flags	See EDIT_SetFloatMode().

Table 19.193: GUI_EditFloat() parameter list

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Additional information

The routine returns after pressing <ENTER> or <ESC>. The content of the given text will be modified only if <ENTER> is pressed.

GUI_EditHex()

Description

Edits a hexadecimal value at the current cursor position.

Prototype

```
U32 GUI_EditHex(U32 Value, U32 Min, U32 Max, int Len, int xSize);
```

Parameter	Description
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Len	Number of digits to edit.
xSize	Pixel-size in X of the edit field.

Table 19.194: GUI_EditHex() parameter list

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Additional information

The routine returns after pressing <ENTER> or <ESC>. The content of the given text will be modified only if <ENTER> is pressed.

GUI_EditString()

Description

Edits a string at the current cursor position.

Prototype

```
void GUI_EditString(char * pString, int Len, int xSize);
```

Parameter	Description
pString	Pointer to the string to be edited.
Len	Maximum number of characters.
xSize	Pixel-size in X of the edit field.

Table 19.195: GUI_EditString() parameter list

Additional information

The routine returns after pressing <ENTER> or <ESC>. The content of the given text will be modified only if <ENTER> is pressed.

19.8.6 Examples

The `Sample` folder contains the following examples which show how the widget can be used:

- `WIDGET_Edit.c`
- `WIDGET_EditWinmode.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Edit.c:

Screenshot of WIDGET_EditWinmode.c:

19.9 FRAMEWIN: Frame window widget

FRAMEWIN widgets give your application a PC application-window appearance. They consist of a surrounding frame, a title bar and a user area. The color of the title bar changes to show whether the window is active or inactive, as seen below:

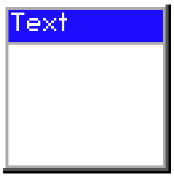
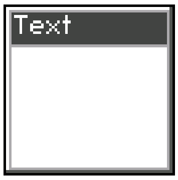
Active frame window	Inactive frame window
	

Table 19.196: Active and inactive frame window example

You can attach predefined buttons to the title bar as seen below or you can attach your own buttons to a title bar:

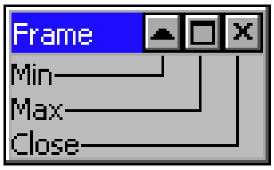
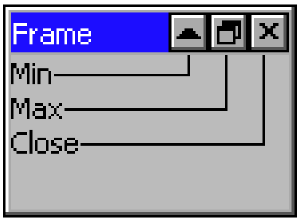
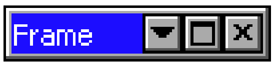
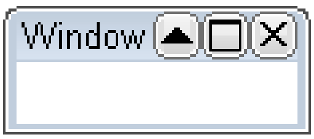
Description	Frame window
Frame window with minimize-, maximize- and close button.	
Frame window with minimize-, maximize- and close button in maximized state.	
Frame window with minimize-, maximize- and close button in minimized state	

Table 19.197: Frame window button description

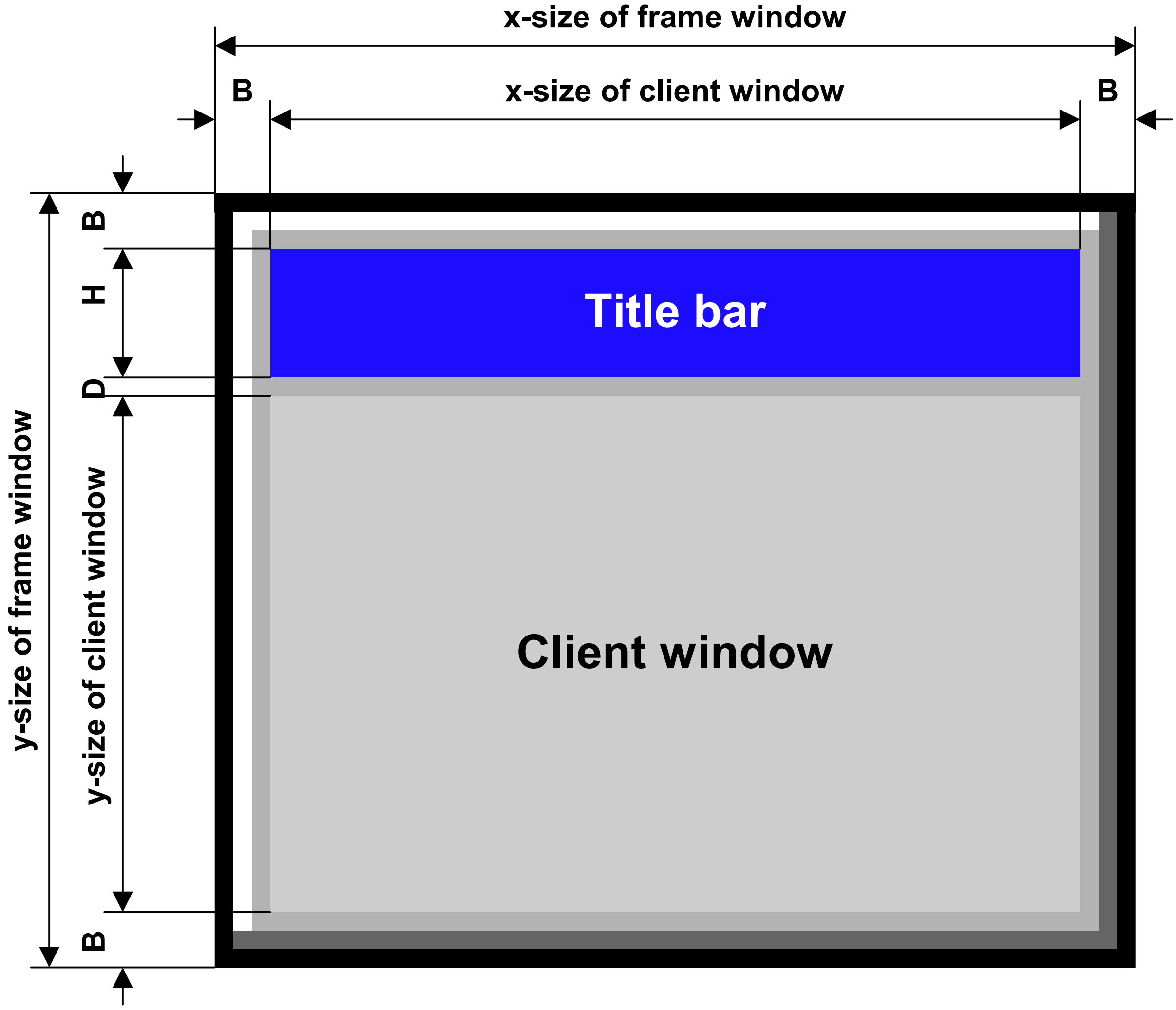
Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter "Skinning" on page 871.

19.9.1 Structure of the frame window

The following diagram shows the detailed structure and looks of a frame window:



The frame window actually consists of 2 windows; the main window and a child window. The child window is called `Client window`. It is important to be aware of this when dealing with callback functions: There are 2 windows with 2 different callback functions. When creating child windows, these child windows are typically created as children of the client window; their parent is therefor the client window.

Detail	Description
`B`	Border size of the frame window. The default size of the border is 3 pixels.
`H`	Height of the title bar. Depends on the size of the used font for the title.
`D`	Spacing between title bar and client window. (1 pixel)
`Title bar`	The title bar is part of the frame window and not a separate window.
`Client window`	The client window is a separate window created as a child window of the frame window.

Table 19.198: Frame window description

19.9.2 Configuration options

Type	Macro	Default	Description
B	FRAMEWIN_ALLOW_DRAG_ON_FRAME	1	Allows dragging the widget on the surrounding frame.
N	FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT	0xff0000	Title bar background color, active state.
N	FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT	0x404040	Title bar background color, inactive state.
N	FRAMEWIN_BORDER_DEFAULT	3	Outer border width, in pixels.
N	FRAMEWIN_CLIENTCOLOR_DEFAULT (with WIDGET_USE_FLEX_SKIN == 0) (with WIDGET_USE_FLEX_SKIN == 1)	0xc0c0c0 GUI_WHITE	Color of client window area.
S	FRAMEWIN_DEFAULT_FONT (with WIDGET_USE_FLEX_SKIN == 0) (with WIDGET_USE_FLEX_SKIN == 1)	&GUI_Font8_1 &GUI_Font13_1	Font used for title bar text.
N	FRAMEWIN_FRAMECOLOR_DEFAULT	0xaaaaaa	Frame color.
N	FRAMEWIN_IBORDER_DEFAULT	1	Inner border width, in pixels.
N	FRAMEWIN_TEXTCOLOR_ACTIVE_DEFAULT (with WIDGET_USE_FLEX_SKIN == 0) (with WIDGET_USE_FLEX_SKIN == 1)	GUI_WHITE GUI_BLACK	Title bar text color, active state.
N	FRAMEWIN_TEXTCOLOR_INACTIVE_DEFAULT (with WIDGET_USE_FLEX_SKIN == 0) (with WIDGET_USE_FLEX_SKIN == 1)	GUI_WHITE GUI_BLACK	Title bar text color, inactive state.
N	FRAMEWIN_TITLEHEIGHT_DEFAULT	0	Default height of title bar.

Table 19.199: Configuration options

19.9.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

19.9.4 FRAMEWIN API

The table below lists the available emWin FRAMEWIN-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
FRAMEWIN_AddButton()	Adds a button in the title bar.
FRAMEWIN_AddCloseButton()	Adds a close button in the title bar.
FRAMEWIN_AddMaxButton()	Adds a maximize button in the title bar.
FRAMEWIN_AddMenu()	Adds a menu widget to the frame window.
FRAMEWIN_AddMinButton()	Adds a minimize button in the title bar.
FRAMEWIN_Create()	Creates a FRAMEWIN widget. (Obsolete)
FRAMEWIN_CreateAsChild()	Creates a FRAMEWIN widget as a child window. (Obsolete)
FRAMEWIN_CreateEx()	Creates a FRAMEWIN widget.
FRAMEWIN_CreateIndirect()	Creates a FRAMEWIN widget from a resource table entry.
FRAMEWIN_CreateUser()	Creates a FRAMEWIN widget using extra bytes as user data.
FRAMEWIN_GetActive()	Returns if the frame window is in active state.
FRAMEWIN_GetBarColor()	Returns the color of the title bar.
FRAMEWIN_GetBorderSize()	Returns the size of the border.
FRAMEWIN_GetDefaultBarColor()	Returns the default color of the title bar.
FRAMEWIN_GetDefaultBorderSize()	Returns the default border size

Table 19.200: FRAMEWIN API list

Routine	Description
FRAMEWIN_GetDefaultClientColor()	Returns the default color of the client area.
FRAMEWIN_GetDefaultFont()	Returns the default font used for the title bar
FRAMEWIN_GetDefaultTextColor()	Returns the default text color of the title.
FRAMEWIN_GetDefaultTitleHeight()	Returns the default size of the title bar
FRAMEWIN_GetFont()	Returns the font used for the title text.
FRAMEWIN_GetText()	Returns the title text.
FRAMEWIN_GetTextAlign()	Returns the alignment of the title text.
FRAMEWIN_GetTitleHeight()	Returns the height of the title bar.
FRAMEWIN_GetUserData()	Retrieves the data set with FRAMEWIN_SetUserData().
FRAMEWIN_IsMinimized()	Returns if the FRAMEWIN widget is minimized or not.
FRAMEWIN_IsMaximized()	Returns if the FRAMEWIN widget is maximized or not.
FRAMEWIN_Maximize()	Enlarges the FRAMEWIN widget to the size of its parent.
FRAMEWIN_Minimize()	Hides the client area of the FRAMEWIN widget.
FRAMEWIN_OwnerDraw()	Default function for drawing the title bar.
FRAMEWIN_Restore()	Restores a minimized or maximized FRAMEWIN widget.
FRAMEWIN_SetActive()	Sets the state of the FRAMEWIN widget. (Obsolete)
FRAMEWIN_SetBarColor()	Sets the color of the title bar.
FRAMEWIN_SetBorderSize()	Sets the border size of the FRAMEWIN widget.
FRAMEWIN_SetClientColor()	Sets the color of the client area.
FRAMEWIN_SetDefaultBarColor()	Sets the default color of the title bar.
FRAMEWIN_SetDefaultBorderSize()	Sets the default border size.
FRAMEWIN_SetDefaultClientColor()	Sets the default color of the client area.
FRAMEWIN_SetDefaultFont()	Sets the default font used for the title bar.
FRAMEWIN_SetDefaultTextColor()	Sets the default text color of the title.
FRAMEWIN_SetDefaultTitleHeight()	Sets the default height of the title bar
FRAMEWIN_SetFont()	Selects the font used for the title text.
FRAMEWIN_SetMoveable()	Sets the FRAMEWIN widget to a moveable/non-moveable state.
FRAMEWIN_SetOwnerDraw()	Enables the FRAMEWIN widget to be owner drawn.
FRAMEWIN_SetResizeable()	Sets the FRAMEWIN widget to resizable state.
FRAMEWIN_SetText()	Sets the title text.
FRAMEWIN_SetTextAlign()	Sets the alignment of the title text.
FRAMEWIN_SetTextColor()	Sets the color(s) for the title text.
FRAMEWIN_SetTextColorEx()	Sets the color(s) for the title text.
FRAMEWIN_SetTitleHeight()	Sets the height of the title bar.
FRAMEWIN_SetTitleVis()	Sets the visibility flag of the title bar
FRAMEWIN_SetUserData()	Sets the extra data of a FRAMEWIN widget.

Table 19.200: FRAMEWIN API list

FRAMEWIN_AddButton()

Before	After
	

Table 19.201: FRAMEWIN_AddButton() before after screenshots

Description

Adds a button to the title bar of the FRAMEWIN widget.

Prototype

```
WM_HWIN FRAMEWIN_AddButton(FRAMEWIN_Handle hObj, int Flags,
                           int             Off,  int Id);
```

Parameter	Description
`hObj`	Handle of FRAMEWIN widget.
`Flags`	See table below.
`Off`	X-offset used to create the BUTTON widget
`Id`	ID of the BUTTON widget

Table 19.202: FRAMEWIN_AddButton() parameter list

Permitted values for parameter `Flags`	
`FRAMEWIN_BUTTON_LEFT`	The BUTTON will be created at the left side.
`FRAMEWIN_BUTTON_RIGHT`	The BUTTON will be created at the right side.

Return value

Handle of the BUTTON widget.

Additional information

The button will be created as a child window from the FRAMEWIN widget. So the Window Manager keeps sure it will be deleted when the FRAMEWIN widget will be deleted.
The button can be created at the left side or at the right side of the title bar depending on the parameter Flags. The parameter Offset specifies the space between the button and the border of the FRAMEWIN widget or the space between the previous created button.

FRAMEWIN_AddCloseButton()

Before	After
	

Table 19.203: FRAMEWIN_AddCloseButton() before after screenshots

Description

Adds a close button to the title bar of the FRAMEWIN widget.

Prototype

```
WM_HWIN FRAMEWIN_AddCloseButton(FRAMEWIN_Handle hObj, int Flags, int Off);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Flags	See table below.
Off	X-offset used to create the BUTTON widget

Table 19.204: FRAMEWIN_AddCloseButton() parameter list

Permitted values for parameter Index	
FRAMEWIN_BUTTON_LEFT	The BUTTON will be created at the left side.
FRAMEWIN_BUTTON_RIGHT	The BUTTON will be created at the right side.

Return value

Handle of the close button.

Additional information

When the user presses the close button the frame window and all its children will be deleted.

FRAMEWIN_AddMaxButton()

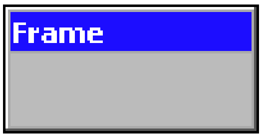
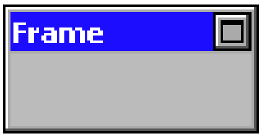
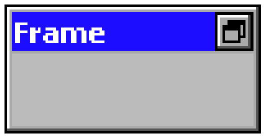
Before	After	Maximized
		

Table 19.205: FRAMEWIN_AddMaxButton() before after screenshots

Description

Adds a maximize button to the title bar of the FRAMEWIN widget.

Prototype

```
WM_HWIN FRAMEWIN_AddMaxButton(FRAMEWIN_Handle hObj, int Flags, int Off);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Flags	See table below.
Off	X-offset used to create the BUTTON widget

Table 19.206: FRAMEWIN_AddMaxButton() parameter list

Permitted values for parameter Index	
FRAMEWIN_BUTTON_LEFT	The BUTTON will be created at the left side.
FRAMEWIN_BUTTON_RIGHT	The BUTTON will be created at the right side.

Return value

Handle of the maximize button.

Additional information

When the user presses the maximize button the first time the FRAMEWIN widget will be enlarged to the size of its parent window. The second use of the button will reduce the frame window to its old size and restores the old position.

FRAMEWIN_AddMenu()

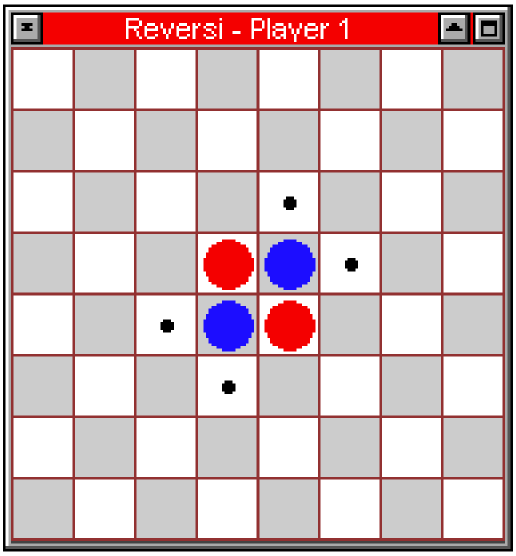
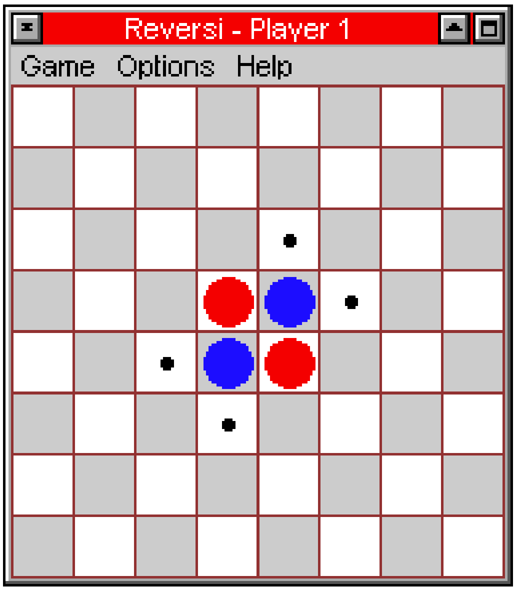
<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 19.207: FRAMEWIN_AddMenu() before after screenshots

Description

Adds the given menu to a FRAMEWIN widget. The menu is shown below the title bar.

Prototype

```
void FRAMEWIN_AddMenu(FRAMEWIN_Handle hObj, WM_HWIN hMenu);
```

Parameter	Description
hObj	Handle of frame window.
hMenu	Handle of MENU widget to be added.

Table 19.208: FRAMEWIN_AddMenu() parameter list

Additional information

The added MENU is attached as a child of the FRAMEWIN widget. If the FRAMEWIN widget has been created with a callback routine, the function makes sure, that the `WM_MENU` messages are passed to the client window of the FRAMEWIN widget.

FRAMEWIN_AddMinButton()

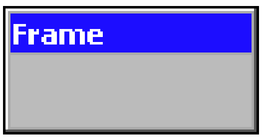
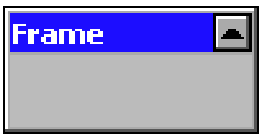
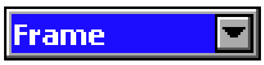
<table>
<tr><th>Before</th><th>After</th><th>Minimized window</th></tr>
<tr><td>
</td><td>
</td><td>
</td></tr>
</table>

Table 19.209: FRAMEWIN_AddMinButton() before after screenshots

Description

Adds a minimize button to the title bar of the FRAMEWIN widget.

Prototype

```
WM_HWIN FRAMEWIN_AddMinButton(FRAMEWIN_Handle hObj, int Flags, int Off);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Flags	See table below.
Off	X-offset used to create the BUTTON widget

Table 19.210: FRAMEWIN_AddMinButton() parameter list

Permitted values for parameter Index	
FRAMEWIN_BUTTON_LEFT	The BUTTON will be created at the left side.
FRAMEWIN_BUTTON_RIGHT	The BUTTON will be created at the right side.

Return value

Handle of the minimize button.

Additional information

When the user presses the minimize button the first time the client area of the FRAMEWIN widget will be hidden and only the title bar remains visible. The second use of the button will restore the FRAMEWIN widget to its old size.

FRAMEWIN_Create()

(Obsolete, `FRAMEWIN_CreateEx()` should be used instead)

Description

Creates a FRAMEWIN widget of a specified size at a specified location.

Prototype

```
FRAMEWIN_Handle FRAMEWIN_Create(const char * pTitle, WM_CALLBACK * cb,
                                int          Flags,
                                int          x0,     int           y0,
                                int          xSize,  int           ySize);
```

Parameter	Description
pTitle	Title displayed in the title bar.
cb	Pointer to callback routine of client area.
Flags	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
x0	Leftmost pixel of the frame window (in parent coordinates).
y0	Topmost pixel of the frame window (in parent coordinates).
xSize	Horizontal size of the frame window (in pixels).
ySize	Vertical size of the frame window (in pixels).

Table 19.211: FRAMEWIN_Create() parameter list

Return value

Handle of the created FRAMEWIN widget; 0 if the function fails.

FRAMEWIN_CreateAsChild()

(Obsolete, `FRAMEWIN_CreateEx` should be used instead)

Description

Creates a FRAMEWIN widget as a child window.

Prototype

```
FRAMEWIN_Handle FRAMEWIN_CreateAsChild(int            x0,      int y0,
                                       int            xSize,   int ySize,
                                       WM_HWIN        hParent,
                                       const char   * pText,
                                       WM_CALLBACK  * cb,      int Flags);
```

Parameter	Description
x0	X-position of the FRAMEWIN widget relative to the parent window.
y0	Y-position of the FRAMEWIN widget relative to the parent window.
xSize	Horizontal size of the FRAMEWIN widget (in pixels).
ySize	Vertical size of the FRAMEWIN widget (in pixels).
hParent	Handle of parent window.
pText	Text to be displayed in the title bar.
cb	Optional pointer to a custom callback function for the client window.
Flags	Window create flags (see FRAMEWIN_Create()).

Table 19.212: FRAMEWIN_CreateAsChild() parameter list

Return value

Handle of the created FRAMEWIN widget; 0 if the function fails.

FRAMEWIN_CreateEx()

Description

Creates a FRAMEWIN widget of a specified size at a specified location.

Prototype

```
FRAMEWIN_Handle FRAMEWIN_CreateEx(int            x0,      int y0,
                                  int            xSize,   int ySize,
                                  WM_HWIN        hParent, int WinFlags,
                                  int            ExFlags, int Id,
                                  const char   * pTitle,
                                  WM_CALLBACK  * cb);
```

Parameter	Description
x0	Leftmost pixel of the FRAMEWIN widget (in parent coordinates).
y0	Topmost pixel of the FRAMEWIN widget (in parent coordinates).
xSize	Horizontal size of the FRAMEWIN widget (in pixels).
ySize	Vertical size of the FRAMEWIN widget (in pixels).
hParent	Handle of parent window. If 0, the new FRAMEWIN widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter “The Window Manager (WM)” on page 375 for a list of available parameter values).
ExFlags	See table below.
Id	Window ID of the FRAMEWIN widget.
pTitle	Title displayed in the title bar.
cb	Optional pointer to a custom callback function for the client window.

Table 19.213: FRAMEWIN_CreateEx() parameter list

Permitted values for parameter ExFlags	
0	No function.
FRAMEWIN_CF_MOVEABLE	Sets the new frame window to a moveable state. For details, refer to “FRAMEWIN_SetMoveable()” on page 552.

Return value

Handle of the created FRAMEWIN widget; 0 if the function fails.

Additional information

The user callback routine is typically used for 2 purposes:
- to paint the client window (if filling with a color is not desired)
- to react to messages of child windows, typically dialog elements

The normal behaviour of the client window is to paint itself, filling the entire window with the client color.
If the user callback also fills the client window (or a part of it), it can be desirable to set the client color to `GUI_INVALID_COLOR`, causing the window callback to not fill the client window.
The user callback of the client window does not receive all messages sent to the client window; some system messages are completely handled by the window callback routine and are not passed to the user callback. All notification messages as well as WM_PAINT and all user messages are sent to the user callback routine.
The handle received by the user callback is the handle of the frame window (the parent window of the client window), except for the WM_PAINT message, which receives the handle of the client window.

FRAMEWIN_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `FRAMEWIN_CreateEx()`.

FRAMEWIN_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `FRAMEWIN_CreateEx()` can be referred to.

FRAMEWIN_GetActive()

Description

Returns if the given FRAMEWIN widget is in active state or not.

Prototype

```
GUI_COLOR FRAMEWIN_GetBarColor(FRAMEWIN_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.214: FRAMEWIN_GetActive() parameter list

Return value

1 if FRAMEWIN widget is in active state, 0 if not.

FRAMEWIN_GetBarColor()

Description

Returns the color of the title bar of the given FRAMEWIN widget.

Prototype

```
GUI_COLOR FRAMEWIN_GetBarColor(FRAMEWIN_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Index	See table below.

Table 19.215: FRAMEWIN_GetBarColor() parameter list

Permitted values for parameter Index	
0	Returns the bar color used when frame window is inactive.
1	Returns the bar color used when frame window is active.

Return value

Color of the title bar as RGB value.

FRAMEWIN_GetBorderSize()

Description

Returns the border size of the given FRAMEWIN widget.

Prototype

```
int FRAMEWIN_GetBorderSize(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.216: FRAMEWIN_GetBorderSize() parameter list

Return value

The border size of the given FRAMEWIN widget.

FRAMEWIN_GetDefaultBarColor()

Description

Returns the default color for title bars in FRAMEWIN widgets.

Prototype

```
const GUI_COLOR * FRAMEWIN_GetDefaultBarColor(unsigned int Index);
```

Parameter	Description
Index	See table below.

Table 19.217: FRAMEWIN_GetDefaultBarColor() parameter list

Permitted values for parameter Index	
0	Returns the bar color used when frame window is inactive.
1	Returns the bar color used when frame window is active.

Return value

Pointer to the default title bar color used for FRAMEWIN widgets in the specified state.

FRAMEWIN_GetDefaultBorderSize()

Description

Returns the default border size of FRAMEWIN widgets.

Prototype

```
int FRAMEWIN_GetDefaultBorderSize(void);
```

Return value

Default border size of FRAMEWIN widgets.

FRAMEWIN_GetDefaultClientColor()

Description

Returns the default color of client areas in FRAMEWIN widgets.

Prototype

```
const GUI_COLOR * FRAMEWIN_GetDefaultClientColor(void);
```

Return value

Pointer to the default client area color.

FRAMEWIN_GetDefaultFont()

Description

Returns the default font used for captions of FRAMEWIN widgets.

Prototype

```
const GUI_FONT* FRAMEWIN_GetDefaultFont(void);
```

Return value

Pointer to the default font used for captions of captions widgets.

FRAMEWIN_GetDefaultTextColor()

Description

Returns the default text color of the title.

Prototype

```
GUI_COLOR FRAMEWIN_GetDefaultTextColor(unsigned Index);
```

Parameter	Description
Index	See table below.

Table 19.218: FRAMEWIN_GetDefaultTextColor() parameter list

Permitted values for parameter Index	
0	Color to be used when frame window is inactive.
1	Color to be used when frame window is active.

Return value

Default text color of the title.

FRAMEWIN_GetFont()

Description

Returns a pointer to the font used to draw the title text.

Prototype

```
const GUI_FONT * FRAMEWIN_GetFont(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.219: FRAMEWIN_GetFont() parameter list

Return value

Pointer to the font used to draw the title text.

FRAMEWIN_GetText()

Description

Returns the title text.

Prototype

```
void FRAMEWIN_GetText(FRAMEWIN_Handle hObj, char * pBuffer, int MaxLen);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
pBuffer	Pointer to buffer to be filled with the title text.
MaxLen	Buffer size in bytes.

Table 19.220: FRAMEWIN_GetText() parameter list

Additional information

If the buffer size is smaller than the title text the function copies `MaxLen`.

FRAMEWIN_GetTextAlign()

Description

Returns the text alignment of the title text.

Prototype

```
int FRAMEWIN_GetTextAlign(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.221: FRAMEWIN_GetTextAlign() parameter list

Return value

The currently used text alignment. For details about text alignment, refer to"GUI_SetTextAlign()" on page 105.

FRAMEWIN_GetDefaultTitleHeight()

Description

Returns the default height of title bars in FRAMEWIN widget.

Prototype

```
int FRAMEWIN_GetDefaultCaptionSize(void);
```

Return value

Default title bar height. For more information about the title height, refer to "FRAMEWIN_SetDefaultTitleHeight()" on page 551.

FRAMEWIN_GetTitleHeight()

Description

Returns the height of title bar of the given FRAMEWIN widget.

Prototype

```
int FRAMEWIN_GetTitleHeight(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.222: FRAMEWIN_GetTitleHeight() parameter list

Return value

The height of title bar of the given FRAMEWIN widget. For more information about the title height, refer to “FRAMEWIN_SetDefaultTitleHeight()” on page 551.

FRAMEWIN_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

FRAMEWIN_IsMaximized()

Description

Returns if the FRAMEWIN widget is maximized or not.

Prototype

```
int FRAMEWIN_IsMaximized(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.223: FRAMEWIN_IsMaximized() parameter list

Return value

1 if the FRAMEWIN widget is maximized, 0 if not.

FRAMEWIN_IsMinimized()

Description

Returns if the FRAMEWIN widget is minimized or not.

Prototype

```
int FRAMEWIN_IsMinimized(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.224: FRAMEWIN_IsMinimized() parameter list

Return value

1 if the FRAMEWIN widget is minimized, 0 if not.

FRAMEWIN_Maximize()

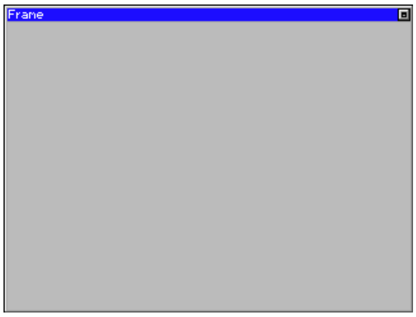
Before	After
	

Table 19.225: FRAMEWIN_Maximize() before after screenshots

Description

Enlarges a FRAMEWIN widget to the size of its parent window.

Prototype

```
void FRAMEWIN_Maximize(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.226: FRAMEWIN_Maximize() parameter list

Additional information

When calling this function the FRAMEWIN widget will show the same behavior as when the user presses the maximize button. The FRAMEWIN widget will be enlarged to the size of its parent window.

FRAMEWIN_Minimize()

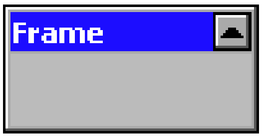
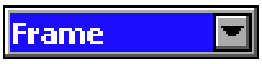
Before	After
	

Table 19.227: FRAMEWIN_Minimize() before after screenshots

Description

Hides the client area of the given FRAMEWIN widget.

Prototype

```
void FRAMEWIN_Minimize(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.228: FRAMEWIN_Minimize() parameter list

Additional information

When calling this function the FRAMEWIN widget will show the same behavior as when the user presses the minimize button. The client area of the frame window will be hidden and only the title bar remains visible.

FRAMEWIN_OwnerDraw()

Description

Default function for drawing the title bar of a FRAMEWIN widget.

Prototypes

```
int FRAMEWIN_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Description
pDrawItemInfo	Pointer to a WIDGET_ITEM_DRAW_INFO structure.

Table 19.229: FRAMEWIN_OwnerDraw() parameter list

Additional information

This function is useful if FRAMEWIN_SetOwnerDraw() is used. It should be called for all unhandled commands passed to the owner draw function. For more information, refer to the section explaining user drawn widgets and FRAMEWIN_SetOwnerDraw().

FRAMEWIN_Restore()

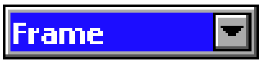
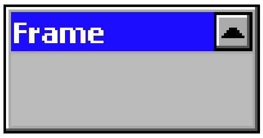
Before	After
	

Table 19.230: FRAMEWIN_Restore() before after screenshots

Description

Restores a minimized or maximized FRAMEWIN widget to its old size and position.

Prototype

```
void FRAMEWIN_Restore(FRAMEWIN_Handle hObj);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.

Table 19.231: FRAMEWIN_Restore() parameter list

Additional information

If the given FRAMEWIN widget is neither maximized nor minimized the function takes no effect.

FRAMEWIN_SetActive()

Before	After
	

Table 19.232: FRAMEWIN_SetActive() before after screenshots

Description

Sets the state of a specified FRAMEWIN widget. Depending on the state, the color of the title bar will change.

Prototype

```
void FRAMEWIN_SetActive(F0RAMEWIN_Handle hObj, int State);
```

Parameter	Description
`hObj`	Handle of FRAMEWIN widget.
`State`	State of FRAMEWIN widget. See table below.

Table 19.233: FRAMEWIN_SetActive() parameter list

Permitted values for parameter `State`	
0	FRAMEWIN widget is inactive.
1	FRAMEWIN widget is active.

Additional information

This function is obsolete. If pointing with a input device to a child of a FRAMEWIN widget the FRAMEWIN widget will become active automatically. It is not recommended to use this function. If using this function to set a FRAMEWIN widget to active state, it is not warranted that the state becomes inactive if an other FRAMEWIN widget becomes active.

FRAMEWIN_SetBarColor()

Before	After
	

Table 19.234: FRAMEWIN_SetBarColor() before after screenshots

Description

Sets the color of the title bar of a specified FRAMEWIN widget.

Prototype

```
void FRAMEWIN_SetBarColor(FRAMEWIN_Handle hObj,   unsigned int Index,
```



```
GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Index	Index for state of frame window. See table below.
Color	Color to be set.

Table 19.235: FRAMEWIN_SetBarColor() parameter list

Permitted values for parameter Index	
0	Sets the color to be used when FRAMEWIN widget is inactive.
1	Sets the color to be used when FRAMEWIN widget is active.

FRAMEWIN_SetBorderSize()

Before	After
	

Table 19.236: FRAMEWIN_SetBorderSize() before after screenshots

Description

Sets the border size of a specified FRAMEWIN widget.

Prototype

```
void FRAMEWIN_SetBorderSize(FRAMEWIN_Handle hObj, unsigned Size);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Size	New border size of the FRAMEWIN widget.

Table 19.237: FRAMEWIN_SetBorderSize() parameter list

FRAMEWIN_SetClientColor()

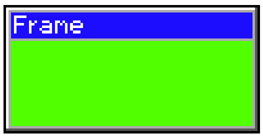
Before	After
	

Table 19.238: FRAMEWIN_SetClientColor() before after screenshots

Description

Sets the color of the client window area of a specified FRAMEWIN widget.

Prototype

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Color	Color to be set.

Table 19.239: FRAMEWIN_SetClientColor() parameter list

FRAMEWIN_SetDefaultBarColor()

Description

Sets the default color for title bars in FRAMEWIN widgets.

Prototype

```
void FRAMEWIN_SetDefaultBarColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Index for state of FRAMEWIN widgets. See table below.
Color	Color to be set.

Table 19.240: FRAMEWIN_SetDefaultBarColor()

Permitted values for parameter Index	
0	Sets the color to be used when FRAMEWIN widget is inactive.
1	Sets the color to be used when FRAMEWIN widget is active.

FRAMEWIN_SetDefaultBorderSize()

Description

Sets the default border size of FRAMEWIN widgets.

Prototype

```
void FRAMEWIN_SetDefaultBorderSize(int BorderSize);
```

Parameter	Description
BorderSize	Size to be set.

Table 19.241: FRAMEWIN_SetDefaultBorderSize() parameter list

FRAMEWIN_SetDefaultClientColor()

Description

Sets the default color for client areas in FRAMEWIN widgets.

Prototype

```
void FRAMEWIN_SetDefaultClientColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color to be set.

Table 19.242: FRAMEWIN_SetDefaultClientColor() parameter list

FRAMEWIN_SetDefaultFont()

Description

Sets the default font used to display the title in FRAMEWIN widgets.

Prototype

```
void FRAMEWIN_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to font to be used as default.

Table 19.243: FRAMEWIN_SetDefaultFont() parameter list

FRAMEWIN_SetDefaultTextColor()

Description

Sets the default text color of the title.

Prototype

```
void FRAMEWIN_SetDefaultTextColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Description
Index	See table below.
Color	Color to be used.

Table 19.244: FRAMEWIN_SetDefaultTextColor() parameter list

Permitted values for parameter Index	
0	Color to be used when frame window is inactive.
1	Color to be used when frame window is active.

FRAMEWIN_SetDefaultTitleHeight()

Description

Sets the size in Y for the title bar.

Prototype

```
void FRAMEWIN_SetDefaultTitleHeight(int Height);
```

Parameter	Description
Height	Size to be set

Table 19.245: FRAMEWIN_SetDefaultTitleHeight() parameter list

Additional information

The default value of the title height is 0. That means the height of the title depends on the font used to display the title text. If the default value is set to a value > 0 each new FRAMEWIN widget will use this height for the title height and not the height of the font of the title.

FRAMEWIN_SetFont()

Before	After
	

Table 19.246: FRAMEWIN_SetFont() before after screenshots

Description

Sets the title font of the FRAMEWIN widget.

Prototype

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
pFont	Pointer to the font.

Table 19.247: FRAMEWIN_SetFont() parameter list

FRAMEWIN_SetMoveable()

Description

Sets a FRAMEWIN widget to a moveable or fixed state.

Prototype

```
void FRAMEWIN_SetMoveable(FRAMEWIN_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
State	State of frame window. See table below.

Table 19.248: FRAMEWIN_SetMoveable() parameter list

Permitted values for parameter State	
0	Frame window is fixed (non-moveable).
1	Frame window is moveable.

Additional information

The default state of a FRAMEWIN widget after creation is fixed.
Moveable state means, the FRAMEWIN widget can be dragged with a pointer input device (PID). To move the FRAMEWIN widget, it first needs to be touched with a PID in pressed state in the title area. Moving the pressed PID now moves also the widget. If the config macro `FRAMEWIN_ALLOW_DRAG_ON_FRAME` is 1 (default), the FRAMEWIN widget can also be dragged on the surrounding frame. This works only if the FRAMEWIN widget is not in resizable state.

FRAMEWIN_SetOwnerDraw()

Description

Enables the FRAMEWIN widget to be owner drawn.

Prototype

```
void FRAMEWIN_SetOwnerDraw(FRAMEWIN_Handle          hObj,
                           WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
pfDrawItem	Pointer to owner draw function.

Table 19.249: FRAMEWIN_SetOwnerDraw() parameter list

Additional information

This function sets a function pointer to a function which will be called by the widget if a FRAMEWIN widget has to be drawn. It gives you the possibility to draw a complete customized title bar, not just plain text. `pfDrawItem` is a pointer to an application-defined function of type `WIDGET_DRAW_ITEM_FUNC` which is explained at the beginning of the chapter.

Example

The following shows a typical owner draw function:

```
int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  GUI_RECT Rect;
  char acBuffer[20];
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW:
    Rect.x0 = pDrawItemInfo->x0 + 1;
    Rect.x1 = pDrawItemInfo->x1 - 1;
    Rect.y0 = pDrawItemInfo->y0 + 1;
    Rect.y1 = pDrawItemInfo->y1;
    FRAMEWIN_GetText(pDrawItemInfo->hWin, acBuffer, sizeof(acBuffer));
    GUI_DrawGradientH(pDrawItemInfo->x0, pDrawItemInfo->y0,
                      pDrawItemInfo->x1, pDrawItemInfo->y1,
                      GUI_RED, GUI_GREEN);
    GUI_SetFont(FRAMEWIN_GetFont(pDrawItemInfo->hWin));
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_SetColor(GUI_YELLOW);
    GUI_DispStringInRect(acBuffer, &Rect,
                         FRAMEWIN_GetTextAlign(pDrawItemInfo->hWin));
    return 0;
  }
  return FRAMEWIN_OwnerDraw(pDrawItemInfo);
}

void CreateFrameWindow(void) {
  ...
  FRAMEWIN_SetOwnerDraw(hWin, _OwnerDraw);
  ...
}
```

Screenshot of above example

FRAMEWIN_SetResizeable()

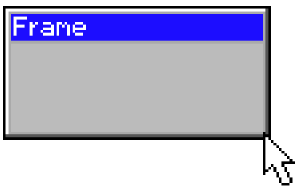
Before	After
	

Table 19.250: FRAMEWIN_SetResizeable() before after screenshots

Description

Sets the resizable state of the given FRAMEWIN widget.

Prototype

```
void FRAMEWIN_SetResizeable(FRAMEWIN_Handle hObj, int State);
```

Parameter	Description
`hObj`	Handle of FRAMEWIN widget.
`State`	1 if the frame window should be resizable, 0 if not.

Table 19.251: FRAMEWIN_SetResizeable() parameter list

Additional information

If the FRAMEWIN widget is in resizable state its size can be changed by dragging the borders. If a pointer input device points over the border, the cursor will change to a resize cursor (if cursor is on and if optional mouse support is enabled).

If pointing to the edge of the border, the X and Y size of the window can be changed simultaneously.

FRAMEWIN_SetText()

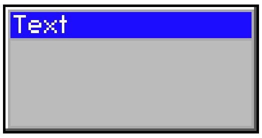
Before	After
	

Table 19.252: FRAMEWIN_SetText() before after screenshots

Description

Sets the title text.

Prototype

```
void FRAMEWIN_SetText(FRAMEWIN_Handle hObj, const char * s);
```

Parameter	Description
`hObj`	Handle of FRAMEWIN widget.
`s`	Text to display as the title.

Table 19.253: FRAMEWIN_SetText() parameter list

FRAMEWIN_SetTextAlign()

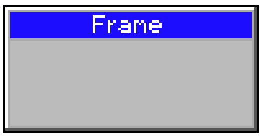
Before	After
	

Table 19.254: FRAMEWIN_SetTextAlign() before after screenshots

Description

Sets the text alignment of the title bar.

Prototype

```
void FRAMEWIN_SetTextAlign(FRAMEWIN_Handle hObj, int Align);
```

Parameter	Description
`hObj`	Handle of FRAMEWIN widget.
`Align`	Alignment attribute for the title. See table below.

Table 19.255: FRAMEWIN_SetTextAlign() parameter list

Permitted values for parameter `Align`	
`GUI_TA_HCENTER`	Centers the title (default).
`GUI_TA_LEFT`	Displays the title to the left.
`GUI_TA_RIGHT`	Displays the title to the right.

Additional information

If this function is not called, the default behavior is to display the text centered.

FRAMEWIN_SetTextColor()

Before	After
	

Table 19.256: FRAMEWIN_SetTextColor() before after screenshots

Description

Sets the color of the title text for both states, active and inactive.

Prototype

```
void FRAMEWIN_SetTextColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Color	Color to be set.

Table 19.257: FRAMEWIN_SetTextColor() parameter list

FRAMEWIN_SetTextColorEx()

Before	After
	

Table 19.258: FRAMEWIN_SetTextColorEx() before after screenshots

Description

Sets the text color for the given state.

Prototype

```
void FRAMEWIN_SetTextColorEx(FRAMEWIN_Handle hObj,   unsigned Index,
                             GUI_COLOR       Color);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Index	See table below.
Color	Color to be used.

Table 19.259: FRAMEWIN_SetTextColorEx() parameter list

Permitted values for parameter Index	
0	Color to be used when frame window is inactive.
1	Color to be used when frame window is active.

FRAMEWIN_SetTitleHeight()

Before	After
	

Table 19.260: FRAMEWIN_SetTitleHeight() before after screenshots

Description

Sets the height of the title bar.

Prototype

```
int FRAMEWIN_SetTitleHeight(FRAMEWIN_Handle hObj, int Height);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Height	Height of the title bar.

Table 19.261: FRAMEWIN_SetTitleHeight() parameter list

Additional information

Per default the height of the title bar depends on the size on the font used to display the title text. When using FRAMEWIN_SetTitleHeight the height will be fixed to the given value. Changes of the font takes no effect concerning the height of the title bar. A value of 0 will restore the default behavior.

FRAMEWIN_SetTitleVis()

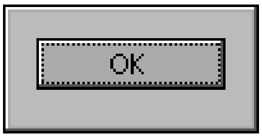
Before	After
	

Table 19.262: FRAMEWIN_SetTitleVis() before after screenshots

Description

Sets the visibility flag of the title bar.

Prototype

```
void FRAMEWIN_SetTitleVis(FRAMEWIN_Handle hObj, int Show);
```

Parameter	Description
hObj	Handle of FRAMEWIN widget.
Show	1 for visible (default), 0 for hidden

Table 19.263: FRAMEWIN_SetTitleVis() parameter list

FRAMEWIN_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

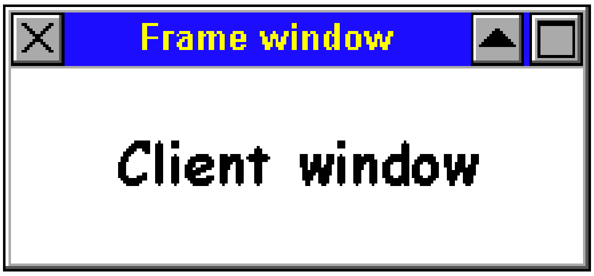
19.9.5 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_FrameWin.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_FrameWin.c:



19.10 GRAPH: Graph widget

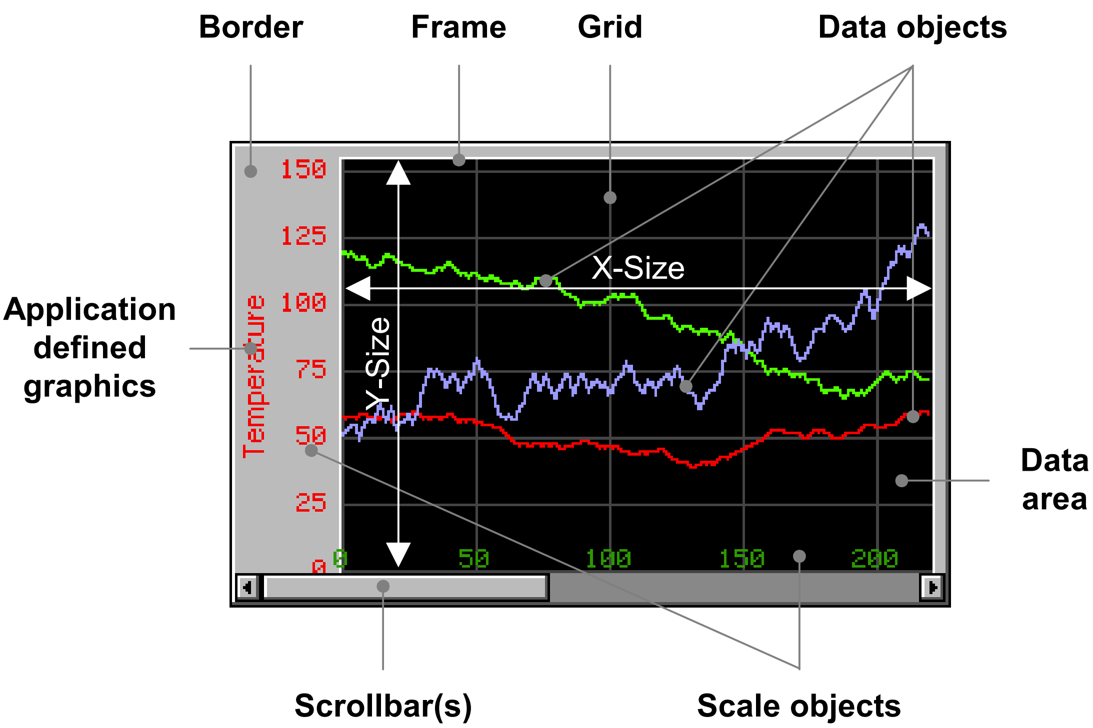
GRAPH widgets can be used to visualize data. Typical applications for GRAPH widgets are showing measurement values or the curve of a function graph. Multiple curves can be shown simultaneously. Horizontal and vertical scales can be used to label the curves. A grid with different horizontal and vertical spacing can be shown on the background. If the data array does not fit into the visible area of the graph, the widget can automatically show scroll bars which allow scrolling through large data arrays.

19.10.1 Structure of the graph widget

A GRAPH widget consists of different kinds objects:

- The GRAPH widget itself to which data objects and scale objects can be attached.
- Optionally one or more data objects.
- Optionally one or more scale objects.

The following diagram shows the detailed structure of a graph widget:



The following table explains the details of the diagram above:

Detail	Description
Border	The optional border is part of the graph widget.
Frame	A thin line around the data area, part of the graph widget.
Grid	Shown in the background of the data area, part of the graph widget.
Data area	Area, in which grid and data objects are shown.
Data object(s)	For each curve one data object should be added to the graph widget.
Application defined graphic	An application defined callback function can be used to draw any application defined text and/or graphics.
Scrollbar(s)	If the range of the data object is bigger than the data area of the graph widget, the graph widget can automatically show a horizontal and/or a vertical scroll bar.
Scale object(s)	Horizontal and vertical scales can be attached to the graph widget.
X-Size	X-Size of the data area.
Y-Size	Y-Size of the data area.

Table 19.264: Description of the graph widget structure

19.10.2 Creating and deleting a graph widget

The process of creating a GRAPH widget should be the following:

- Create the GRAPH widget and set the desired attributes.
- Create the data object(s).
- Attach the data object(s) to the GRAPH widget.
- Create the optional scale object(s).
- Attach the scale object(s) to the GRAPH widget.

Once attached to the graph widget the data and scale objects need not to be deleted by the application. This is done by the graph widget.

Example

The following shows a small example how to create and delete a GRAPH widget:

```
GRAPH_DATA_Handle  hData;
GRAPH_SCALE_Handle hScale;
WM_HWIN hGraph;
hGraph = GRAPH_CreateEx(10, 10, 216, 106, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_GRAPH0);
hData  = GRAPH_DATA_YT_Create(GUI_DARKGREEN, NumDataItems, aData0, MaxNumDataItems);
GRAPH_AttachData(hGraph, hData);
hScale = GRAPH_SCALE_Create(28, GUI_TA_RIGHT, GRAPH_SCALE_CF_VERTICAL, 20);
GRAPH_AttachScale(hGraph, hScale);
/*
  Do something with the widget...
*/
WM_DeleteWindow(hGraph);
```

19.10.3 Drawing process

As explained above a GRAPH widget consists of different parts and 'sub' objects. The following will explain, in which sequence the widget is drawn:

1. Filling the background with the background color.
2. Calling an optional callback routine. This makes it possible to draw for example a user defined grid.
3. Drawing the grid (if enabled).
4. Drawing the data objects and the border area.
5. Drawing the scale objects.
6. Calling an optional callback routine. This makes it possible to draw for example a user defined scale or some additional text and/or graphics.

19.10.4 Supported types of curves

The requirements for showing a curve with continuously updated measurement values can be different to the requirements when showing a function graph with X/Y coordinates. For that reason the widget currently supports 2 kinds of data objects, which are shown in the table below:

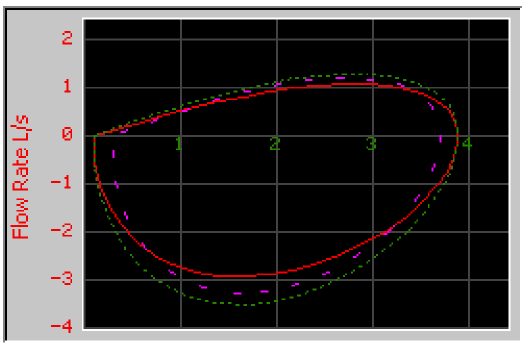
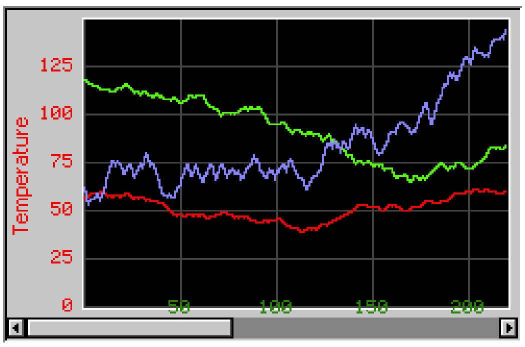
<table>
<tr><th>GRAPH_DATA_XY</th><th>GRAPH_DATA_YT</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 19.265: Supported types of curves

19.10.4.1 GRAPH_DATA_XY

This data object is used to show curves which consist of an array of points. The object data is drawn as a polyline. A typical application for using this data object is drawing a function graph.

19.10.4.2 GRAPH_DATA_YT

This data object is used to show curves with one Y-value for each X-position on the graph. A typical application for using this data object is showing a curve with continuously updated measurement values.

19.10.5 Configuration options

19.10.5.1 Graph widget

Type	Macro	Default	Description
N	GRAPH_BKCOLOR_DEFAULT	GUI_BLACK	Default background color of the data area.
N	GRAPH_BORDERCOLOR_DEFAULT	0xC0C0C0	Default background color of the border.
N	GRAPH_FRAMECOLOR_DEFAULT	GUI_WHITE	Default color of the thin frame line.
N	GRAPH_GRIDCOLOR_DEFAULT	GUI_DARKGRAY	Default color used to draw the grid.
N	GRAPH_GRIDSPACING_X_DEFAULT	50	Default horizontal spacing of the grid.
N	GRAPH_GRIDSPACING_Y_DEFAULT	50	Default vertical spacing of the grid.
N	GRAPH_BORDER_L_DEFAULT	0	Default size of the left border.
N	GRAPH_BORDER_T_DEFAULT	0	Default size of the top border.
N	GRAPH_BORDER_R_DEFAULT	0	Default size of the right border.
N	GRAPH_BORDER_B_DEFAULT	0	Default size of the bottom border.

Table 19.266: Graph configuration options

19.10.5.2 Scale object

Type	Macro	Default	Description
N	GRAPH_SCALE_TEXTCOLOR_DEFAULT	GUI_WHITE	Default text color.
S	GRAPH_SCALE_FONT_DEFAULT	&GUI_Font6x8	Default font used to draw the values.

Table 19.267: Scale configuration options

19.10.6 Predefined IDs

The following symbols define IDs which may be used to make GRAPH widgets distinguishable from creation: `GUI_ID_GRAPH0` - `GUI_ID_GRAPH3`

19.10.7 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

19.10.8 GRAPH API

The table below lists the available emWin GRAPH-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
Common routines	
GRAPH_AttachData()	Attaches a data object to a GRAPH widget.
GRAPH_AttachScale()	Attaches a scale object to a GRAPH widget.
GRAPH_CreateEx()	Creates a GRAPH widget.

Table 19.268: GRAPH API list

Routine	Description
GRAPH_CreateIndirect()	Creates a GRAPH widget from a resource table entry
GRAPH_CreateUser()	Creates a GRAPH widget using extra bytes as user data.
GRAPH_DetachData()	Detaches a data object from a GRAPH widget.
GRAPH_DetachScale()	Detaches a scale object from a GRAPH widget.
GRAPH_GetScrollValue()	Returns the current scroll value for the given scroll bar.
GRAPH_GetUserData()	Retrieves the data set with GRAPH_SetUserData().
GRAPH_SetAutoScrollbar()	Sets the automatic use of scroll bars.
GRAPH_SetBorder()	Sets the size (right, left, top and bottom) of the border.
GRAPH_SetColor()	Sets the color of the GRAPH widget.
GRAPH_SetGridDistX()	Sets the horizontal grid spacing.
GRAPH_SetGridDistY()	Sets the vertical grid spacing.
GRAPH_SetGridFixedX()	Fixes the grid in X-axis.
GRAPH_SetGridOffY()	Adds an offset to the y grid lines.
GRAPH_SetGridVis()	Enables the drawing of a grid.
GRAPH_SetLineStyleH()	Sets the line style for the horizontal grid lines.
GRAPH_SetLineStyleV()	Sets the line style for the vertical grid lines.
GRAPH_SetScrollValue()	Sets the scroll value for the given scroll bar.
GRAPH_SetUserData()	Sets the extra data of a GRAPH widget.
GRAPH_SetUserDraw()	Sets the user callback function.
GRAPH_SetVSizeX()	Sets the horizontal range of the GRAPH widget.
GRAPH_SetVSizeY()	Sets the vertical range of the GRAPH widget.
GRAPH_DATA_YT related routines	
GRAPH_DATA_YT_AddValue()	Adds one data item to the GRAPH_DATA_YT object.
GRAPH_DATA_YT_Clear()	Clears all data items of the GRAPH_DATA_YT object.
GRAPH_DATA_YT_Create()	Creates a GRAPH_DATA_YT object.
GRAPH_DATA_YT_Delete()	Deletes a GRAPH_DATA_YT object.
GRAPH_DATA_YT_MirrorX()	Mirrors the x-axis.
GRAPH_DATA_YT_SetAlign()	Sets the alignment of the given GRAPH_DATA_YT object.
GRAPH_DATA_YT_SetOffY()	Sets a vertical offset for drawing the data.
GRAPH_DATA_XY related routines	
GRAPH_DATA_XY_AddPoint()	Adds one point to the GRAPH_DATA_XY object.
GRAPH_DATA_XY_Create()	Creates a GRAPH_DATA_XY object.
GRAPH_DATA_XY_Delete()	Deletes a GRAPH_DATA_XY object.
GRAPH_DATA_XY_SetLineStyle()	Sets the line style used to draw the data.
GRAPH_DATA_XY_SetOffX()	Sets a horizontal offset for drawing the data.
GRAPH_DATA_XY_SetOffY()	Sets a vertical offset for drawing the data.
GRAPH_DATA_XY_SetOwnerDraw()	Sets the owner callback function.
GRAPH_DATA_XY_SetPenSize()	Sets the pen size used to draw the data.
Scale related routines	
GRAPH_SCALE_Create()	Creates a GRAPH_SCALE object.
GRAPH_SCALE_Delete()	Deletes a GRAPH_SCALE object.
GRAPH_SCALE_SetFactor()	Sets a calculation factor used to calculate from pixels to the desired unit.
GRAPH_SCALE_SetFont()	Sets the font used to draw the numbers.
GRAPH_SCALE_SetNumDecs()	Sets the number of digits of the fractional portion.
GRAPH_SCALE_SetOff()	Sets an optional offset which is added to the numbers.
GRAPH_SCALE_SetPos()	Sets the horizontal or vertical position of the scale.
GRAPH_SCALE_SetTextColor()	Sets the text color of the scale.
GRAPH_SCALE_SetTickDist()	Sets the distance in pixels between the tick marks.

Table 19.268: GRAPH API list

19.10.8.1 Common routines

GRAPH_AttachData()

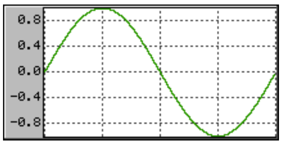
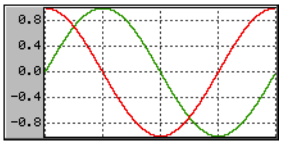
Before	After
	

Table 19.269: GRAPH_AttachData() before after screenshots

Description

Attaches a data object to an existing GRAPH widget.

Prototype

```
void GRAPH_AddGraph(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

Parameter	Description
hObj	Handle of widget
hData	Handle of the data object to be added to the widget. The data object should be created with `GRAPH_DATA_YT_Create()` or `GRAPH_DATA_XY_Create()`

Table 19.270: GRAPH_AttachData() parameter list

Additional information

Once attached to a GRAPH widget the application needs not to destroy the data object.
The GRAPH widget deletes all attached data objects when it is deleted.
For details about how to create data objects, refer to “GRAPH_DATA_YT_Create()” on page 573 and “GRAPH_DATA_XY_Create()” on page 576.

GRAPH_AttachScale()

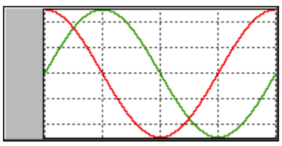
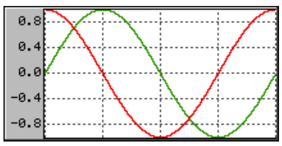
Before	After
	

Table 19.271: GRAPH_AttachScale() before after screenshots

Description

Attaches a scale object to an existing GRAPH widget.

Prototype

```
void GRAPH_AttachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

Parameter	Description
hObj	Handle of GRAPH widget
hScale	Handle of the scale to be added.

Table 19.272: GRAPH_AttachScale() parameter list

Additional information

Once attached to a GRAPH widget the application needs not to destroy the scale object. The GRAPH widget deletes all attached scale objects when it is deleted.
For details about how to create scale objects, refer to “GRAPH_SCALE_Create()” on page 580.

GRAPH_CreateEx()

Description

Creates a new GRAPH widget of a specified size at a specified location.

Prototype

```
GRAPH_Handle GRAPH_CreateEx(int      x0,       int y0,
                            int      xSize,    int ySize,
                            WM_HWIN hParent,  int WinFlags,
                            int      ExFlags,  int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new button window will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to “WM_CreateWindow()” on page 404 for a list of available parameter values).
ExFlags	See table below.
Id	Window Id of the widget.

Table 19.273: GRAPH_CreateEx() parameter list

Permitted values for parameter ExFlags	
GRAPH_CF_GRID_FIXED_X	This flag ’fixes’ the grid in X-axis. That means if horizontal scrolling is used, the grid remains on its position.

Return value

Handle of the created GRAPH widget; 0 if the function fails.

GRAPH_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `GRAPH_CreateEx()`.

GRAPH_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `GRAPH_CreateEx()` can be referred to.

GRAPH_DetachData()

Description

Detaches a data object from a GRAPH widget.

Prototype

```
void GRAPH_DetachData(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

Parameter	Description
hObj	Handle of GRAPH widget
hData	Handle of the data object to be detached from the widget.

Table 19.274: GRAPH_DetachData() parameter list

Additional information

Once detached from a GRAPH widget the application needs to destroy the data object.
Detaching a data object does not delete it. For more details about deleting data objects, refer to "GRAPH_DATA_YT_Delete()" on page 573 and "GRAPH_DATA_XY_Delete()" on page 577.

GRAPH_DetachScale()

Description

Detaches a scale object from a GRAPH widget.

Prototype

```
void GRAPH_DetachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

Parameter	Description
hObj	Handle of GRAPH widget
hScale	Handle of the scale object to be detached from the widget.

Table 19.275: GRAPH_DetachScale() parameter list

Additional information

Once detached from a GRAPH widget the application needs to destroy the scale object.
Detaching a scale object does not delete it. For more details about deleting scale objects, refer to "GRAPH_SCALE_Delete()" on page 581.

GRAPH_GetScrollValue()

Description

Returns the current scroll value for the given scroll bar.

Prototype

```
int GRAPH_GetScrollValue(GRAPH_Handle hObj, U8 Coord);
```

Parameter	Description
hObj	Handle of the GRAPH widget
Coord	See table below.

Table 19.276: GRAPH_GetScrollValue() parameter list

Permitted values for parameter Coord	
GUI_COORD_X	Get the horizontal scroll value.
GUI_COORD_Y	Get the vertical scroll value.

Return value

Current scroll value.
-1, if scroll value could not be determined.

GRAPH_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

GRAPH_SetAutoScrollbar()

Description

Sets the automatic use of scroll bars.

Prototype

```
void GRAPH_SetAutoScrollbar(GRAPH_Handle hObj, U8 Coord, U8 OnOff);
```

Parameter	Description
`hObj`	Handle of GRAPH widget
`Coord`	See table below.
`OnOff`	1 the scroll bar should be used automatically. 0, if the scroll bar should not be created automatically.

Table 19.277: GRAPH_SetAutoScrollbar() parameter list

Permitted values for parameter `Coord`	
`GUI_COORD_X`	Toggle automatic creation of the horizontal scroll bar.
`GUI_COORD_Y`	Toggle automatic creation of the vertical scroll bar.

GRAPH_SetBorder()

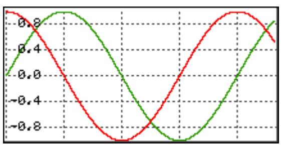
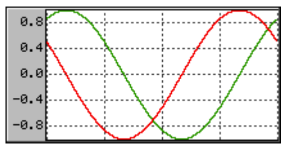
Before	After
	

Table 19.278: GRAPH_SetBorder() before after screenshots

Description

Sets the left, top, right and bottom border of the given GRAPH widget.

Prototype

```
void GRAPH_SetBorder(GRAPH_Handle hObj,
                     unsigned     BorderL, unsigned BorderT,
                     unsigned     BorderR, unsigned BorderB);
```

Parameter	Description
`hObj`	Handle of GRAPH widget.
`BorderL`	Size in pixels from the left border.
`BorderT`	Size in pixels from the top border.
`BorderR`	Size in pixels from the right border.
`BorderB`	Size in pixels from the bottom border.

Table 19.279: GRAPH_SetBorder() parameter list

Additional information

The border size is the number of pixels between the widget effect frame and the data area of the GRAPH widget. The frame, the thin line around the data area, is only visible if the border size is at least one pixel. For details about how to set the color of the border and the thin frame, refer to "GRAPH_SetColor()" on page 566.

GRAPH_SetColor()

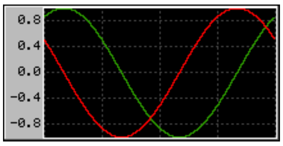
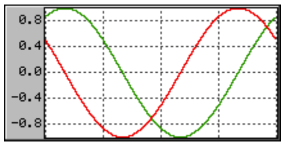
Before	After
	

Table 19.280: GRAPH_SetColor() before after screenshots

Description

Sets the desired color of the given GRAPH widget.

Prototype

```
GUI_COLOR GRAPH_SetColor(GRAPH_Handle hObj,   GUI_COLOR Color,
                         unsigned     Index);
```

Parameter	Description
`hObj`	Handle of GRAPH widget.
`Color`	Color to be used for the desired item.
`Index`	See table below.

Table 19.281: GRAPH_SetColor() parameter list

Permitted values for parameter `Index`	
`GRAPH_CI_BK`	Sets the background color.
`GRAPH_CI_BORDER`	Sets the color of the border area.
`GRAPH_CI_FRAME`	Sets the color of the thin frame line.
`GRAPH_CI_GRID`	Sets the color of the grid.

Return value

Previous color used for the desired item.

GRAPH_SetGridDistX()

GRAPH_SetGridDistY()

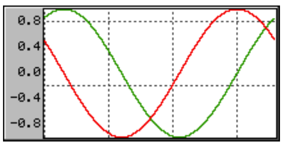
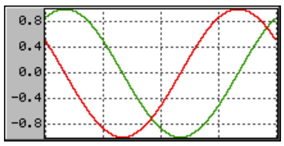
Before	After
	

Table 19.282: GRAPH_SetGridDistX / GRAPH_SetGridDistY() before after screenshots

Description

These functions set the distance from one grid line to the next.

Prototypes

```
unsigned GRAPH_SetGridDistX(GRAPH_Handle hObj, unsigned Value);
```



```
unsigned GRAPH_SetGridDistY(GRAPH_Handle hObj, unsigned Value);
```

Parameter	Description
hObj	Handle of GRAPH widget
Value	Distance in pixels from one grid line to the next, default is 50 pixel.

Table 19.283: GRAPH_SetGridDistX() / GRAPH_SetGridDistY() parameter list

Return value

Previous grid line distance.

Additional information

The first vertical grid line is drawn at the leftmost position of the data area and the first horizontal grid line is drawn at the bottom position of the data area, except an offset is used.

GRAPH_SetGridFixedX()

Description

Fixes the grid in X-axis.

Prototype

```
unsigned GRAPH_SetGridFixedX(GRAPH_Handle hObj, unsigned OnOff);
```

Parameter	Description
hObj	Handle of GRAPH widget.
OnOff	1 if grid should be fixed in X-axis, 0 if not (default).

Table 19.284: GRAPH_SetGridFixedX() parameter list

Return value

Previous value used

Additional information

In some situations it can be useful to fix the grid in X-axis. A typical application would be a YT-graph, to which continuously new values are added and horizontal scrolling is possible. In this case it could be desirable to fix the grid in the background.
For details about how to activate scrolling for a GRAPH widget, refer to "GRAPH_SetVSizeX()" on page 571.

GRAPH_SetGridOffY()

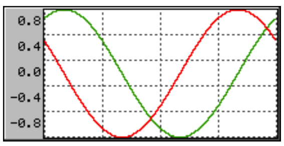
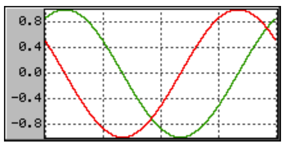
Before	After
	

Table 19.285: GRAPH_SetGridOffY() before after screenshots

Description

Adds an offset used to show the horizontal grid lines.

Prototype

```
unsigned GRAPH_SetGridOffY(GRAPH_Handle hObj, unsigned Value);
```

Parameter	Description
hObj	Handle of GRAPH widget.
Value	Offset to be used.

Table 19.286: GRAPH_SetGridOffY() parameter list

Return value

Previous offset used to draw the horizontal grid lines.

Additional information

When rendering the grid the widget starts drawing the horizontal grid lines from the bottom of the data area and uses the current spacing. In case of a zero point in the middle of the Y-axis it could happen, that there is no grid line in the middle. In this case the grid can be shifted in Y-axis by adding an offset with this function. A positive value shifts the grid down and negative values shifts it up.
For details about how to set the grid spacing, refer to the functions "GRAPH_SetGridDistX()" on page 566.

GRAPH_SetGridVis()

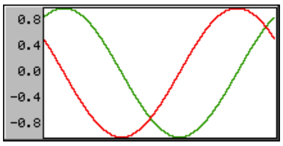
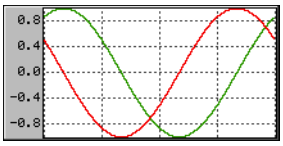
Before	After
	

Table 19.287: GRAPH_SetGridVis() before after screenshots

Description

Sets the visibility of the grid lines.

Prototype

```
unsigned GRAPH_SetGridVis(GRAPH_Handle hObj, unsigned OnOff);
```

Parameter	Description
hObj	Handle of GRAPH widget.
OnOff	1 if the grid should be visible, 0 if not (default).

Table 19.288: GRAPH_SetGridVis() parameter list

Return value

Previous value of the grid visibility.

GRAPH_SetLineStyleH()

GRAPH_SetLineStyleV()

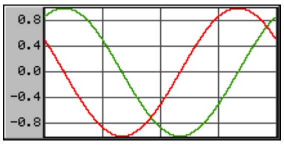
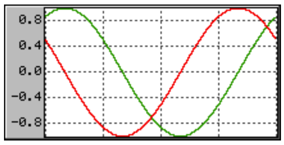
Before	After
	

Table 19.289: GRAPH_SetLineStyleH / GRAPH_SetLineStyleV() before after screenshots

Description

These functions are used to set the line style used to draw the horizontal and vertical grid lines.

Prototypes

```
U8 GRAPH_SetLineStyleH(GRAPH_Handle hObj, U8 LineStyle);
U8 GRAPH_SetLineStyleV(GRAPH_Handle hObj, U8 LineStyle);
```

Parameter	Description
`hObj`	Handle of GRAPH widget.
`LineStyle`	Line style to be used. For details about the supported line styles, refer to "GUI_SetLineStyle()" on page 147. Default is `GUI_LS_SOLID`.

Table 19.290: GRAPH_SetLineStyleH() / GRAPH_SetLineStyleV() parameter list

Return value

Previous line style used to draw the horizontal/vertical grid lines.

Additional information

Note that using other styles than `GUI_LS_SOLID` will need more time to show the grid.

GRAPH_SetScrollValue()

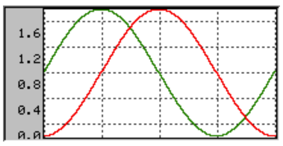
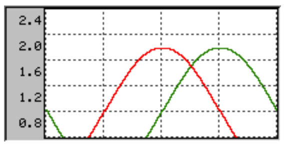
Before	After
	

Table 19.291: GRAPH_SetScrollValue() before after screenshots

Description

Sets the scroll value for the given scroll bar.

Prototype

```
void GRAPH_SetScrollValue(GRAPH_Handle hObj, U8 Coord, U32 Value);
```

Parameter	Description
hObj	Handle of GRAPH widget.
Coord	See table below.
Value	Scroll value to set.

Table 19.292: GRAPH_SetScrollValue() parameter list

Permitted values for parameter Coord	
GUI_COORD_X	Set the horizontal scroll value.
GUI_COORD_Y	Set the vertical scroll value.

GRAPH_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

GRAPH_SetUserDraw()

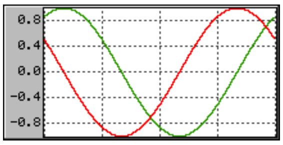
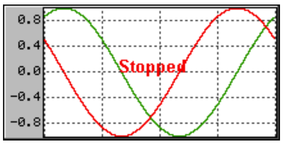
Before	After
	

Table 19.293: GRAPH_SetUserDraw() before after screenshots

Description

Sets the user draw function. This function is called by the widget during the drawing process to give the application the possibility to draw user defined data.

Prototype

```
void GRAPH_SetUserDraw(GRAPH_Handle hObj,
                       void (* pUserDraw)(WM_HWIN hObj, int Stage));
```

Parameter	Description
hObj	Handle of GRAPH widget
pUserDraw	Pointer to application function to be called by the widget during the drawing process.

Table 19.294: GRAPH_SetUserDraw() parameter list

Permitted values for parameter Stage	
GRAPH_DRAW_FIRST	Gives the application the possibility to perform drawing operations at the beginning of the drawing process.
GRAPH_DRAW_AFTER_BORDER	Gives the application the possibility to perform drawing operations after the border was drawn.
GRAPH_DRAW_LAST	Gives the application the possibility to perform drawing operations after

Additional information

The user draw function is called at the beginning after filling the background of the data area and after drawing all GRAPH items like described at the beginning of the chapter. On the first call the clipping region is limited to the data area. On the last call it is limited to the complete GRAPH widget area except the effect frame.

Example

The following small example shows the use of a user draw function:

```
static void _UserDraw(WM_HWIN hWin, int Stage) {
  switch (Stage) {
  case GRAPH_DRAW_FIRST:
    //
    // Draw for example a user defined grid...
    //
    break;
  case GRAPH_DRAW_LAST:
    //
    // Draw for example a user defined scale or additional text...
    //
    break;
  }
}

static void _CreateGraph(void) {
  WM_HWIN hGraph;
  hGraph = GRAPH_CreateEx(10, 10, 216, 106, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_GRAPH0);
  GRAPH_SetUserDraw(hGraph, _UserDraw);  // Enable user draw
  ...
}
```

GRAPH_SetVSizeX()

GRAPH_SetVSizeY()

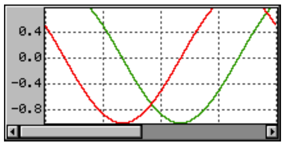
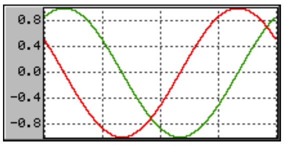
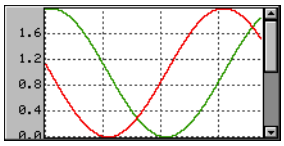
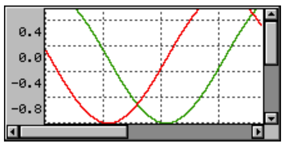
Before	After
	
	
	

Table 19.295: GRAPH_SetVSizeX / GRAPH_SetVSizeY() before after screenshots

Description

The functions set the virtual size in X and Y-axis.

Prototypes

```
unsigned GRAPH_SetVSizeX(GRAPH_Handle hObj, unsigned Value);
unsigned GRAPH_SetVSizeY(GRAPH_Handle hObj, unsigned Value);
```

Parameter	Description
`hObj`	Handle of GRAPH widget.
`Value`	Virtual size in pixels in X or Y axis.

Table 19.296: GRAPH_SetVSizeX() / GRAPH_SetVSizeY() parameter list

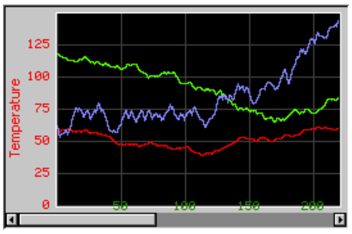
Return value

Previous virtual size of the widgets data area in X or Y-axis.

Additional information

If the widgets virtual size is bigger than the visible size of the data area, the widget automatically shows a scroll bar. If for example a data object, created by the function `GRAPH_DATA_YT_Create()`, contains more data than can be shown in the data area, the function `GRAPH_SetVSizeX()` can be used to enable scrolling. A function call like `GRAPH_SetVSizeX(NumDataItems)` enables the horizontal scroll bar, provided that the number of data items is bigger than the X-size of the visible data area.

19.10.8.2GRAPH_DATA_YT related routines



GRAPH_DATA_YT_AddValue()

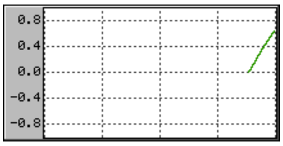
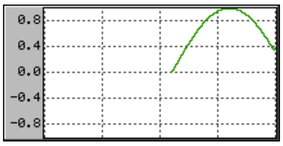
Before	After
	

Table 19.297: GRAPH_DATA_YT_AddValue() before after screenshots

Description

Adds a new data item to a GRAPH_DATA_YT object.

Prototype

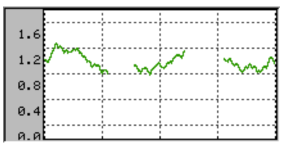
```
void GRAPH_DATA_YT_AddValue(GRAPH_DATA_Handle hDataObj, I16 Value);
```

Parameter	Description
`hDataObj`	Handle of data object.
`Value`	Value to be added to the data object.

Table 19.298: GRAPH_DATA_YT_AddValue() parameter list

Additional information

The given data value is added to the data object. If the data object is 'full', that means it contains as many data items as specified in parameter `MaxNumItems` during the creation, it first shifts the data items by one before adding the new value. So the first data item is shifted out when adding a data item to a 'full' object.
The value 0x7FFF can be used to handle invalid data values. These values are excluded when drawing the GRAPH. The following screenshot shows a graph with 2 gaps of invalid data:



GRAPH_DATA_YT_Clear()

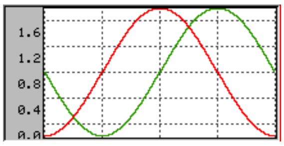
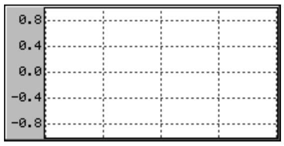
Before	After
	

Table 19.299: GRAPH_DATA_YT_Clear() before after screenshots

Description

Clears all data items of the data object.

Prototype

```
void GRAPH_DATA_YT_Clear(GRAPH_DATA_Handle hDataObj);
```

Parameter	Description
`hDataObj`	Handle of data object.

Table 19.300: GRAPH_DATA_YT_Clear() parameter list

GRAPH_DATA_YT_Create()

Description

Creates a GRAPH_DATA_YT object. This kind of object requires for each point on the x-axis a value on the y-axis. Typically used for time related graphs.

Prototype

```
GRAPH_DATA_Handle GRAPH_DATA_YT_Create(GUI_COLOR   Color,
                                       unsigned    MaxNumItems,
                                       I16       * pData,
                                       unsigned    NumItems);
```

Parameter	Description
`Color`	Color to be used to draw the data.
`MaxNumItems`	Maximum number of data items.
`pData`	Pointer to data to be added to the object. The pointer should point to an array of `I16` values.
`NumItems`	Number of data items to be added.

Table 19.301: GRAPH_DATA_YT_Create() parameter list

Return value

Handle of data object if creation was successful, otherwise 0.

Additional information

The last data item is shown at the rightmost column of the data area. If a data object contains more data as can be shown in the data area of the GRAPH widget, the function `GRAPH_SetVSizeX()` can be used to show a scroll bar which makes it possible to scroll through large data objects.
Once attached to a GRAPH widget a data object needs not to be deleted by the application. This is automatically done during the deletion of the GRAPH widget.

GRAPH_DATA_YT_Delete()

Description

Deletes the given data object.

Prototype

```
void GRAPH_DATA_YT_Delete(GRAPH_DATA_Handle hDataObj);
```

Parameter	Description
hDataObj	Handle of data object to be deleted.

Table 19.302: GRAPH_DATA_YT_Delete() parameter list

Additional information

When a GRAPH widget is deleted it deletes all currently attached data objects. So the application needs only to delete unattached data objects.

GRAPH_DATA_YT_MirrorX()

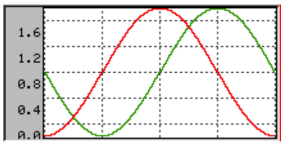
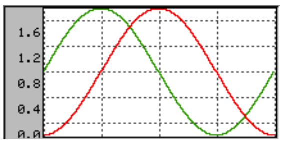
Before	After
	

Table 19.303: GRAPH_DATA_YT_MirrorX() before after screenshots

Description

Mirrors the x-axis of the widget.

Prototype

```
void GRAPH_DATA_YT_MirrorX(GRAPH_DATA_Handle hDataObj, int Value);
```

Parameter	Description
hDataObj	Handle of data object.
OnOff	1 for mirroring the x-axis, 0 for default view.

Table 19.304: GRAPH_DATA_YT_MirrorX() parameter list

Additional information

Per default the data is drawn from the right to the left. After calling this function the data is drawn from the left to the right.

GRAPH_DATA_YT_SetAlign()

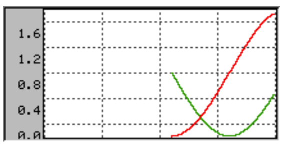
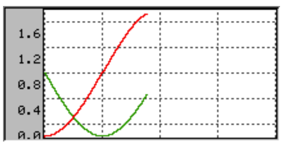
Before	After
	

Table 19.305: GRAPH_DATA_YT_SetAlign() before after screenshots

Description

Sets the alignment of the data.

Prototype

```
void GRAPH_DATA_YT_SetAlign(GRAPH_DATA_Handle hDataObj, int Align);
```

Parameter	Description
hDataObj	Handle of data object.
Align	See table below.

Table 19.306: GRAPH_DATA_YT_SetAlign() parameter list

Permitted values for parameter Align	
GRAPH_ALIGN_RIGHT	The data is aligned at the right edge (default).
GRAPH_ALIGN_LEFT	The data is aligned at the left edge.

GRAPH_DATA_YT_SetOffY()

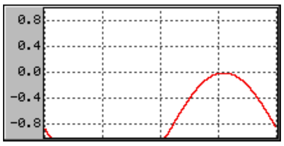
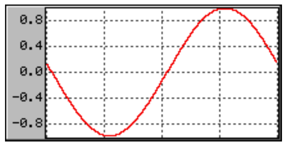
Before	After
	

Table 19.307: GRAPH_DATA_YT_SetOffY() before after screenshots

Description

Sets a vertical offset used to draw the object data.

Prototype

```
void GRAPH_DATA_YT_SetOffY(GRAPH_DATA_Handle hDataObj, int Off);
```

Parameter	Description
hDataObj	Handle of data object.
Off	Vertical offset which should be used to draw the data.

Table 19.308: GRAPH_DATA_YT_SetOffY() parameter list

Additional information

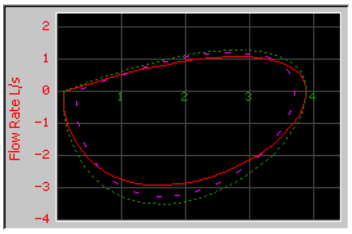
The vertical range of data, which is shown by the data object, is the range (0) - (Y-size of data area - 1). In case of using a scroll bar the current scroll position is added to the range.

Example

If for example the visible data range should be -200 to -100 the data needs to be shifted in positive direction by 200 pixels:

```
GRAPH_DATA_YT_SetOffY(hDataObj, 200);
```


19.10.8.3GRAPH_DATA_XY related routines



GRAPH_DATA_XY_AddPoint()

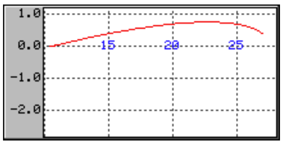
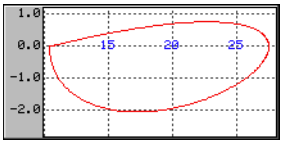
Before	After
	

Table 19.309: GRAPH_DATA_XY_AddPoint() before after screenshots

Description

Adds a new data item to a GRAPH_DATA_XY object.

Prototype

```
void GRAPH_DATA_XY_AddPoint(GRAPH_DATA_Handle hDataObj, GUI_POINT * pPoint);
```

Parameter	Description
`hDataObj`	Handle of data object.
`pPoint`	Pointer to a GUI_POINT structure to be added to the data object.

Table 19.310: GRAPH_DATA_XY_AddPoint() parameter list

Additional information

The given point is added to the data object. If the data object is 'full', that means it contains as many points as specified in parameter `MaxNumItems` during the creation, it first shifts the data items by one before adding the new point. So the first point is shifted out when adding a new point to a 'full' object.

GRAPH_DATA_XY_Create()

Description

Creates a GRAPH_DATA_XY object. This kind of object is able to store any pairs of values which will be connected by adding order.

Prototype

```
GRAPH_DATA_Handle GRAPH_DATA_XY_Create(GUI_COLOR   Color,
                                       unsigned    MaxNumItems,
                                       GUI_POINT * pData,
                                       unsigned    NumItems);
```

Parameter	Description
`Color`	Color to be used to draw the data.

Table 19.311: GRAPH_DATA_XY_Create() parameter list

Parameter	Description
`MaxNumItems`	Maximum number of points.
`pData`	Pointer to data to be added to the object. The pointer should point to a `GUI_POINT` array.
`NumItems`	Number of points to be added.

Table 19.311: GRAPH_DATA_XY_Create() parameter list

Return value

Handle of data object if creation was successful, otherwise 0.

Additional information

Once attached to a GRAPH widget a data object needs not to be deleted by the application. This is automatically done during the deletion of the GRAPH widget.

GRAPH_DATA_XY_Delete()

Description

Deletes the given data object.

Prototype

```
void GRAPH_DATA_XY_Delete(GRAPH_DATA_Handle hDataObj);
```

Parameter	Description
`hDataObj`	Data object to be deleted.

Table 19.312: GRAPH_DATA_XY_Delete() parameter list

Additional information

When a GRAPH widget is deleted it deletes all currently attached data objects. So the application needs only to delete unattached data objects.

GRAPH_DATA_XY_SetOffX()

GRAPH_DATA_XY_SetOffY()

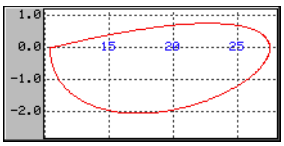
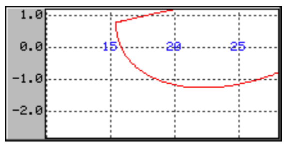
Before	After
	

Table 19.313: GRAPH_DATA_XY_SetOffX / GRAPH_DATA_XY_SetOffY() before after screenshots

Description

Sets a vertical or horizontal offset used to draw the polyline.

Prototype

```
void GRAPH_DATA_XY_SetOffX(GRAPH_DATA_Handle hDataObj, int Off);
void GRAPH_DATA_XY_SetOffY(GRAPH_DATA_Handle hDataObj, int Off);
```

Parameter	Description
`hDataObj`	Handle of data object.
`Off`	Horizontal/vertical offset which should be used to draw the polyline.

Table 19.314: GRAPH_DATA_XY_SetOffX() / GRAPH_DATA_XY_SetOffY() parameter list

Additional information

The range of data shown by the data object is (0, 0) - (X-size of data area - 1, Y-size of data area - 1). In case of using scroll bars the current scroll position is added to the respective range. To make other ranges of data visible this functions should be used to set an offset, so that the data is in the visible area.

Example

If for example the visible data range should be (100, -1200) - (200, -1100) the following offsets need to be used:

```
GRAPH_DATA_XY_SetOffX(hDataObj, -100);
GRAPH_DATA_XY_SetOffY(hDataObj, 1200);
```

GRAPH_DATA_XY_SetOwnerDraw()

Description

Sets the owner callback function. This function is called by the widget during the drawing process to give the application the possibility to draw additional items on top of the widget.

Prototype

```
void GRAPH_DATA_XY_SetOwnerDraw(GRAPH_DATA_Handle hDataObj,
  void (* pOwnerDraw)(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo));
```

Parameter	Description
hDataObj	Handle of data object to be deleted.
pOwnerDraw	Pointer to application function to be called by the widget during the drawing process.

Table 19.315: GRAPH_DATA_XY_SetOwnerDraw() parameter list

Additional information

The owner draw function is called after background, scales and grid lines are drawn.

Example

The following code snippet shows an example of an user draw function:

```
static int _cbData(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW:
    GUI_DrawRect(pDrawItemInfo->x0 - 3, pDrawItemInfo->y0 - 3,
                 pDrawItemInfo->x0 + 3, pDrawItemInfo->y0 + 3);
    break;
  }
  return 0;
}

void MainTask(void) {
  WM_HWIN           hGraph;
  GRAPH_DATA_Handle hData;
  GUI_Init();
  hGraph = GRAPH_CreateEx (140, 100, 171, 131, 0, WM_CF_SHOW, 0, GUI_ID_GRAPH0);
  hData  = GRAPH_DATA_XY_Create(USER_DEFINED_COLOR, 126, 0, 0);
  GRAPH_DATA_XY_SetOwnerDraw(hData, _cbData);
}
```


GRAPH_DATA_XY_SetLineStyle()

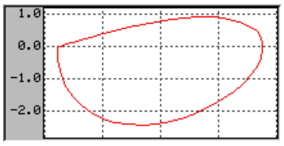
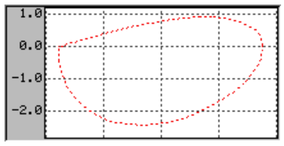
Before	After
	

Table 19.316: GRAPH_DATA_XY_SetLineStyle() before after screenshots

Description

Sets the line style used to draw the polyline.

Prototype

```
void GRAPH_DATA_XY_SetLineStyle(GRAPH_DATA_Handle hDataObj, U8 LineStyle);
```

Parameter	Description
`hDataObj`	Handle of data object.
`LineStyle`	New line style to be used. For details about the supported line styles, refer to “GUI_SetLineStyle()” on page 147.

Table 19.317: GRAPH_DATA_XY_SetLineStyle() parameter list

Limitations

Note that only curves with line style `GUI_LS_SOLID` (default) can be drawn with a pen size >1.

GRAPH_DATA_XY_SetPenSize()

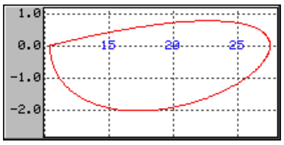
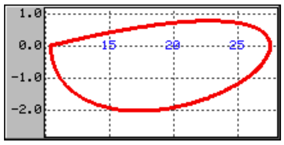
Before	After
	

Table 19.318: GRAPH_DATA_XY_SetPenSize() before after screenshots

Description

Sets the pen size used to draw the polyline.

Prototype

```
void GRAPH_DATA_XY_SetPenSize(GRAPH_DATA_Handle hDataObj, U8 PenSize);
```

Parameter	Description
`hDataObj`	Handle of data object.
`PenSize`	Pen size which should be used to draw the polyline.

Table 19.319: GRAPH_DATA_XY_SetPenSize() parameter list

Limitations

Note that only curves with line style `GUI_LS_SOLID` (default) can be drawn with a pen size >1.

19.10.8.4Scale related routines

The GRAPH widget supports horizontal and vertical scales for labeling purpose. The following describes the available functions for using scales.

GRAPH_SCALE_Create()

Description

Creates a GRAPH_SCALE object.

Prototype

```
GRAPH_SCALE_Handle GRAPH_SCALE_Create(int      Pos,    int      TextAlign,
                                      unsigned Flags,  unsigned TickDist);
```

Parameter	Description
Pos	Position relative to the left/top edge of the GRAPH widget.
TextAlign	Text alignment used to draw the numbers. See table below.
Flags	See table below.
TickDist	Distance from one tick mark to the next.

Table 19.320: GRAPH_SCALE_Create() parameter list

Permitted values for parameter TextAlign (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right.
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

Permitted values for parameter Flags	
GRAPH_SCALE_CF_HORIZONTAL	Creates a horizontal scale object.
GRAPH_SCALE_CF_VERTICAL	Creates a vertical scale object.

Return value

Handle of the scale object if creation was successful, otherwise 0.

Additional information

A horizontal scale object starts labeling from the bottom edge of the data area to the top and a vertical scale object from the left edge (horizontal scale) to the right, where the first position is the zero point. The parameter `TickDist` specifies the distance between the numbers.
The parameter `Pos` specifies in case of a horizontal scale the vertical distance in pixels from the top edge of the GRAPH widget to the scale text. In case of a vertical scale the parameter specifies the horizontal distance from the left edge of the GRAPH widget to the horizontal text position. Note that the actual text position also depends on the text alignment specified with parameter `TextAlign`.
The scale object draws a number for each position which is within the data area. In case of a horizontal scale there is one exception: If the first position is 0 no number is drawn at this position.
Once attached to a GRAPH widget a scale object needs not to be deleted by the application. This is automatically done during the deletion of the GRAPH widget.

GRAPH_SCALE_Delete()

Description

Deletes the given scale object.

Prototype

```
void GRAPH_SCALE_Delete(GRAPH_SCALE_Handle hScaleObj);
```

Parameter	Description
hScaleObj	Handle of scale object to be deleted.

Table 19.321: GRAPH_SCALE_Delete() parameter list

Additional information

When a GRAPH widget is deleted it deletes all currently attached scale objects. So the application needs only to delete unattached scale objects.

GRAPH_SCALE_SetFactor()

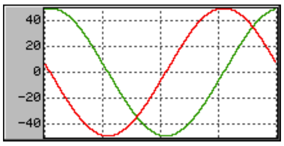
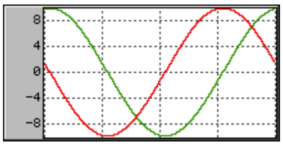
Before	After
	

Table 19.322: GRAPH_SCALE_SetFactor() before after screenshots

Description

Sets a factor used to calculate the numbers to be drawn.

Prototype

```
float GRAPH_SCALE_SetFactor(GRAPH_SCALE_Handle hScaleObj, float Factor);
```

Parameter	Description
hScaleObj	Handle of scale object.
Factor	Factor to be used to calculate the number.

Table 19.323: GRAPH_SCALE_SetFactor() parameter list

Return value

Old factor used to calculate the numbers.

Additional information

Without using a factor the unit of the scale object is 'pixel'. So the given factor should convert the pixel value to the desired unit.

GRAPH_SCALE_SetFont()

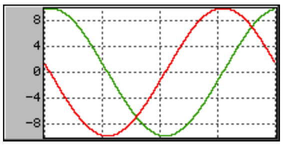
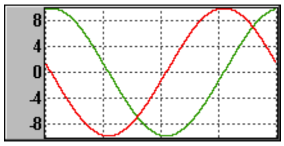
Before	After
	

Table 19.324: GRAPH_SCALE_SetFont() before after screenshots

Description

Sets the font used to draw the scale numbers.

Prototype

```
const GUI_FONT * GRAPH_SCALE_SetFont(GRAPH_SCALE_Handle   hScaleObj,
                                     const GUI_FONT     * pFont);
```

Parameter	Description
hScaleObj	Handle of scale object.
pFont	Font to be used.

Table 19.325: GRAPH_SCALE_SetFont() parameter list

Return value

Previous used font used to draw the numbers.

GRAPH_SCALE_SetNumDecs()

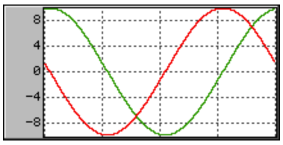
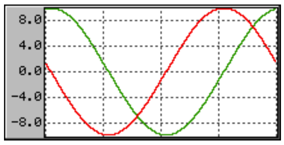
Before	After
	

Table 19.326: GRAPH_SCALE_SetNumDecs() before after screenshots

Description

Sets the number of post decimal positions to be shown.

Prototype

```
int GRAPH_SCALE_SetNumDecs(GRAPH_SCALE_Handle hScaleObj, int NumDecs);
```

Parameter	Description
hScaleObj	Handle of scale object.
NumDecs	Number of post decimal positions.

Table 19.327: GRAPH_SCALE_SetNumDecs() parameter list

Return value

Previous number of post decimal positions.

GRAPH_SCALE_SetOff()

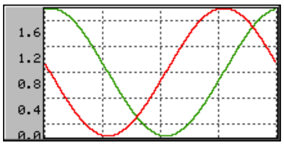
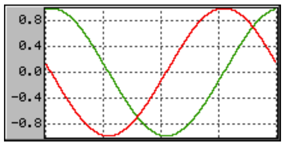
Before	After
	

Table 19.328: GRAPH_SCALE_SetOff() before after screenshots

Description

Sets an offset used to 'shift' the scale object in positive or negative direction.

Prototype

```
int GRAPH_SCALE_SetOff(GRAPH_SCALE_Handle hScaleObj, int Off);
```

Parameter	Description
hScaleObj	Handle of scale object.
Off	Offset used for drawing the scale.

Table 19.329: GRAPH_SCALE_SetOff() parameter list

Return value

Previous used offset.

Additional information

As described under the function GRAPH_SCALE_Create() a horizontal scale object starts labeling from the bottom edge of the data area to the top and a vertical scale object from the left edge (horizontal scale) to the right, where the first position is the zero point. In many situations it is not desirable, that the first position is the zero point. If the scale should be 'shifted' in positive direction, a positive offset should be added, for negative direction a negative value.

GRAPH_SCALE_SetPos()

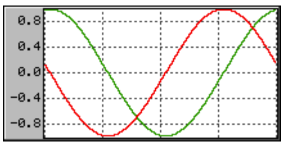
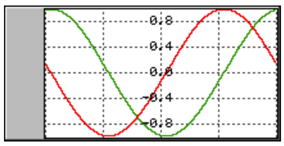
Before	After
	

Table 19.330: GRAPH_SCALE_SetPos() before after screenshots

Description

Sets the position for showing the scale object within the GRAPH widget.

Prototype

```
int GRAPH_SCALE_SetPos(GRAPH_SCALE_Handle hScaleObj, int Pos);
```

Parameter	Description
hScaleObj	Handle of scale object.
Pos	Position, at which the scale should be shown.

Table 19.331: GRAPH_SCALE_SetPos() parameter list

Return value

Previous position of the scale object.

Additional information

The parameter `Pos` specifies in case of a horizontal scale the vertical distance in pixels from the top edge of the GRAPH widget to the scale text. In case of a vertical scale the parameter specifies the horizontal distance from the left edge of the GRAPH widget to the horizontal text position. Note that the actual text position also depends on the text alignment of the scale object.

GRAPH_SCALE_SetTextColor()

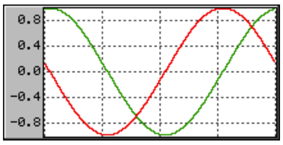
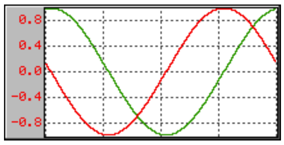
Before	After
	

Table 19.332: GRAPH_SCALE_SetTextColor() before after screenshots

Description

Sets the text color used to draw the numbers.

Prototype

```
GUI_COLOR GRAPH_SCALE_SetTextColor(GRAPH_SCALE_Handle hScaleObj,
                                   GUI_COLOR          Color);
```

Parameter	Description
hScaleObj	Handle of scale object.
Color	Color to be used to show the numbers.

Table 19.333: GRAPH_SCALE_SetTextColor() parameter list

Return value

Previous color used to show the numbers.

GRAPH_SCALE_SetTickDist()

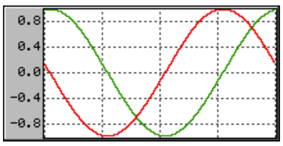
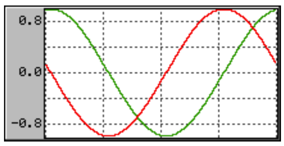
Before	After
	

Table 19.334: GRAPH_SCALE_SetTickDist() before after screenshots

Description

Sets the distance from one number to the next.

Prototype

```
unsigned GRAPH_SCALE_SetTickDist(GRAPH_SCALE_Handle hScaleObj,
```



```
unsigned            Dist);
```

Parameter	Description
hScaleObj	Handle of scale object.
Dist	Distance in pixels between the numbers.

Table 19.335: GRAPH_SCALE_SetTickDist() parameter list

Return value

Previous distance between the numbers.

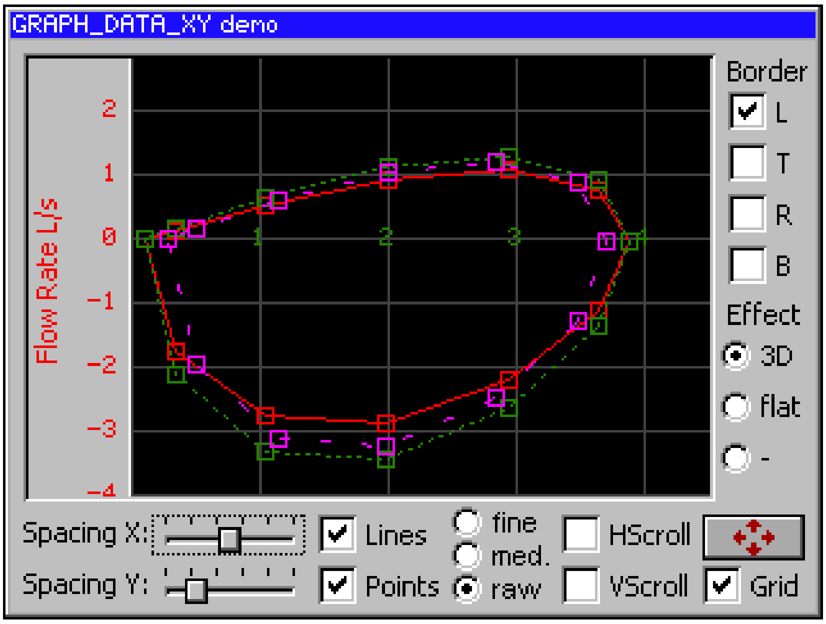
19.10.9 Examples

The `Sample` folder contains the following examples which show how the widget can be used:

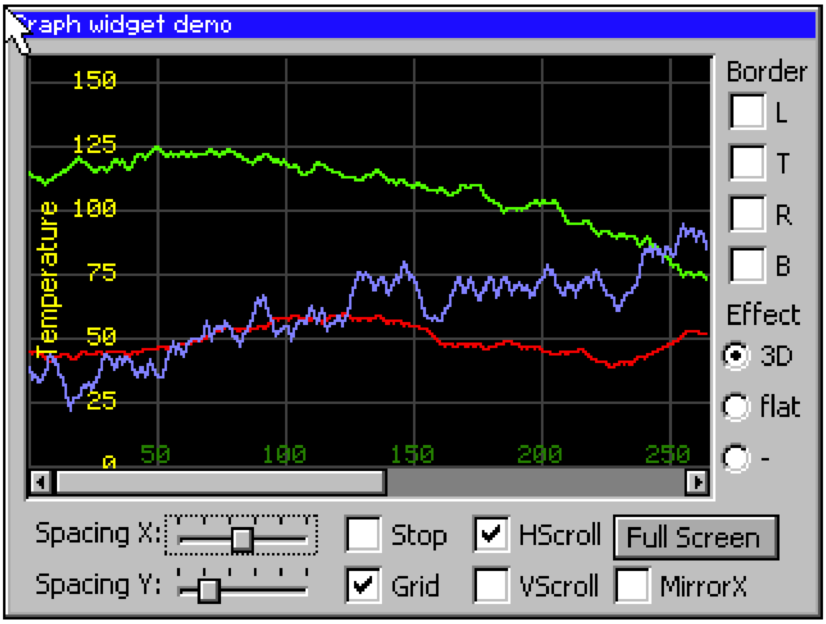
- `WIDGET_GraphXY.c`
- `WIDGET_GraphYT.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_GraphXY.c:



Screenshot of WIDGET_GraphYT.c:



19.11 HEADER: Header widget

HEADER widgets are used to label columns of a table:

If a pointer input device (PID) is used, the width of the HEADER items can be managed by dragging the dividers by the PID.

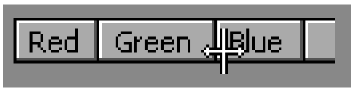
Behavior with mouse

If mouse support is enabled, the cursor is on and the PID is moved nearby a divider the cursor will change to signal, that the divider can be dragged at the current position.

Behavior with touch screen

If the widget is pressed nearby a divider and the cursor is on the cursor will change to signal, that the divider can now be dragged.

Screenshot of drag-able divider



Predefined cursors

There are 2 predefined cursors as shown below:

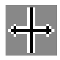
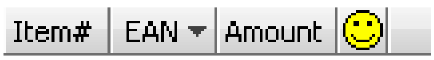
GUI_CursorHeaderM (default)	GUI_CursorHeaderMI
	

Table 19.336: Predefined cursors

You can also create and use your own cursors when using a HEADER widget as described later in this chapter.

Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter “Skinning” on page 871.

19.11.1 Configuration options

Type	Macro	Default	Description
N	HEADER_BKCOLOR_DEFAULT	0xAAAAAA	Default value of background color
S	HEADER_CURSOR_DEFAULT	&GUI_CursorHeaderM	Default cursor
S	HEADER_FONT_DEFAULT	&GUI_Font13_1	Default font
N	HEADER_BORDER_H_DEFAULT	2	Horizontal space between text and border
N	HEADER_BORDER_V_DEFAULT	0	Vertical space between text and border
B	HEADER_SUPPORT_DRAG	1	Enable/disable dragging support
N	HEADER_TEXTCOLOR_DEFAULT	GUI_BLACK	Default value of text color

Table 19.337: Configuration options

19.11.2 Notification codes

The following events are sent from a HEADER widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget without releasing.

Table 19.338: Notification codes

19.11.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

19.11.4 HEADER API

The table below lists the available emWin HEADER-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
HEADER_AddItem()	Adds one item at the right side
HEADER_Create()	Creates a HEADER widget. (Obsolete)
HEADER_CreateAttached()	Creates a HEADER widget attached to a window
HEADER_CreateEx()	Creates a HEADER widget.
HEADER_CreateIndirect()	Creates a HEADER widget from a resource table entry
HEADER_CreateUser()	Creates a HEADER widget using extra bytes as user data.
HEADER_GetDefaultBkColor()	Returns the default background color
HEADER_GetDefaultBorderH()	Returns the value of the horizontal spacing.
HEADER_GetDefaultBorderV()	Returns the value of the vertical spacing.
HEADER_GetDefaultCursor()	Returns the a pointer to the default cursor.
HEADER_GetDefaultFont()	Returns a pointer to the default font.
HEADER_GetDefaultTextColor()	Returns the default text color.
HEADER_GetHeight()	Returns the height of the widget.
HEADER_GetItemWidth()	Returns the item width.
HEADER_GetNumItems()	Returns the number of items.
HEADER_GetUserData()	Retrieves the data set with HEADER_SetUserData().
HEADER_SetBitmap()	Sets the bitmap used when displaying the given item.
HEADER_SetBitmapEx()	Sets the bitmap used when displaying the given item.
HEADER_SetBkColor()	Sets the background color of the widget.
HEADER_SetBMP()	Sets the bitmap used when displaying the given item.

Table 19.339: HEADER API list

Routine	Description
HEADER_SetBMPEx()	Sets the bitmap used when displaying the given item.
HEADER_SetDefaultBkColor()	Sets the default background color.
HEADER_SetDefaultBorderH()	Sets the default value for the horizontal spacing.
HEADER_SetDefaultBorderV()	Sets the default value for the vertical spacing.
HEADER_SetDefaultCursor()	Sets the default cursor.
HEADER_SetDefaultFont()	Sets the default font.
HEADER_SetDefaultTextColor()	Sets the default text color.
HEADER_SetDragLimit()	Sets the limit for dragging the items on or off.
HEADER_SetFont()	Sets the font of the widget.
HEADER_SetHeight()	Sets the height of the widget.
HEADER_SetItemText()	Sets the text of a given item.
HEADER_SetItemWidth()	Sets the width of a given item.
HEADER_SetStreamedBitmap()	Sets the bitmap used when displaying the given item.
HEADER_SetStreamedBitmapEx()	Sets the bitmap used when displaying the given item.
HEADER_SetTextAlign()	Sets the alignment of the given item.
HEADER_SetTextColor()	Sets the Text color of the widget.
HEADER_SetUserData()	Sets the extra data of a HEADER widget.

Table 19.339: HEADER API list

HEADER_AddItem()

Description

Adds an item to an already existing HEADER widget.

Prototype

```
void HEADER_AddItem(HEADER_Handle   hObj, int Width,
                    const char    * s,    int Align);
```

Parameter	Description
hObj	Handle of HEADER widget
Width	Width of the new item
s	Text to be displayed
Align	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

Table 19.340: HEADER_AddItem() parameter list

Permitted values for parameter Align (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right (default).
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

Additional information

The Width-parameter can be 0. If Width = 0 the width of the new item will be calculated by the given text and by the default value of the horizontal spacing.

HEADER_Create()

(Obsolete, HEADER_CreateEx() should be used instead)

Description

Creates a HEADER widget of a specified size at a specified location.

Prototype

```
HEADER_Handle HEADER_Create(int      x0,      int y0,
                            int      xSize,   int ySize,
                            WM_HWIN hParent, int Id,
                            int      Flags,   int SpecialFlags);
```

Parameter	Description
x0	Leftmost pixel of the HEADER widget (in parent coordinates).
y0	Topmost pixel of the HEADER widget (in parent coordinates).
xSize	Horizontal size of the HEADER widget (in pixels).
ySize	Vertical size of the HEADER widget (in pixels).
hParent	Handle of the parent window
Id	Id of the new HEADER widget
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
SpecialFlags	(Reserved for later use)

Table 19.341: HEADER_Create() parameter list

Return value

Handle of the created HEADER widget; 0 if the function fails.

HEADER_CreateAttached()

Description

Creates a HEADER widget which is attached to an existing window.

Prototype

```
HEADER_Handle HEADER_CreateAttached(WM_HWIN hParent,       int Id,
                                    int     SpecialFlags);
```

Parameter	Description
hObj	Handle HEADER of widget
Id	Id of the HEADER widget
SpecialFlags	(Not used, reserved for later use)

Table 19.342: HEADER_CreateAttached() parameter list

Return value

Handle of the created HEADER widget; 0 if the function fails.

Additional information

An attached HEADER widget is essentially a child window which will position itself on the parent window and operate accordingly.

HEADER_CreateEx()

Description

Creates a HEADER widget of a specified size at a specified location.

Prototype

```
HEADER_Handle HEADER_CreateEx(int      x0,      int y0,
                              int      xSize,   int ySize,
                              WM_HWIN  hParent, int WinFlags,
                              int      ExFlags, int Id);
```

Parameter	Description
`x0`	Leftmost pixel of the widget (in parent coordinates).
`y0`	Topmost pixel of the widget (in parent coordinates).
`xSize`	Horizontal size of the widget (in pixels).
`ySize`	Vertical size of the widget (in pixels).
`hParent`	Handle of parent window. If 0, the new HEADER widget will be a child of the desktop (top-level window).
`WinFlags`	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
`ExFlags`	Not used, reserved for future use.
`Id`	Window ID of the widget.

Table 19.343: HEADER_CreateEx() parameter list

Return value

Handle of the created HEADER widget; 0 if the function fails.

HEADER_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. For details the function `<WIDGET>_CreateIndirect()` should be referred to. The element `Flags` is used according to the parameter `WinFlags` of the function `HEADER_CreateEx()`. The element `Para` is not used.

HEADER_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `HEADER_CreateEx()` can be referred to.

HEADER_GetDefaultBkColor()

Description

Returns the default background color used when creating a HEADER widget.

Prototype

```
GUI_COLOR HEADER_GetDefaultBkColor(void);
```

Return value

Default background color used when creating a HEADER widget.

HEADER_GetDefaultBorderH()

Description

Returns the value used for the horizontal spacing when creating a HEADER widget. Horizontal spacing means the horizontal distance in pixel between text and the horizontal border of the item.

Prototype

```
int HEADER_GetDefaultBorderH(void);
```

Return value

Value used for the horizontal spacing when creating a HEADER widget.

Additional information

Horizontal spacing takes effect only if the given width of a new item is 0.

HEADER_GetDefaultBorderV()

Description

Returns the value used for the vertical spacing when creating a HEADER widget. Vertical spacing means the vertical distance in pixel between text and the vertical border of the HEADER widget.

Prototype

```
int HEADER_GetDefaultBorderV(void);
```

Return value

Value used for the vertical spacing when creating a HEADER widget.

HEADER_GetDefaultCursor()

Description

Returns a pointer to the cursor displayed when dragging the width of an item.

Prototype

```
const GUI_CURSOR * HEADER_GetDefaultCursor(void);
```

Return value

pointer to the cursor displayed when dragging the width of an item.

HEADER_GetDefaultFont()

Description

Returns a pointer to the default font used when creating a HEADER widget.

Prototype

```
const GUI_FONT * HEADER_GetDefaultFont(void);
```

Return value

Pointer to the default font used when creating a HEADER widget.

HEADER_GetDefaultTextColor()

Description

Returns the default text color used when creating a HEADER widget.

Prototype

```
GUI_COLOR HEADER_GetDefaultTextColor(void);
```

Return value

Default text color used when creating a HEADER widget.

HEADER_GetHeight()

Description

Returns the height of the given HEADER widget.

Prototype

```
int HEADER_GetHeight(HEADER_Handle hObj);
```

Parameter	Description
hObj	Handle of HEADER widget

Table 19.344: HEADER_GetHeight() parameter list

Return value

Height of the given HEADER widget.

HEADER_GetItemWidth()

Description

Returns the item width of the given HEADER widget.

Prototype

```
int HEADER_GetItemWidth(HEADER_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of the item

Table 19.345: HEADER_GetItemWidth() parameter list

Return value

Width of the item.

HEADER_GetNumItems()

Description

Returns the number of items of the given HEADER widget.

Prototype

```
int HEADER_GetNumItems(HEADER_Handle hObj);
```

Parameter	Description
hObj	Handle of HEADER widget

Table 19.346: HEADER_GetNumItems() parameter list

Return value

Number of items of the given HEADER widget.

HEADER_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

HEADER_SetBitmap()

Description

Sets the bitmap used when displaying the specified item.

Prototype

```
void HEADER_SetBitmap(HEADER_Handle      hObj,
                      unsigned int       Index,
```



```
                                    const GUI_BITMAP * pBitmap);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of the item
pBitmap	Pointer to a bitmap structure to be displayed

Table 19.347: HEADER_SetBitmap() parameter list

Additional information

One item of a HEADER widget can contain text and a bitmap. (look at `sample` under `HEADER_SetBitmapEx`)

HEADER_SetBitmapEx()

Description

Sets the bitmap used when displaying the specified item.

Prototype

```
void HEADER_SetBitmapEx(HEADER_Handle      hObj,     unsigned int Index,
                        const GUI_BITMAP * pBitmap, int x,  int y);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of the item
pBitmap	Pointer to a bitmap structure to be displayed
x	Additional offset in x
y	Additional offset in y

Table 19.348: HEADER_SetBitmapEx() parameter list

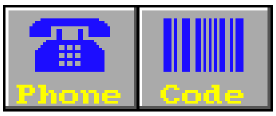
Additional information

One item of a HEADER widget can contain text and a bitmap.

Example

```
...
HEADER_Handle hHeader;
GUI_Init();
HEADER_SetDefaultTextColor(GUI_YELLOW);
HEADER_SetDefaultFont(&GUI_Font8x8);
hHeader = HEADER_Create(10, 10, 100, 40, WM_HBKWIN, 1234, WM_CF_SHOW, 0);
HEADER_AddItem(hHeader, 50, "Phone", GUI_TA_BOTTOM | GUI_TA_HCENTER);
HEADER_AddItem(hHeader, 50, "Code" , GUI_TA_BOTTOM | GUI_TA_HCENTER);
HEADER_SetBitmapEx(hHeader, 0, &bmPhone, 0, -15);
HEADER_SetBitmapEx(hHeader, 1, &bmCode,  0, -15);
...
```

Screenshot of example above:



HEADER_SetBkColor()

Description

Sets the background color of the given HEADER widget.

Prototype

```
void HEADER_SetBkColor(HEADER_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of HEADER widget
Color	Background color to be set

Table 19.349: HEADER_SetBkColor() parameter list

HEADER_SetBMP()

Description

Sets the bitmap used when displaying the specified item.

Prototype

```
void HEADER_SetBMP(HEADER_Handle   hObj,     unsigned int Index,
                   const void    * pBitmap);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of HEADER item
pBitmap	Pointer to bitmap file data

Table 19.350: HEADER_SetBMP() parameter list

Additional information

For additional information regarding bitmap files, refer to chapter "Displaying bitmap files" on page 161.

HEADER_SetBMPEx()

Description

Sets the bitmap used when displaying the specified item.

Prototype

```
void HEADER_SetBMPEx(HEADER_Handle   hObj,     unsigned int Index,
                     const void    * pBitmap,
                     int             x,        int          y);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of the item
pBitmap	Pointer to bitmap file data
x	Additional offset in x
y	Additional offset in y

Table 19.351: HEADER_SetBMPEx() parameter list

Additional information

For additional information regarding bitmap files, refer to chapter "Displaying bitmap files" on page 161.

HEADER_SetDefaultBkColor()

Description

Sets the default background color used when creating a HEADER widget.

Prototype

```
GUI_COLOR HEADER_SetDefaultBkColor(GUI_COLOR Color);
```

Parameter	Description
Color	Background color to be used

Table 19.352: HEADER_SetDefaultBkColor() parameter list

Return value

Previous default background color.

HEADER_SetDefaultBorderH()

Description

Sets the value used for the horizontal spacing when creating a HEADER widget. Horizontal spacing means the horizontal distance in pixel between text and the horizontal border of the item.

Prototype

```
int HEADER_SetDefaultBorderH(int Spacing);
```

Parameter	Description
Spacing	Value to be used

Table 19.353: HEADER_SetDefaultBorderH() parameter list

Return value

Previous default value.

Additional information

Horizontal spacing takes effect only if the given width of a new item is 0.

HEADER_SetDefaultBorderV()

Description

Sets the value used for the vertical spacing when creating a HEADER widget. Vertical spacing means the vertical distance in pixel between text and the vertical border of the HEADER widget.

Prototype

```
int HEADER_SetDefaultBorderV(int Spacing);
```

Parameter	Description
Spacing	Value to be used

Table 19.354: HEADER_SetDefaultBorderV() parameter list

Return value

Previous default value.

HEADER_SetDefaultCursor()

Description

Sets the cursor which will be displayed when dragging the width of an HEADER item.

Prototype

```
const GUI_CURSOR * HEADER_SetDefaultCursor(const GUI_CURSOR * pCursor);
```

Parameter	Description
pCursor	Pointer to the cursor to be shown when dragging the width of an HEADER item

Table 19.355: HEADER_SetDefaultCursor() parameter list

Return value

Pointer to the previous default cursor.

Additional information

There are 2 predefined cursors shown at the beginning of this chapter.

HEADER_SetDefaultFont()

Description

Sets the default font used when creating a HEADER widget.

Prototype

```
const GUI_FONT * HEADER_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to font to be used

Table 19.356: HEADER_SetDefaultFont() parameter list

Return value

Pointer to previous default font.

HEADER_SetDefaultTextColor()

Description

Returns the default text color used when creating a HEADER widget.

Prototype

```
GUI_COLOR HEADER_SetDefaultTextColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color to be used

Table 19.357: HEADER_SetDefaultTextColor() parameter list

Return value

Previous default value.

HEADER_SetDragLimit()

Description

Sets the limit for dragging the dividers on or off. If the limit is on, a divider can only be dragged within the widget area. If the limit is off, it can be dragged outside the widget.

Prototype

```
void HEADER_SetDragLimit(HEADER_Handle hObj, unsigned OnOff);
```

Parameter	Description
hObj	Handle of HEADER widget
OnOff	1 for setting the drag limit on, 0 for off.

Table 19.358: HEADER_SetDragLimit() parameter list

HEADER_SetFont()

Description

Sets the font used when displaying the given HEADER widget

Prototype

```
void HEADER_SetFont(HEADER_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of HEADER widget
pFont	Pointer to font to be used

Table 19.359: HEADER_SetFont() parameter list

HEADER_SetHeight()

Description

Sets the height of the given HEADER widget.

Prototype

```
void HEADER_SetHeight(HEADER_Handle hObj, int Height);
```

Parameter	Description
hObj	Handle of HEADER widget
Height	New height

Table 19.360: HEADER_SetHeight() parameter list

HEADER_SetItemText()

Description

Sets the text used when displaying the specified item.

Prototype

```
void HEADER_SetItemText(HEADER_Handle   hObj, unsigned int Index,
                        const char    * s);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of HEADER item
s	Pointer to string to be displayed

Table 19.361: HEADER_SetItemText() parameter list

Additional information

One HEADER item can contain a string and a bitmap.

HEADER_SetItemWidth()

Description

Sets the width of the specified HEADER item.

Prototype

```
void HEADER_SetItemWidth(HEADER_Handle hObj, unsigned int Index, int Width);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of HEADER item
Width	New width

Table 19.362: HEADER_SetItemWidth() parameter list

HEADER_SetStreamedBitmap()

Description

Sets the bitmap used when displaying the specified item.

Prototype

```
void HEADER_SetStreamedBitmap(HEADER_Handle              hObj,
                              unsigned int               Index,
                              const GUI_BITMAP_STREAM *  pBitmap);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of the item
pBitmap	Pointer to streamed bitmap data to be displayed

Table 19.363: HEADER_SetStreamedBitmap() parameter list

Additional information

For additional information regarding streamed bitmap files, refer to the chapter "2-D Graphic Library" on page 117.

HEADER_SetStreamedBitmapEx()

Description

Sets the bitmap used when displaying the specified item.

Prototype

```
void HEADER_SetStreamedBitmapEx(HEADER_Handle              hObj,
                                unsigned int               Index,
                                const GUI_BITMAP_STREAM *  pBitmap,
                                int                        x,
                                int                        y);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of the item
pBitmap	Pointer to streamed bitmap data to be displayed
x	Additional offset in x
y	Additional offset in y

Table 19.364: HEADER_SetStreamedBitmapEx() parameter list

Additional information

For additional information regarding streamed bitmap files, refer to the chapter "2-D Graphic Library" on page 117.

HEADER_SetTextAlign()

Description

Sets the text alignment of the specified HEADER item.

Prototype

```
void HEADER_SetTextAlign(HEADER_Handle hObj, unsigned int Index, int Align);
```

Parameter	Description
hObj	Handle of HEADER widget
Index	Index of HEADER item
Align	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

Table 19.365: HEADER_SetTextAlign() parameter list

Permitted values for parameter Align (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right (default).
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

HEADER_SetTextColor()

Description

Sets the text color used when displaying the widget.

Prototype

```
void HEADER_SetTextColor(HEADER_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of HEADER widget
Color	Color to be used

Table 19.366: HEADER_SetTextColor() parameter list

HEADER_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

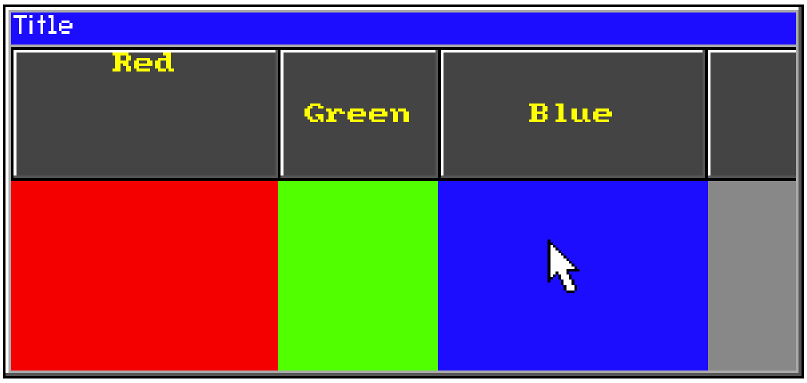
19.11.5 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_Header.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

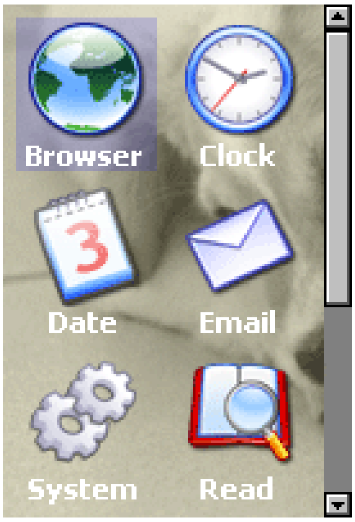
Screenshot of WIDGET_Header.c:



19.12 ICONVIEW: Icon view widget

The ICONVIEW widget can be used for icon based menus often required in hand held devices like mobile telephones or pocket organizers. It shows a list of icons where each icon can be labeled with optional text. Icon view widgets support transparency and alpha blending. So any content can be shown in the background. The currently selected icon can be highlighted by a solid color or with an alpha blending effect, which lets the background shine through. If required a scroll bar can be shown.

ICONVIEW with transparency



ICONVIEW without transparency

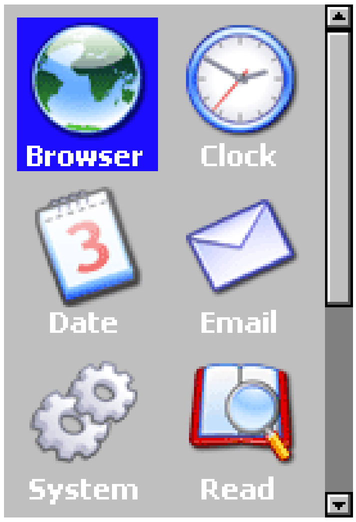


Table 19.367: ICONVIEW with and without transparency

All ICONVIEW-related routines are in the file(s) `ICONVIEW*.c`, `ICONVIEW*.h`. All identifiers are prefixed ICONVIEW.

19.12.1 Configuration options

Type	Macro	Default	Description
N	ICONVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	ICONVIEW_BKCOLOR1_DEFAULT	GUI_BLUE	Background color, selected state.
N	ICONVIEW_TEXTCOLOR0_DEFAULT	GUI_WHITE	Text color, unselected state.
N	ICONVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color, selected state.
S	ICONVIEW_FONT_DEFAULT	GUI_Font13_1	Font to be used for drawing the labels.
N	ICONVIEW_FRAMEX_DEFAULT	5	Free space between the icons and the left and right border of the widget.
N	ICONVIEW_FRAMEY_DEFAULT	5	Free space between the icons and the top and bottom border of the widget.
N	ICONVIEW_SPACEX_DEFAULT	5	Free horizontal space between the icons.
N	ICONVIEW_SPACEY_DEFAULT	5	Free vertical space between the icons.
N	ICONVIEW_ALIGN_DEFAULT	GUI_TA_HCENTER | GUI_TA_BOTTOM	Default alignment to be used for drawing the labels.

Table 19.368: Configuration options

19.12.2 Predefined IDs

The following symbols define IDs which may be used to make ICONVIEW widgets distinguishable from creation: GUI_ID_ICONVIEW0 - GUI_ID_ICONVIEW3

19.12.3 Notification codes

The following events are sent from an ICONVIEW widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget area without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scroll bar has been changed.
WM_NOTIFICATION_SEL_CHANGED	The selection of the widget has been changed.

Table 19.369: Notification codes

19.12.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	Moves the selection to the next icon.
GUI_KEY_LEFT	Moves the selection to the previous icon.
GUI_KEY_DOWN	Moves the selection down.
GUI_KEY_UP	Moves the selection up.
GUI_KEY_HOME	Moves the selection to the first icon.
GUI_KEY_END	Moves the selection to the last icon.

Table 19.370: Keyboard reaction

19.12.5 ICONVIEW API

The table below lists the available emWin ICONVIEW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
ICONVIEW_AddBitmapItem()	Adds a new icon to the ICONVIEW widget.
ICONVIEW_AddStreamedBitmapItem()	Adds a new icon to the ICONVIEW widget using a streamed bitmap.
ICONVIEW_CreateEx()	Creates an ICONVIEW widget.
ICONVIEW_CreateIndirect()	Creates an ICONVIEW widget from a resource table entry.
ICONVIEW_CreateUser()	Creates an ICONVIEW widget using extra bytes as user data.
ICONVIEW_DeleteItem()	Deletes an existing item.
ICONVIEW_EnableStreamAuto()	Enables full support for streamed bitmaps.
ICONVIEW_GetItemText()	Retrieves the text of a specified ICONVIEW item.
ICONVIEW_GetItemUserData()	Retrieves the previously stored user data from a specific item.
ICONVIEW_GetNumItems()	Returns the number of items in the given ICONVIEW widget.
ICONVIEW_GetSel()	Returns the index of the currently selected icon.
ICONVIEW_GetUserData()	Retrieves the data set with ICONVIEW_SetUserData().
ICONVIEW_InsertBitmapItem()	Inserts a new icon to the ICONVIEW widget at the given position.

Table 19.371: ICONVIEW API list

Routine	Description
ICONVIEW_InsertStreamedBitmapItem()	Inserts a new icon to the ICONVIEW widget at the given position using a streamed bitmap.
ICONVIEW_SetBitmapItem()	Sets a bitmap to be used by a specific item.
ICONVIEW_SetBkColor()	Sets the background color.
ICONVIEW_SetFont()	Sets the font to be used for drawing the labels.
ICONVIEW_SetFrame()	Sets the size of the frame between the border of the widget and the icons.
ICONVIEW_SetIconAlign()	Sets the icon alignment.
ICONVIEW_SetItemText()	Sets the text of a specific item.
ICONVIEW_SetItemUserData()	Stores user data in a specific item.
ICONVIEW_SetSel()	Sets the current selection.
ICONVIEW_SetSpace()	Sets the space between icons in x- or y-direction.
ICONVIEW_SetStreamedBitmapItem()	Sets a streamed bitmap to be used by a specific item.
ICONVIEW_SetTextAlign()	Sets the alignment of the text.
ICONVIEW_SetTextColor()	Sets the color to be used to draw the labels.
ICONVIEW_SetUserData()	Sets the extra data of an ICONVIEW widget.
ICONVIEW_SetWrapMode()	Sets the wrapping mode of the given ICONVIEW widget.

Table 19.371: ICONVIEW API list

ICONVIEW_AddBitmapItem()

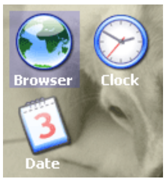
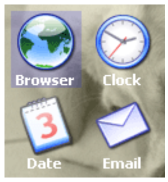
Before	After
	

Table 19.372: CONVIEW_AddBitmapItem() before after screenshots

Description

Adds a new bitmap icon to the widget.

Prototype

```
int ICONVIEW_AddBitmapItem(ICONVIEW_Handle    hObj,
                           const GUI_BITMAP * pBitmap,
                           const char       * pText);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
pBitmap	Pointer to a bitmap structure used to draw the icon.
pText	Text to be used to label the icon.

Table 19.373: CONVIEW_AddBitmapItem() parameter list

Return value

0 on success, !=0 on error.

Additional information

Note that the bitmap pointer needs to remain valid.

ICONVIEW_AddStreamedBitmapItem()

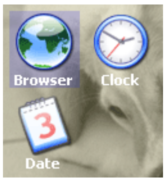
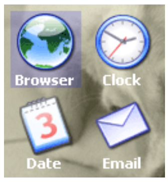
Before	After
	

Table 19.374: ICONVIEW_AddStreamedBitmapItem() before after screenshots

Description

Adds a new streamed bitmap icon to the widget.

Prototype

```
int ICONVIEW_AddStreamedBitmapItem(ICONVIEW_Handle   hObj,
                                   const void      * pStreamedBitmap,
                                   const char      * pText);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
pStreamedBitmap	Pointer to a bitmap stream used to draw the icon.
pText	Text to be used to label the icon.

Table 19.375: ICONVIEW_AddStreamedBitmapItem() parameter list

Return value

0 on success, !=0 on error.

Additional information

The pointer to the bitmap stream needs to remain valid.

ICONVIEW_CreateEx()

Description

Creates an ICONVIEW widget of a specified size at a specified location.

Prototype

```
ICONVIEW_Handle ICONVIEW_CreateEx(int     x0,         int y0,
                                  int     xSize,      int ySize,
                                  WM_HWIN hParent,    int WinFlags,
                                  int     ExFlags,    int Id,
                                  int     xSizeItems, int ySizeItems);
```

Parameter	Description
x0	Leftmost pixel of the widget in parent coordinates.
y0	Topmost pixel of the widget in parent coordinates.
xSize	Horizontal size of the widget in pixels.
ySize	Vertical size of the widget in pixels.
hParent	Handle of parent window. If 0, the new widget will be a child window of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to "WM_CreateWindow()" on page 404 for a list of available parameter values).
ExFlags	See table below.

Table 19.376: ICONVIEW_CreateEx() parameter list

Parameter	Description
Id	Window ID of the widget.
xSizeItem	Horizontal icon size in pixels.
ySizeItem	Vertical icon size in pixels.

Table 19.376: ICONVIEW_CreateEx() parameter list

Permitted values for parameter ExFlags	
0	(default)
ICONVIEW_CF_AUTOSCROLLBAR_V	A vertical scroll bar will be added if the widget area is too small to show all icons.

Return value

Handle of the new widget, 0 if the function fails.

Additional information

If the widget should be transparent, the parameter WinFlags should be or-combined with WM_CF_HASTRANS.

ICONVIEW_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect().
The upper 16 bit of the element Para of the according GUI_WIDGET_CREATE_INFO structure are used according to the parameter xSizeItems of the function ICONVIEW_CreateEx(). The lower 16 bit of the element Para are used according to the parameter ySizeItems of the function ICONVIEW_CreateEx(). The element Flags is used according to the parameter WinFlags of the function ICONVIEW_CreateEx().

ICONVIEW_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function ICONVIEW_CreateEx() can be referred to.

ICONVIEW_DeleteItem()

Description

Deletes an existing item of the ICONVIEW widget.

Prototype

```
void ICONVIEW_DeleteItem(ICONVIEW_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item to be deleted.

Table 19.377: ICONVIEW_DeleteItem() parameter list

ICONVIEW_EnableStreamAuto()

Description

Enables full support for streamed bitmaps.

Prototype

```
void ICONVIEW_EnableStreamAuto(void);
```

Additional information

The ICONVIEW widget supports only index based streamed bitmaps by default. Calling this function enables support for all kinds of streamed bitmaps. This causes all drawing functions for streamed bitmaps to be referenced by the linker.

ICONVIEW_GetItemText()

Description

Retrieves the text of a specified ICONVIEW item.

Prototype

```
int ICONVIEW_GetItemText(ICONVIEW_Handle hObj,    int Index,
                         char *           pBuffer, int MaxSize);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item to be deleted.
pBuffer	Buffer to retrieve the text.
MaxSize	Maximum length of text to copy to the buffer.

Table 19.378: ICONVIEW_GetItemText() parameter list

Return value

The length of the actually copied text is returned.

ICONVIEW_GetItemUserData()

Description

Retrieves the previously stored user data from a specific item.

Prototype

```
U32 ICONVIEW_GetItemUserData(ICONVIEW_Handle hObj, int Index);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item.

Table 19.379: ICONVIEW_GetItemUserData() parameter list

Return value

User data stored in the item as U32.

ICONVIEW_GetNumItems()

Description

Returns the number of items in the given ICONVIEW widget.

Prototype

```
int ICONVIEW_GetNumItems(ICONVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.

Table 19.380: ICONVIEW_GetNumItems() parameter list

Return value

Number of items.

ICONVIEW_GetSel()

Description

Returns the zero based index of the currently selected icon.

Prototype

```
int ICONVIEW_GetSel(ICONVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.

Table 19.381: ICONVIEW_GetSel() parameter list

Return value

Zero based index of the currently selected icon.

ICONVIEW_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

ICONVIEW_InsertBitmapItem()

Description

Inserts a new bitmap icon to the widget. See “ICONVIEW_AddBitmapItem()” on page 604 for screenshots.

Prototype

```
int ICONVIEW_InsertBitmapItem(ICONVIEW_Handle    hObj,
                              const GUI_BITMAP * pBitmap,
                              const char       * pText
                              int                Index);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
pBitmap	Pointer to a bitmap structure used to draw the icon.
pText	Text to be used to label the icon.
Index	Index position to insert the item at.

Table 19.382: ICONVIEW_InsertBitmapItem() parameter list

Return value

0 on success, !=0 on error.

Additional information

Note that the bitmap pointer needs to remain valid.

ICONVIEW_InsertStreamedBitmapItem()

Description

Inserts a new streamed bitmap icon to the widget. See “ICONVIEW_AddBitmapItem()” on page 604 for screenshots.

Prototype

```
int ICONVIEW_InsertStreamedBitmapItem(ICONVIEW_Handle   hObj,
                                      const void      * pStreamedBitmap,
                                      const char      * pText,
                                      int               Index);
```


Parameter	Description
hObj	Handle of ICONVIEW widget.
pStreamedBitmap	Pointer to a bitmap stream used to draw the icon.
pText	Text to be used to label the icon.
Index	Index position to insert the item at.

Table 19.383: ICONVIEW_InsertStreamedBitmapItem() parameter list

Return value

0 on success, !=0 on error.

Additional information

The pointer to the bitmap stream needs to remain valid.

ICONVIEW_SetBitmapItem()

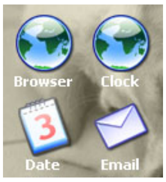
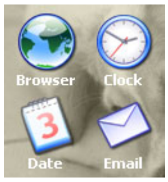
Before	After
	

Table 19.384: ICONVIEW_SetBitmapItem() before after screenshots

Description

Sets a bitmap to be used by a specific item.

Prototype

```
void ICONVIEW_SetBitmapItem(ICONVIEW_Handle    hObj,
                            int                Index,
                            const GUI_BITMAP * pBitmap);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item.
pBitmap	Pointer to the bitmap to be used.

Table 19.385: ICONVIEW_SetBitmapItem() parameter list

Additional information

The pointer to the bitmap structure needs to remain valid.

ICONVIEW_SetBkColor()

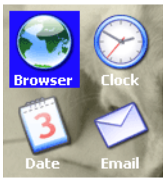
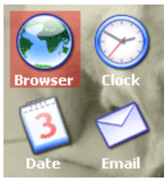
Before	After
	

Table 19.386: ICONVIEW_SetBkColor() before after screenshots

Description

Sets the background color of the widget.

Prototype

```
void ICONVIEW_SetBkColor(ICONVIEW_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	See table below.
Color	Color to be used for drawing the background.

Table 19.387: ICONVIEW_SetBkColor() parameter list

Permitted values for parameter Index	
ICONVIEW_CI_BK	Color used to draw the widget background.
ICONVIEW_CI_SEL	Color used to highlight the currently selected item.

Additional information

The upper 8 bits of the 32 bit color value can be used for an alpha blending effect. For more details about alpha blending, refer to “GUI_SetAlpha()” on page 131.

ICONVIEW_SetFont()

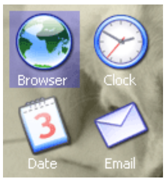
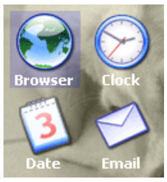
Before	After
	

Table 19.388: ICONVIEW_SetFont() before after screenshots

Description

Sets the font to be used for drawing the icon labels.

Prototype

```
void ICONVIEW_SetFont(ICONVIEW_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
pFont	Pointer to a GUI_FONT structure to be used to draw the icon labels.

Table 19.389: ICONVIEW_SetFont() parameter list

ICONVIEW_SetFrame()

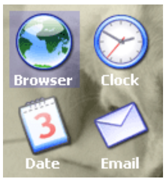
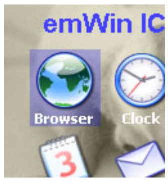
Before	After
	

Table 19.390: ICONVIEW_SetFrame() before after screenshots

Description

Sets the size of the frame between the border of the widget and the icons.

Prototype

```
void ICONVIEW_SetFrame(ICONVIEW_Handle hObj,
                       int             Coord,
                       int             Value);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Coord	See permitted values for this parameter below.
Value	Distance to be set.

Table 19.391: ICONVIEW_SetFrame() parameter list

Permitted values for parameter Coord	
GUI_COORD_X	X-direction.
GUI_COORD_Y	Y-direction.

ICONVIEW_SetIconAlign()

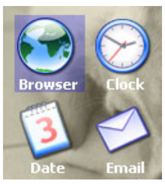
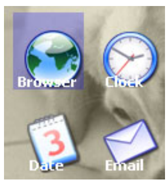
Before	After
	

Table 19.392: ICONVIEW_SetIconAlign() before after screenshots

Description

Sets the icon alignment.

Prototype

```
void ICONVIEW_SetIconAlign(ICONVIEW_Handle hObj, int IconAlign);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
IconAlign	Alignment of the icons. See table below.

Table 19.393: ICONVIEW_SetIconAlign() parameter list

Permitted values for parameter IconAlign (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
ICONVIEW_IA_LEFT	Align X-position left.
ICONVIEW_IA_HCENTER	Center X-position. (default)
ICONVIEW_IA_RIGHT	Align X-position right.
Vertical alignment	
ICONVIEW_IA_TOP	Align Y-position with top of characters.
ICONVIEW_IA_VCENTER	Center Y-position. (default)
ICONVIEW_IA_BOTTOM	Align Y-position with bottom pixel line of font.

ICONVIEW_SetItemText()

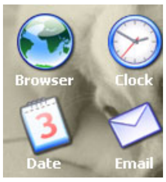
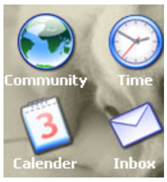
Before	After
	

Table 19.394: ICONVIEW_SetItemText() before after screenshots

Description

Sets the text of a specific item.

Prototype

```
void ICONVIEW_SetItemText(ICONVIEW_Handle   hObj,
                          int               Index,
                          const char      * pText);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item.
pText	Pointer to the text to be used.

Table 19.395: ICONVIEW_SetItemText() parameter list

ICONVIEW_SetItemUserData()

Description

Stores user data in a specific item.

Prototype

```
void ICONVIEW_SetItemUserData(ICONVIEW_Handle hObj,
                              int             Index,
                              U32             UserData);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item.
UserData	32 bit user data to be stored.

Table 19.396: ICONVIEW_SetItemUserData() parameter list

ICONVIEW_SetSel()

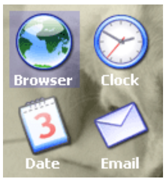
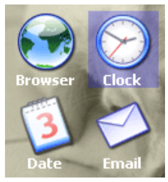
Before	After
	

Table 19.397: ICONVIEW_SetSel() before after screenshots

Description

Sets the current selection.

Prototype

```
void ICONVIEW_SetSel(ICONVIEW_Handle hObj, int Sel);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Sel	New selection.

Table 19.398: ICONVIEW_SetSel() parameter list

ICONVIEW_SetSpace()

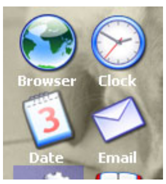
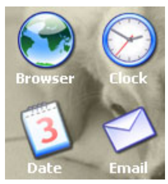
Before	After
	

Table 19.399: ICONVIEW_SetSpace() before after screenshots

Description

Sets the space between icons in x- or y-direction.

Prototype

```
void ICONVIEW_SetSpace(ICONVIEW_Handle hObj, int Coord, int Value);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Coord	See permitted values for this parameter below.
Value	Distance to be set.

Table 19.400: ICONVIEW_SetSpace() parameter list

Permitted values for parameter Coord	
GUI_COORD_X	X-direction.
GUI_COORD_Y	Y-direction.

ICONVIEW_SetStreamedBitmapItem()

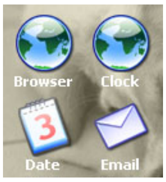
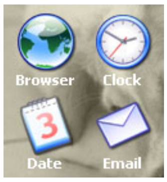
Before	After
	

Table 19.401: ICONVIEW_SetStreamedBitmapItem() before after screenshots

Description

Sets a streamed bitmap to be used by a specific item.

Prototype

```
void ICONVIEW_SetStreamedBitmapItem(ICONVIEW_Handle   hObj,
                                    int               Index,
                                    const void      * pStreamedBitmap);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	Index of the item.
pStreamedBitmap	Pointer to the bitmap stream to be used.

Table 19.402: ICONVIEW_SetStreamedBitmapItem() parameter list

Additional information

The pointer to the bitmap stream needs to remain valid.

ICONVIEW_SetTextAlign()

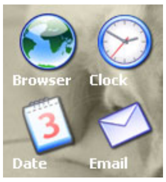
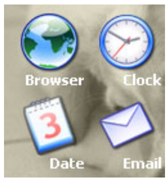
Before	After
	

Table 19.403: ICONVIEW_SetTextAlign() before after screenshots

Description

Sets the color to be used to draw the labels.

Prototype

```
void ICONVIEW_SetTextAlign(ICONVIEW_Handle hObj, int TextAlign);
```

Parameter	Description
`hObj`	Handle of ICONVIEW widget.
`TextAlign`	See table below.

Table 19.404: ICONVIEW_SetTextAlign() parameter list

Permitted values for parameter `TextAlign` (horizontal and vertical flags are `OR`-combinable)	
Horizontal alignment	
`GUI_TA_LEFT`	Align X-position left (default).
`GUI_TA_HCENTER`	Center X-position.
`GUI_TA_RIGHT`	Align X-position right (default).
Vertical alignment	
`GUI_TA_TOP`	Align Y-position with top of characters (default).
`GUI_TA_VCENTER`	Center Y-position.
`GUI_TA_BOTTOM`	Align Y-position with bottom pixel line of font.

ICONVIEW_SetTextColor()

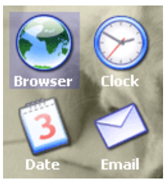
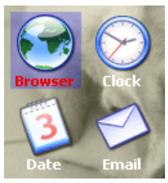
Before	After
	

Table 19.405: ICONVIEW_SetTextColor() before after screenshots

Description

Sets the color to be used to draw the labels.

Prototype

```
void ICONVIEW_SetTextColor(ICONVIEW_Handle hObj,   int Index,
```



```
GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
Index	See table below.
Color	Color to be used

Table 19.406: ICONVIEW_SetTextColor() parameter list

Permitted values for parameter Index	
ICONVIEW_CI_UNSEL	Color used to draw the labels in unselected state.
ICONVIEW_CI_SEL	Color used to draw the labels in selected state.

ICONVIEW_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

ICONVIEW_SetWrapMode()

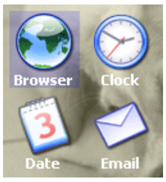
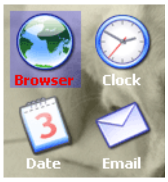
Before	After
	

Table 19.407: ICONVIEW_SetWrapMode() before after screenshots

Description

Sets the wrapping mode to be used for the given ICONVIEW widget.

Prototype

```
void ICONVIEW_SetTextColor(ICONVIEW_Handle hObj, GUI_WRAPMODE WrapMode);
```

Parameter	Description
hObj	Handle of ICONVIEW widget.
WrapMode	See table below.

Table 19.408: ICONVIEW_SetWrapMode() parameter list

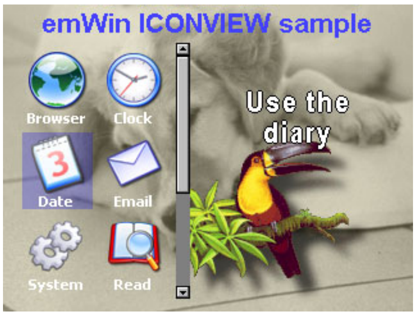
Permitted values for parameter WrapMode	
GUI_WRAPMODE_NONE	No wrapping will be performed.
GUI_WRAPMODE_WORD	Text is wrapped word wise.
GUI_WRAPMODE_CHAR	Text is wrapped char wise.

19.12.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

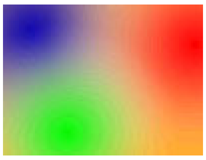
- WIDGET_IconView

Screenshot of WIDGET_Iconview.c:



19.13 IMAGE: Image widget

IMAGE widgets are used to display images of different formats from internal as well as from external memory.



All IMAGE-related routines are located in the file(s) IMAGE*.c, IMAGE.h. All identifiers are prefixed IMAGE.

19.13.1 Configuration options

The IMAGE widget can be configured using an or-combination of the following symbols as 'ExFlags'-parameter at creation. See IMAGE_CreateEx() below.

Configuration flag	Default	Description
IMAGE_CF_MEMDEV	not set	Makes the IMAGE widget use an internal Memory Device for drawing. Contrary to the Memory Device which is created by the Window Manager's automatic use of Memory Devices (WM_CF_MEMDEV), this device stays valid all the time. It has to be ensured that the emWin memory pool which is defined by the function GUI_ALLOC_AssignMemory (in GUIConf.c), is big enough to store the complete data. If the Memory Device can not be created, the image is drawn directly. This might possibly mean loss of performance.
IMAGE_CF_TILE	not set	Use tiling to fill the whole widget area.
IMAGE_CF_ALPHA	not set	Support PNG images using alpha blending.
IMAGE_CF_ATTACHED	not set	Fix the widget size to the borders of the parent window.
IMAGE_CF_AUTOSIZE	not set	Set the widget size to the size of the image.

Table 19.409: Configuration options

19.13.2 Predefined IDs

The following symbols define IDs which may be used to make IMAGE widgets distinguishable from creation: GUI_ID_IMAGE0 - GUI_ID_IMAGE9

19.13.3 Notification codes

The following events are sent from an IMAGE widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	The widget has been clicked.
WM_NOTIFICATION_RELEASED	The widget has been released.
WM_NOTIFICATION_MOVED_OUT	The pointer was moved out of the widget area while the PID was in pressed state.

Table 19.410: Notification codes

19.13.4 IMAGE API

The table below lists the available IMAGE-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
IMAGE_CreateEx()	Creates an IMAGE widget.
IMAGE_CreateIndirect()	Creates a IMAGE widget from a resource table entry.
IMAGE_CreateUser()	Creates a IMAGE widget using extra bytes as user data.

Table 19.411: IMAGE API list

Routine	Description
IMAGE_SetBitmap()	Sets a bitmap to be displayed.
IMAGE_SetBMP()	Sets a BMP file to be displayed.
IMAGE_SetBMPEx()	Sets a BMP file to be displayed from external memory.
IMAGE_SetDTA()	Sets a DTA file to be displayed.
IMAGE_SetDTAEx()	Sets a DTA file to be displayed from external memory.
IMAGE_SetGIF()	Sets a GIF file to be displayed.
IMAGE_SetGIFEx()	Sets a GIF file to be displayed from external memory.
IMAGE_SetJPEG()	Sets a JPEG file to be displayed.
IMAGE_SetJPEGEx()	Sets a JPEG file to be displayed from external memory.
IMAGE_SetPNG()	Sets a PNG file to be displayed.
IMAGE_SetPNGEx()	Sets a PNG file to be displayed from external memory.

Table 19.411: IMAGE API list

IMAGE_CreateEx()

Description

Creates an IMAGE widget of a specified size at a specified location.

Prototype

```
IMAGE_Handle IMAGE_CreateEx(int      x0,       int y0,
                            int      xSize,    int ySize,
                            WM_HWIN  hParent,  int WinFlags,
                            int      ExFlags,  int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the IMAGE widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	See permitted values below.
Id	Window ID of the widget.

Table 19.412: IMAGE_CreateEx() parameter list

Permitted values for parameter ExFlags	
IMAGE_CF_MEMDEV	Widget uses an internal memory device which speeds up use of compressed images (GIF, JPEG, PNG).
IMAGE_CF_TILE	Uses tiling to fill up the whole area of the widget.
IMAGE_CF_ALPHA	Needs to be set if alpha blending is required (PNG).
IMAGE_CF_ATTACHED	Widget size is fixed to the parent border.
IMAGE_CF_AUTOSIZE	Widget size is taken from the attached image.

Return value

Handle of the created IMAGE widget; 0 if the function fails.

Additional information

If the possibility of storing user data is a matter the function IMAGE_CreateUser() should be used instead.

IMAGE_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. For details the function `<WIDGET>_CreateIndirect()` should be referred to. The element `Flags` is used according to the parameter `WinFlags` of the function `IMAGE_CreateEx()`. The element `Para` is used according to the parameter `ExFlags` of the function `IMAGE_CreateEx()`. For a detailed description of the parameters the function `IMAGE_CreateEx()` can be referred to.

IMAGE_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `IMAGE_CreateEx()` can be referred to.

IMAGE_SetBitmap()

Description

Sets a bitmap to be displayed.

Prototype

```
void IMAGE_SetBitmap(IMAGE_Handle hWin, const GUI_BITMAP * pBitmap);
```

Parameter	Description
hWin	Handle of IMAGE widget.
pBitmap	Pointer to the bitmap.

Table 19.413: IMAGE_SetBitmap() parameter list

IMAGE_SetBMP()

IMAGE_SetDTA()

IMAGE_SetGIF()

IMAGE_SetJPEG()

IMAGE_SetPNG()

Description

These functions set a file of one of the formats listed below to be displayed:

- BMP
- DTA
- GIF
- JPEG
- PNG

Prototype

```
void IMAGE_Set<FORMAT>(IMAGE_Handle hObj, const void * pData, U32 FileSize);
```

Parameter	Description
hObj	Handle of IMAGE widget.
pData	Pointer to the IMAGE data.
FileSize	Size of the IMAGE data.

Table 19.414: IMAGE_Set<FORMAT>() parameter list

Additional information

The PNG functionality requires the PNG library which can be downloaded from www.segger.com/link/emwin_png.zip. Animated GIF files are displayed automatically.

IMAGE_SetBMPEx()

IMAGE_SetDTAEx()

IMAGE_SetGIFEx()

IMAGE_SetJPEGEx()

IMAGE_SetPNGEx()

Description

These functions set a file of one of the formats listed below to be displayed from external memory:

- BMP
- DTA
- GIF
- JPEG
- PNG

Prototype

```
void IMAGE_Set<FORMAT>Ex(IMAGE_Handle hObj, GUI_GET_DATA_FUNC * pfGetData,
                                           void        * pVoid);
```

Parameter	Description
hObj	Handle of IMAGE widget.
pfGetData	Pointer to the GetData()-function. Details on how to implement a GetData()-function can be found under "Getting data with the ...Ex() functions" on page 187.
pVoid	Pointer to the IMAGE data. It is passed to the GetData()-function as first parameter.

Table 19.415: IMAGE_Set...Ex() parameter list

Additional information

The PNG functionality requires the PNG library which can be downloaded from www.segger.com/link/emwin_png.zip. Animated GIF files are displayed automatically.

19.14 KNOB: Knob widget

The KNOB widget is used to change a value with a knob. The KNOB widget is transparent per default and it requires a memory device containing a drawn knob to be visible. If the memory device contains transparency, any content can be shown in the background.

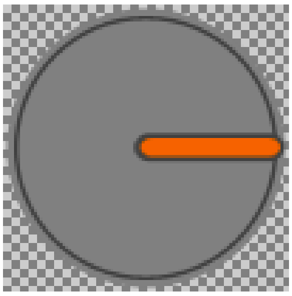
Drawn knob with transparency	How it appears used as KNOB widget
	

Table 19.416: Drawn knob in a memory device and used as widget

All KNOB-related routines are in the file(s) KNOB.c and KNOB.h. All identifiers are prefixed with KNOB.

Ticks and Ticksize

A Tick describes the smallest range of movement of the KNOB widget. The smallest size of one Tick (Ticksize 1) equates 1/10 of a degree. So it takes 3600 Ticks to fulfill one round. The size of one Tick can be set with the function `KNOB_SetTickSize()`.

Examples	
Ticksize	Ticks for one round
60	60
36	100
300	12

Table 19.417: Ticksize description

19.14.1 Requirements

This widget requires the optional memory devices. Without the optional memory devices package the widget can’t be used! The memory device which contains the drawn knob will not be deleted if the KNOB widget gets deleted. Make sure to delete the memory device by yourself if it is not any longer required.

Memory requirements

At least the widget uses two memory devices with a color depth of 32bpp (one with the drawn knob, one for internal use). The required amount of memory depends on using an additional memory device for the background or not. Detailed information on how to calculate memory requirements can be found in the chapter “Memory Devices” on page 319.

App. memory usage without a background device:

```
XSIZE * 4 * YSIZE * 2
```

19.14.2 Configuration options

Type	Macro	Default	Description
N	KNOB_BKCOLOR_DEFAULT	GUI_TRANSPARENT	Background color
N	KNOB_KEYVALUE_DEFAULT	1	Range the KNOB will rotate on one key press in 1/10 of degree.
N	KNOB_OFFSET_DEFAULT	0	An offset added to initial point of the KNOB in 1/10 of degree.
N	KNOB_PERIOD_DEFAULT	1500	Time the KNOB takes to stop in ms.
N	KNOB_SNAP_DEFAULT	0	Interval where the KNOB automatically stops in 1/10 of degree.
N	KNOB_TICKSIZE_DEFAULT	1	Size of one tick in 1/10 of degree.

Table 19.418: Configuration options

19.14.3 Predefined IDs

The following symbols define IDs which may be used to make KNOB widgets distinguishable from creation: GUI_ID_KNOB0 - GUI_ID_KNOB9.

19.14.4 Notification codes

The following events are sent from a KNOB widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	KNOB has been clicked.
WM_NOTIFICATION_RELEASED	KNOB has been released.
WM_NOTIFICATION_MOVED_OUT	KNOB has been clicked and pointer has been moved out of the KNOB area without releasing.
WM_NOTIFICATION_VALUE_CHANGED	The value of the KNOB widget has been changed.

Table 19.419: Notification codes

19.14.5 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	The KNOB rotates CW.
GUI_KEY_LEFT	The KNOB rotates CCW.
GUI_KEY_DOWN	The KNOB rotates CW.
GUI_KEY_UP	The KNOB rotates CCW.

Table 19.420: Keyboard reaction

19.14.6 KNOB API

The table below lists the available emWin KNOB-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
KNOB_AddValue()	Adds a value to the position of the KNOB around its rotary axis.
KNOB_CreateEx()	Creates a KNOB widget.
KNOB_CreateIndirect()	Creates a KNOB widget from a resource table entry.
KNOB_CreateUser()	Creates a KNOB widget using extra bytes as user data.
KNOB_GetUserData()	Retrieves the data set with KNOB_SetUserData().
KNOB_GetValue()	Returns the current value of the KNOB widget.
KNOB_SetBkColor()	Sets the background color of the KNOB widget.

Table 19.421: KNOB API list

Routine	Description
KNOB_SetBkDevice()	Sets a memory device with a drawn background as background.
KNOB_SetDevice()	Sets a memory device with a drawing used as knob.
KNOB_SetKeyValue()	Sets a value how far the knob rotates on one key press.
KNOB_SetOffset()	Sets an offset which is added to the initial position of the knob.
KNOB_SetPeriod()	Sets a time period which the knob needs to stop.
KNOB_SetPos()	Sets a new position for the knob around its rotary axis.
KNOB_SetRange()	Sets a minimum and maximum range between the knob is able to rotate.
KNOB_SetSnap()	Sets a range between snap positions where the knob stops automatically.
KNOB_SetTickSize()	Sets the size of one tick.
KNOB_SetUserData()	Sets the extra data of a KNOB widget.

Table 19.421: KNOB API list

KNOB_AddValue()

Description

Adds a value to the position of the KNOB widget around its rotary axis.

Prototype

```
void KNOB_AddValue(KNOB_Handle hObj, I32 Value);
```

Parameter	Description
hObj	Handle of the KNOB widget.
Value	Value to be added to the position around the rotary axis in ticks.

Table 19.422: KNOB_AddValue() parameter list

Additional information

A positive `Value` rotates the knob counter-clockwise and a negative `Value` clockwise.

KNOB_CreateEx()

Description

Creates a KNOB widget of a specified size at a specified location with the possibility to assign a parent window.

Prototype

```
KNOB_Handle KNOB_CreateEx(int x0, int y0, int xSize, int ySize,
                          WM_HWIN hParent, int Id, int Flags);
```

Parameter	Description
x0	Leftmost pixel of the KNOB widget (in parent coordinates).
y0	Topmost pixel of the KNOB widget (in parent coordinates).
xSize	Horizontal size of the KNOB widget (in pixels).
ySize	Vertical size of the KNOB widget (in pixels).
hParent	Parent window of the KNOB widget.
Id	ID of the KNOB widget.
Flags	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).

Table 19.423: KNOB_CreateEx() parameter list

Return value

Handle of the created KNOB widget.

Additional information

The created widget is not visible by default. In order to have the widget being drawn a Memory Device needs to be set first. Using the function `KNOB_SetDevice()` a Memory Device containing the actual Knob can be set.

KNOB_CreateIndirect()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateIndirect()`. The elements `Flags` and `Para` of the resource passed as parameter are not used.

KNOB_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `KNOB_CreateEx()` can be referred to.

KNOB_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

KNOB_GetValue()

Description

Returns the value of the current position around its rotary axis. The value describes the angle in relation to the starting point. The starting point is defined by the offset which is set using the function `KNOB_SetOffset()`. Depending on the configured range (`KNOB_SetRange()`) the returned value might be more than 3600.

Prototype

```
I32 KNOB_GetValue(KNOB_Handle hObj);
```

Parameter	Description
`hObj`	Handle of KNOB widget.

Table 19.424: KNOB_GetValue() parameter list

Return value

Value of the current position of the KNOB around its rotary axis in 1/10 of a degree.

KNOB_SetBkColor()

Description

Sets a color for the background of the KNOB widget.

Prototype

```
void KNOB_SetBkColor(KNOB_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
`hObj`	Handle of KNOB widget.
`Color`	Color to be set as background color.

Table 19.425: KNOB_SetBkColor() parameter list

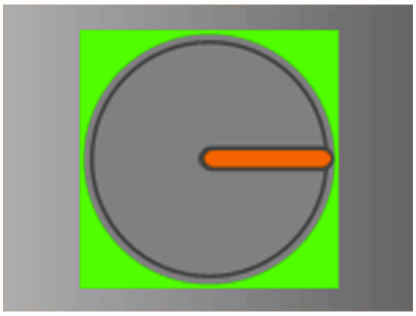
Without background color	With background color
	

Table 19.426: KNOB_SetBkColor() before after screenshots

KNOB_SetBkDevice()

Description

Sets a memory device to be shown as background.

Prototype

```
void KNOB_SetBkDevice(KNOB_Handle hObj, GUI_MEMDEV_Handle hMemBk);
```

Parameter	Description
hObj	Handle of KNOB widget.
hMemBk	Handle of a memory device which should be used as background.

Table 19.427: KNOB_SetBkDevice() parameter list

Additional information

The widget will not delete the background device if the widget gets deleted. So make sure to delete the background device by yourself.

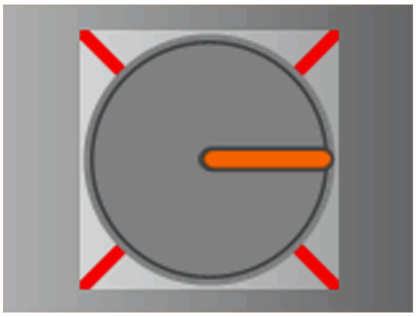
Without background color	With background device
	

Table 19.428: KNOB_SetBkDevice() before after screenshots

KNOB_SetDevice()

Description

Sets a memory device which contains a drawn knob. This drawing defines the appearance of the knob.

Prototype

```
void KNOB_SetDevice(KNOB_Handle hObj, GUI_MEMDEV_Handle hMemSrc);
```

Parameter	Description
hObj	Handle of KNOB widget.
hMemSrc	Handle of a memory device which is used for the appearance of the KNOB.

Table 19.429: KNOB_SetDevice() parameter list

Additional information

The memory device must have a color depth of 32bpp. Without setting a memory device the KNOB will be invisible.

KNOB_SetKeyValue()

Description

Sets a value which defines the movement range of the KNOB on one key press.

Prototype

```
void KNOB_SetKeyValue(KNOB_Handle hObj, I32 KeyValue);
```

Parameter	Description
hObj	Handle of KNOB widget.
KeyValue	Value is used for the movement range on one key press. Unit is 1/10 of a degree.

Table 19.430: KNOB_SetKeyValue() parameter list

Additional information

If a Ticksize larger than 1 is set the KNOB widget uses the Ticksize as `KeyValue`. This function is used only if no Ticksize is set.

KNOB_SetOffset()

Description

Sets an offset angle for the KNOB widget. The knob will appear rotated from the beginning of the application.

Prototype

```
void KNOB_SetOffset(KNOB_Handle hObj, I32 Offset);
```

Parameter	Description
hObj	Handle of KNOB widget.
Offset	Offset position of the knob around its rotary axis in 1/10 of a degree.

Table 19.431: KNOB_SetOffset() parameter list

KNOB_SetPeriod()

Description

Sets a time period which defines how long the KNOB will need to stop.

Prototype

```
void KNOB_SetPeriod(KNOB_Handle hObj, I32 Period);
```

Parameter	Description
hObj	Handle of KNOB widget.
Period	Time it takes to stop the KNOB in ms.

Table 19.432: KNOB_SetPeriod() parameter list

Additional information

The period can't be larger than 46340ms. Default is 1500ms.

KNOB_SetPos()

Description

Sets a new position for the KNOB widget around its rotary axis.

Prototype

```
void KNOB_SetPos(KNOB_Handle hObj, I32 Pos);
```

Parameter	Description
hObj	Handle of KNOB widget.
Pos	New position around the rotary axis in ticks.

Table 19.433: KNOB_SetPos() parameter list

KNOB_SetRange()

Description

Sets a range within the KNOB can be rotated.

Prototype

```
void KNOB_SetRange(KNOB_Handle hObj, I32 MinRange, I32 MaxRange);
```

Parameter	Description
hObj	Handle of KNOB widget.
MinRange	The minimum of the movement range in ticks. This value defines the distance from the starting point to the minimum value which can be reached by rotating the KNOB widget clockwise. MinRange defines the range to move clockwise. MaxRange defines the range ot move counterclockwise.
MaxRange	The maximum of the movement range in ticks. This value defines the distance from the starting point to the maximum value which can be reached by rotating the KNOB widget counterclockwise.

Table 19.434: KNOB_SetRange() parameter list

Additional information

The difference between `MinRange` and `MaxRange` defines the movement area. If the `TickSize` is set to 10 and `MinRange` is set to 0 and MaxRange to 720 the KNOB is able to rotate twice around its rotary axis. If `MinRange` and `MaxRange` are equal the KNOB will be able to rotate freely. Since 0 is the starting point, it is recommended to include 0 in the range.

KNOB_SetSnap()

Description

Sets a range between snap positions where the KNOB automatically stops. After the KNOB starts rotating it calculates the closest snap position to its stopping point and stops at this snap position. If the KNOB is released between to snap positions it rotates to the closest.

Prototype

```
void KNOB_SetSnap(KNOB_Handle hObj, I32 Snap);
```

Parameter	Description
hObj	Handle of KNOB widget.
Snap	Sets the range between snap positions in ticks.

Table 19.435: KNOB_SetSnap() parameter list

Additional information

If `TickSize` is set to 1 (default) and `Snap` is 300 every 30 degrees will be a snap position.

KNOB_SetTickSize()

Description

Sets the Ticksize of the KNOB. The Ticksize defines the minimum movement of the knob in 1/10 of degrees.

Prototype

```
void KNOB_SetTickSize(KNOB_Handle hObj, I32 TickSize);
```

Parameter	Description
hObj	Handle of KNOB widget.
TickSize	Sets the Ticksize of the KNOB widget.

Table 19.436: KNOB_SetTickSize() parameter list

Additional information

The default `TickSize` is 1, it takes 3600 Ticks to fulfill one round. For example if TickSize is set to 10 the minimum movement is 1 degree and it takes 360 Ticks to fulfill one round. This function should be called before any other KNOB function.

KNOB_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

19.15 LISTBOX: List box widget

LISTBOX widgets are used to select one element of a list. A list box can be created without a surrounding frame window, as shown below, or as a child window of a FRAMEWIN widget (see the additional screenshots at the end of the section). As items in a list box are selected, they appear highlighted. Note that the background color of a selected item depends on whether the list box window has input focus.

List box with focus	List box without focus
	

Table 19.437: LISTBOX focus

All LISTBOX-related routines are in the file(s) `LISTBOX*.c`, `LISTBOX.h`. All identifiers are prefixed LISTBOX.

19.15.1 Configuration options

Type	Macro	Default	Description
N	`LISTBOX_BKCOLOR0_DEFAULT`	GUI_WHITE	Background color, unselected state.
N	`LISTBOX_BKCOLOR1_DEFAULT`	GUI_GRAY	Background color, selected state without focus.
N	`LISTBOX_BKCOLOR2_DEFAULT`	GUI_BLUE	Background color, selected state with focus.
S	`LISTBOX_FONT_DEFAULT`	&GUI_Font13_1	Font used.
N	`LISTBOX_TEXTCOLOR0_DEFAULT`	GUI_BLACK	Text color, unselected state.
N	`LISTBOX_TEXTCOLOR1_DEFAULT`	GUI_WHITE	Text color, selected state without focus.
N	`LISTBOX_TEXTCOLOR2_DEFAULT`	GUI_WHITE	Text color, selected state with focus.

Table 19.438: Configuration options

19.15.2 Predefined IDs

The following symbols define IDs which may be used to make LISTBOX widgets distinguishable from creation: `GUI_ID_LISTBOX0` - `GUI_ID_LISTBOX9`.

19.15.3 Notification codes

The following events are sent from a LISTBOX widget to its parent window as part of a `WM_NOTIFY_PARENT` message:

Message	Description
`WM_NOTIFICATION_CLICKED`	LISTBOX has been clicked.
`WM_NOTIFICATION_RELEASED`	LISTBOX has been released.
`WM_NOTIFICATION_MOVED_OUT`	LISTBOX has been clicked and pointer has been moved out of the box without releasing.
`WM_NOTIFICATION_SCROLL_CHANGED`	The scroll position of the optional scroll bar has been changed.
`WM_NOTIFICATION_SEL_CHANGED`	The selection of the LISTBOX widget has changed.

Table 19.439: Notification codes

19.15.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_SPACE	If the widget works in multi selection mode this key toggles the state of the current selected item.
GUI_KEY_RIGHT	If the maximum X-size of the list box items is larger than the list box itself this key scrolls the list box content to the left.
GUI_KEY_LEFT	If the maximum X-size of the list box items is larger than the list box itself this key scrolls the list box content to the right.
GUI_KEY_DOWN	Moves the selection bar down.
GUI_KEY_UP	Moves the selection bar up.

Table 19.440: Keyboard reaction

19.15.5 LISTBOX API

The table below lists the available emWin LISTBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
LISTBOX_AddString()	Adds an item to a LISTBOX widget.
LISTBOX_Create()	Creates a LISTBOX widget. (Obsolete)
LISTBOX_CreateAsChild()	Creates a LISTBOX widget as a child window. (Obsolete)
LISTBOX_CreateEx()	Creates a LISTBOX widget.
LISTBOX_CreateIndirect()	Creates a LISTBOX widget from resource table entry.
LISTBOX_CreateUser()	Creates a LISTBOX widget using extra bytes as user data.
LISTBOX_DecSel()	Decrements selection.
LISTBOX_DeleteItem()	Deletes an element.
LISTBOX_EnableWrapMode()	Enables wrap mode for scrolling immediately from the end to the beginning and vice versa.
LISTBOX_GetDefaultBkColor()	Returns the default background color for LISTBOX widgets.
LISTBOX_GetDefaultFont()	Returns the default font for LISTBOX widgets.
LISTBOX_GetDefaultScrollStepH()	Returns the default number of pixels to be scrolled horizontal.
LISTBOX_GetDefaultTextAlign()	Returns the default text alignment for new LISTBOX widgets.
LISTBOX_GetDefaultTextColor()	Returns the default text color for new LISTBOX widgets.
LISTBOX_GetFont()	Returns the font of the LISTBOX widget.
LISTBOX_GetItemDisabled()	Returns the disabled state of the given item.
LISTBOX_GetItemSel()	Returns the selection state of a LISTBOX entry.
LISTBOX_GetItemText()	Returns the text of a LISTBOX widget entry.
LISTBOX_GetMulti()	Returns if the multi select mode is active.
LISTBOX_GetNumItems()	Returns the number of items in a LISTBOX widget.
LISTBOX_GetScrollStepH()	Returns the number of pixels to be scrolled horizontal.
LISTBOX_GetSel()	Returns the number of the selected item.
LISTBOX_GetTextAlign()	Returns the text alignment of the LISTBOX.
LISTBOX_GetUserData()	Retrieves the data set with LISTBOX_SetUserData().
LISTBOX_IncSel()	Increments selection.
LISTBOX_InsertString()	Inserts an element.
LISTBOX_InvalidateItem()	Invalidates an item of an owner drawn LISTBOX.
LISTBOX_OwnerDraw()	Default function for drawing a LISTBOX entry.
LISTBOX_SetAutoScrollH()	Activates automatic use of a horizontal scroll bar.

Table 19.441: LISTBOX API list

Routine	Description
LISTBOX_SetAutoScrollV()	Activates automatic use of a vertical scroll bar.
LISTBOX_SetBkColor()	Sets the background color.
LISTBOX_SetDefaultBkColor()	Sets the default background color for LISTBOX widgets.
LISTBOX_SetDefaultFont()	Changes the default font for LISTBOX widgets.
LISTBOX_SetDefaultScrollStepH()	Sets the default number of pixels to be scrolled horizontal.
LISTBOX_SetDefaultTextAlign()	Sets the default text alignment for new LISTBOX widgets.
LISTBOX_SetDefaultTextColor()	Sets the default text color for LISTBOX widgets.
LISTBOX_SetFont()	Selects the font.
LISTBOX_SetItemDisabled()	Sets the disabled state of the given item.
LISTBOX_SetItemSel()	Sets the selection state of the given item.
LISTBOX_SetItemSpacing()	Sets a spacing between the items.
LISTBOX_SetMulti()	Sets the multi selection mode on or off.
LISTBOX_SetOwnerDraw()	Enables the list box to be owner drawn.
LISTBOX_SetScrollbarColor()	Sets the colors of the optional scroll bar.
LISTBOX_SetScrollbarWidth()	Sets the width of the scroll bars used by the LISTBOX.
LISTBOX_SetScrollStepH()	Sets the number of pixels to be scrolled horizontal.
LISTBOX_SetSel()	Sets the selected item.
LISTBOX_SetString()	Sets the text of an element.
LISTBOX_SetTextAlign()	Sets the text alignment of the LISTBOX.
LISTBOX_SetTextColor()	Sets the foreground color.
LISTBOX_SetUserData()	Sets the extra data of a LISTBOX widget.

Table 19.441: LISTBOX API list

LISTBOX_AddString()

Description

Adds an item to an already existing LISTBOX widget.

Prototype

```
void LISTBOX_AddString(LISTBOX_Handle hObj, const char * s);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
s	Text to display.

Table 19.442: LISTBOX_AddString() parameter list

LISTBOX_Create()

Description

Creates a LISTBOX widget of a specified size at a specified location.

Prototype

```
LISTBOX_Handle LISTBOX_Create(const GUI_ConstString * ppText,
                              int   x0, int y0,
                              int   xSize, int ySize,
                              int   Flags);
```

Parameter	Description
ppText	Pointer to an array of string pointers containing the elements to be displayed.
x0	Leftmost pixel of the LISTBOX widget (in parent coordinates).
y0	Topmost pixel of the LISTBOX widget (in parent coordinates).

Table 19.443: LISTBOX_Create() parameter list

Parameter	Description
xSize	Horizontal size of the LISTBOX widget (in pixels).
ySize	Vertical size of the LISTBOX widget (in pixels).
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).

Table 19.443: LISTBOX_Create() parameter list

Return value

Handle of the created LISTBOX widget; 0 if the function fails.

Additional information

If the parameter ySize is greater than the required space for drawing the content of the widget, the y-size will be reduced to the required value. The same applies for the xSize parameter.

LISTBOX_CreateAsChild()

Description

Creates a LISTBOX widget as a child window.

Prototype

```
LISTBOX_Handle LISTBOX_CreateAsChild(const   GUI_ConstString * ppText,
                                     WM_HWIN hWinParent,
                                     int     x0,       int y0,
                                     int     xSize,    int ySize,
                                     int     Flags);
```

Parameter	Description
ppText	Pointer to an array of string pointers containing the elements to be displayed.
hParent	Handle of parent window.
x0	X-position of the LISTBOX widget relative to the parent window.
y0	Y-position of the LISTBOX widget relative to the parent window.
xSize	Horizontal size of the LISTBOX widget (in pixels).
ySize	Vertical size of the LISTBOX widget (in pixels).
Flags	Window create flags (see LISTBOX_Create()).

Table 19.444: LISTBOX_CreateAsChild() parameter list

Return value

Handle of the created LISTBOX widget; 0 if the function fails.

Additional information

If the parameter ySize is greater than the space required for drawing the content of the widget, the Y-size will be reduced to the required value. If ySize = 0 the Y-size of the widget will be set to the Y-size of the client area from the parent window. The same applies for the Xsize parameter.

LISTBOX_CreateEx()

Description

Creates a LISTBOX widget of a specified size at a specified location.

Prototype

```
LISTBOX_Handle LISTBOX_CreateEx(int      x0,       int y0,
                                int      xSize,    int ySize,
                                WM_HWIN hParent,   int WinFlags,
                                int      ExFlags,  int Id,
                                const    GUI_ConstString * ppText);
```

Parameter	Description
`x0`	Leftmost pixel of the widget (in parent coordinates).
`y0`	Topmost pixel of the widget (in parent coordinates).
`xSize`	Horizontal size of the widget (in pixels).
`ySize`	Vertical size of the widget (in pixels).
`hParent`	Handle of parent window. If 0, the new HEADER widget will be a child of the desktop (top-level window).
`WinFlags`	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
`ExFlags`	Not used, reserved for future use.
`Id`	Window ID of the widget.
`ppText`	Pointer to an array of string pointers containing the elements to be displayed.

Table 19.445: LISTBOX_CreateEx() parameter list

Return value

Handle of the created LISTBOX widget; 0 if the function fails.

LISTBOX_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `LISTBOX_CreateEx()`.

LISTBOX_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `LISTBOX_CreateEx()` can be referred to.

LISTBOX_DecSel()

Description

Decrement the selection of the LISTBOX widget (moves the selection bar of a specified LISTBOX widget up by one item).

Prototypes

```
void LISTBOX_DecSel(LISTBOX_Handle hObj);
```

Parameter	Description
`hObj`	Handle of LISTBOX widget.

Table 19.446: LISTBOX_DecSel() parameter list

Additional information

Note that the numbering of items always starts from the top with a value of 0; therefore, decrementing the selection will actually move the selection one row up.

LISTBOX_DeleteItem()

Description

Deletes an element from a LISTBOX widget.

Prototypes

```
void LISTBOX_DeleteItem(LISTBOX_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based index of element to be deleted.

Table 19.447: LISTBOX_DeleteItem() parameter list

LISTBOX_EnableWrapMode()

Description

Enables scrolling from the end to the beginning and vice versa. That avoids scrolling from the end through the whole content of the list. If for example the last set of items of the LISTBOX are currently visible and the user attempts scrolling to the next element the beginning of the list will be shown.

Prototypes

```
void LISTBOX_EnableWrapMode(LISTBOX_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
OnOff	1 for enabling wrap mode, 0 (default) for disabling.

Table 19.448: LISTBOX_DeleteItem() parameter list

LISTBOX_GetDefaultBkColor()

Description

Returns the default background color for new LISTBOX widgets.

Prototype

```
GUI_COLOR LISTBOX_GetDefaultBkColor(unsigned Index);
```

Parameter	Description
Index	Zero based index for background color. See table below.

Table 19.449: LISTBOX_GetDefaultBkColor() parameter list

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.

Return value

Default background color for new LISTBOX widgets.

LISTBOX_GetDefaultFont()

Description

Returns the default font used for creating LISTBOX widgets.

Prototype

```
const GUI_FONT * LISTBOX_GetDefaultFont(void);
```


Return value

Pointer to the default font.

LISTBOX_GetDefaultScrollStepH()

Description

Returns the default horizontal scroll step used for creating LISTBOX widgets. The horizontal scroll step defines the number of pixels to be scrolled if needed.

Prototype

```
int LISTBOX_GetDefaultScrollStepH(void);
```

Return value

Default horizontal scroll step.

LISTBOX_GetDefaultTextAlign()

Description

Returns the default text alignment for new LISTBOX widgets.

Prototype

```
int LISTBOX_GetDefaultTextAlign(void);
```

Return value

Default text alignment for new LISTBOX widgets.

Additional information

For more information, refer to “LISTBOX_SetTextAlign()” on page 648.

LISTBOX_GetDefaultTextColor()

Description

Returns the default text color for new LISTBOX widgets.

Prototype

```
GUI_COLOR LISTBOX_GetDefaultTextColor(unsigned Index);
```

Parameter	Description
Index	Zero based index for text color. See table below.

Table 19.450: LISTBOX_GetDefaultTextColor() parameter list

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.

Return value

Default text color for new LISTBOX widgets.

LISTBOX_GetFont()

Description

Returns a pointer to the font used to display the text of the LISTBOX widget.

Prototype

```
const GUI_FONT * LISTBOX_GetFont(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTBOX widget.

Table 19.451: LISTBOX_GetFont() parameter list

Return value

Pointer to the font used to display the text of the LISTBOX widget.

LISTBOX_GetItemDisabled()

Description

Returns if the given item of the LISTBOX widget has been disabled.

Prototype

```
int LISTBOX_GetItemDisabled(LISTBOX_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based index of item.

Table 19.452: LISTBOX_GetItemDisabled() parameter list

Return value

1 if item has been disabled, 0 if not.

LISTBOX_GetItemSel()

Description

Returns the selection state of the given LISTBOX item. The selection state of a LISTBOX item can be modified in multi selection mode only.

Prototype

```
int LISTBOX_GetItemSel(LISTBOX_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based index of item.

Table 19.453: LISTBOX_GetItemSel() parameter list

Return value

1 if item has been selected, 0 if not.

LISTBOX_GetItemText()

Description

Returns the text of the given item of the LISTBOX widget.

Prototype

```
void LISTBOX_GetItemText(LISTBOX_Handle   hObj,    unsigned Index,
```



```
                              char           * pBuffer, int      MaxSize);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based item index.
pBuffer	Pointer to buffer to store the item text.
MaxSize	Size of the buffer.

Table 19.454: LISTBOX_GetItemText() parameter list

Additional information

The function copies the text of the given LISTBOX item into the given buffer.

LISTBOX_GetMulti()

Description

Returns if the multi selection mode of the given LISTBOX widget is active.

Prototype

```
int LISTBOX_GetMulti(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of the LISTBOX widget.

Table 19.455: LISTBOX_GetMulti() parameter list

Return value

1 if active, 0 if not.

LISTBOX_GetNumItems()

Description

Returns the number of items in a specified LISTBOX widget.

Prototypes

```
unsigned LISTBOX_GetNumItems(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTBOX widget.

Table 19.456: LISTBOX_GetNumItems() parameter list

Return value

Number of items in the LISTBOX widget.

LISTBOX_GetScrollStepH()

Description

Returns the horizontal scroll step of the given LISTBOX widget.

Prototype

```
int LISTBOX_GetScrollStepH(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTBOX widget.

Table 19.457: LISTBOX_GetScrollStepH() parameter list

Return value

Horizontal scroll step of the given LISTBOX widget.

LISTBOX_GetSel()

Description

Returns the zero based index of the currently selected item in a specified LISTBOX widget. In multi selection mode the function returns the index of the focused element.

Prototype

```
int LISTBOX_GetSel(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTBOX widget.

Table 19.458: LISTBOX_GetSel() parameter list

Return value

Zero based index of the currently selected item.

Additional information

If no element has been selected the function returns -1.

LISTBOX_GetTextAlign()

Description

Returns the text alignment of the given LISTBOX widget.

Prototype

```
int LISTBOX_GetTextAlign(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTBOX widget.

Table 19.459: LISTBOX_GetTextAlign() parameter list

Return value

Text alignment of the given LISTBOX widget.

Additional information

For more information, refer to "LISTBOX_SetTextAlign()" on page 648.

LISTBOX_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

LISTBOX_IncSel()

Description

Increment the selection of the LISTBOX widget (moves the selection bar of a specified LISTBOX widget down by one item).

Prototypes

```
void LISTBOX_IncSel(LISTBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTBOX widget.

Table 19.460: LISTBOX_IncSel() parameter list

Additional information

Note that the numbering of items always starts from the top with a value of 0; therefore incrementing the selection will actually move the selection one row down.

LISTBOX_InsertString()

Description

Inserts an element into a LISTBOX widget.

Prototypes

```
void LISTBOX_InsertString(LISTBOX_Handle hObj,   const char * s,
                          unsigned int   Index);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
s	Pointer to string to be inserted.
Index	Zero based index of element to be inserted.

Table 19.461: LISTBOX_InsertString() parameter list

LISTBOX_InvalidateItem()

Description

Invalidates an item of a owner drawn LISTBOX widget.

Prototypes

```
void LISTBOX_InvalidateItem(LISTBOX_Handle hObj, int Index);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based index of element to be invalidated or LISTBOX_ALL_ITEMS if all items should be invalidated.

Table 19.462: LISTBOX_InvalidateItem() parameter list

Additional information

This function only needs to be called if an item of an owner drawn LISTBOX widget has been changed. If a LISTBOX API function (like `LISTBOX_SetString()`) has been used to modify a LISTBOX item `LISTBOX_InvalidateItem()` needs not to be called. It needs to be called if the user decides, that for example the vertical size of an item has been changed. With other words if no LISTBOX API function has been used to modify the item this function needs to be called.

LISTBOX_ALL_ITEMS

If all items of a LISTBOX should be invalidated use this define as `Index` parameter.

LISTBOX_OwnerDraw()

Description

Default function to handle a LISTBOX entry.

Prototypes

```
int LISTBOX_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Description
pDrawItemInfo	Pointer to a WIDGET_ITEM_DRAW_INFO structure.

Table 19.463: LISTBOX_OwnerDraw() parameter list

Additional information

This function is useful if `LISTBOX_SetOwnerDraw()` has been used. It can be used from your drawing function to retrieve the original x size of a LISTBOX entry and/or to display the text of a LISTBOX entry and should be called for all unhandled commands.

For more information, refer to the section explaining user drawn widgets, `LISTBOX_SetOwnerDraw()` and to the provided example.

LISTBOX_SetAutoScrollH()

Description

Enables/disables the automatic use of a horizontal scroll bar.

Prototypes

```
void LISTBOX_SetAutoScrollH(LISTBOX_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
OnOff	See table below.

Table 19.464: LISTBOX_SetAutoScrollH() parameter list

Permitted values for parameter OnOff	
0	Disable automatic use of a horizontal scroll bar.
1	Enable automatic use of a horizontal scroll bar.

Additional information

If enabled the LISTBOX widget checks if all elements fits into the LISTBOX widget. If not a horizontal scroll bar will be attached to the window.

LISTBOX_SetAutoScrollV()

Description

Enables/disables the automatic use of a vertical scroll bar.

Prototypes

```
void LISTBOX_SetAutoScrollV(LISTBOX_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
OnOff	See table below.

Table 19.465: LISTBOX_SetAutoScrollV() parameter list

Permitted values for parameter OnOff	
0	Disable automatic use of a vertical scroll bar.
1	Enable automatic use of a vertical scroll bar.

Additional information

If enabled the LISTBOX widget checks if all elements fits into the LISTBOX widget. If not a vertical scroll bar will be added.

LISTBOX_SetBkColor()

Description

Sets the background color of the LISTBOX widget.

Prototype

```
void LISTBOX_SetBkColor(LISTBOX_Handle hObj, unsigned int Index,
```



```
GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Index for background color. See table below.
Color	Color to be set.

Table 19.466: LISTBOX_SetBkColor() parameter list

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.
LISTBOX_CI_DISABLED	Disabled element.

LISTBOX_SetDefaultBkColor()

Description

Sets the default background color for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultBkColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Description
Index	Zero based index for background color. See table below.
Color	Desired background color.

Table 19.467: LISTBOX_SetDefaultBkColor() parameter list

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.
LISTBOX_CI_DISABLED	Disabled element.

LISTBOX_SetDefaultFont()

Description

Sets the default font used for creating LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font.

Table 19.468: LISTBOX_SetDefaultFont() parameter list

LISTBOX_SetDefaultScrollStepH()

Description

Sets the default horizontal scroll step used when creating a LISTBOX widget.

Prototype

```
void LISTBOX_SetDefaultScrollStepH(int Value);
```

Parameter	Description
Value	Number of pixels to be scrolled.

Table 19.469: LISTBOX_SetDefaultScrollStepH() parameter list

LISTBOX_SetDefaultTextAlign()

Description

Sets the default text alignment for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultTextAlign(int Align);
```

Parameter	Description
Align	Default text alignment for new LISTBOX widgets.

Table 19.470: LISTBOX_SetDefaultTextAlign() parameter list

Additional information

For more information, refer to "LISTBOX_SetTextAlign()" on page 648.

LISTBOX_SetDefaultTextColor()

Description

Sets the default text color for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultTextColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Description
Index	Zero based index for text color. See table below.
Color	Desired text color.

Table 19.471: LISTBOX_SetDefaultTextColor() parameter list

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.

LISTBOX_SetFont()

Description

Sets the font of the LISTBOX widget.

Prototype

```
void LISTBOX_SetFont(LISTBOX_Handle hObj, const GUI_FONT * pfont);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
pFont	Pointer to the font.

Table 19.472: LISTBOX_SetFont() parameter list

LISTBOX_SetItemDisabled()

Description

Modifies the disable state of the given list item of the LISTBOX widget.

Prototype

```
void LISTBOX_SetItemDisabled(LISTBOX_Handle hObj, unsigned Index,
                                          int            OnOff);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based index of the LISTBOX item.
OnOff	1 for disabled, 0 for not disabled.

Table 19.473: LISTBOX_SetItemDisabled() parameter list

Additional information

When scrolling through a LISTBOX widget disabled items will be skipped. You can not scroll to a disabled item of a LISTBOX widget.

LISTBOX_SetItemSel()

Description

Modifies the selection state of the given item of the LISTBOX widget.

Prototype

```
void LISTBOX_SetItemSel(LISTBOX_Handle hObj, unsigned Index, int OnOff);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Zero based index of the LISTBOX item.
OnOff	1 for selected, 0 for not selected.

Table 19.474: LISTBOX_SetItemSel() parameter list

Additional information

Setting the selection state of a LISTBOX item makes only sense when using the multi selection mode. See also LISTBOX_SetMulti().

LISTBOX_SetItemSpacing()

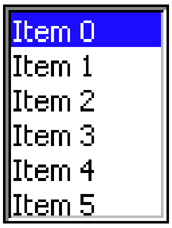
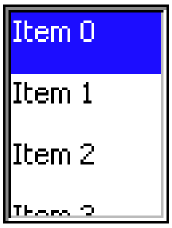
<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 19.475: LISTBOX_SetItemSpacing() befor after screenshots

Description

Sets an additional spacing below the items of a LISTBOX widget.

Prototype

```
void LISTBOX_SetItemSpacing(LISTBOX_Handle hObj, unsigned Value);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Value	Number of pixels used as additional spacing between the items.

Table 19.476: LISTBOX_SetItemSpacing() parameter list

LISTBOX_SetMulti()

Description

Switches the multi selection mode of a LISTBOX on or off.

Prototype

```
void LISTBOX_SetMulti(LISTBOX_Handle hObj, int Mode);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Mode	0 for off, 1 for on.

Table 19.477: LISTBOX_SetMulti() parameter list

Additional information

The multi selection mode enables the LISTBOX widget to have more than one selected element. Using the space key would toggle the selection state of a LISTBOX item.

LISTBOX_SetOwnerDraw()

Description

Sets the LISTBOX widget to be owner drawn.

Prototype

```
void LISTBOX_SetOwnerDraw(LISTBOX_Handle           hObj,
                          WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
pfDrawItem	Pointer to owner draw function.

Table 19.478: LISTBOX_SetOwnerDraw() parameter list

Additional information

This function sets a function pointer to a function which will be called by the widget if a LISTBOX item has to be drawn and when the x or y size of a item is needed. It gives you the possibility to draw anything as LISTBOX item, not just plain text. pfDrawItem is a pointer to a application-defined function of type WIDGET_DRAW_ITEM_FUNC which is explained at the beginning of the chapter.

Structure of the user defined owner draw function

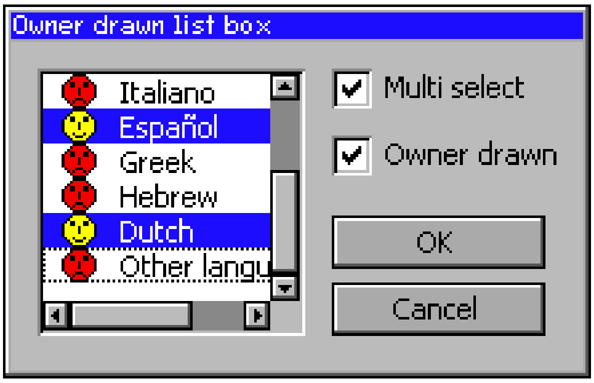
The following shows the structure of a typical owner draw function. It assumes that your LISTBOX entries are 30 pixels wider than and have the same height as the item drawn by the default function:

```
static int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_GET_XSIZE:
    return LISTBOX_OwnerDraw(pDrawItemInfo) + 30; /* Returns the default xSize+10 */
  case WIDGET_ITEM_DRAW:

    /* Your code to be added to draw the LISTBOX item */
```



```
    return 0;
  }
  return LISTBOX_OwnerDraw(pDrawItemInfo);    /* Def. function for unhandled cmds */
}
```

Example



The source code of this example is available in the examples as `WIDGET_ListBoxOwnerDraw`.

LISTBOX_SetScrollbarColor()

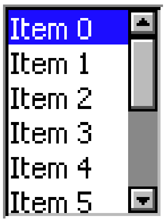
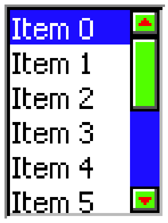
Before	After
	

Table 19.479: LISTBOX_SetScrollbarColor() before after screenshots

Description

Sets the colors of the optional scroll bar.

Prototype

```
void LISTBOX_SetScrollbarColor(LISTBOX_Handle hObj,
                               unsigned int   Index,
                               GUI_COLOR      Color);
```

Parameter	Description
`hObj`	Handle of LISTBOX widget
`Index`	Index of desired item. See table below.
`Color`	Color to be used.

Table 19.480: LISTBOX_SetScrollbarColor() parameter list

Permitted values for parameter `Index`	
`SCROLLBAR_CI_THUMB`	Color of thumb area.
`SCROLLBAR_CI_SHAFT`	Color of shaft.
`SCROLLBAR_CI_ARROW`	Color of arrows.

LISTBOX_SetScrollbarWidth()

Description

Sets the width of the scroll bars used by the given LISTBOX widget.

Prototype

```
void LISTBOX_SetScrollbarWidth(LISTBOX_Handle hObj, unsigned Width);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Width	Width of the scroll bar(s) used by the given LISTBOX widget.

Table 19.481: LISTBOX_SetScrollbarWidth() parameter list

LISTBOX_SetScrollStepH()

Description

Sets the horizontal scroll step of the given LISTBOX widget. The horizontal scroll step defines the number of pixels to be scrolled if needed.

Prototype

```
void LISTBOX_SetScrollStepH(LISTBOX_Handle hObj, int Value);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Value	Number of pixels to be scrolled.

Table 19.482: LISTBOX_SetScrollStepH() parameter list

LISTBOX_SetSel()

Description

Sets the selected item of a specified LISTBOX widget.

Prototype

```
void LISTBOX_SetSel(LISTBOX_Handle hObj, int Sel);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Sel	Element to be selected.

Table 19.483: LISTBOX_SetSel() parameter list

LISTBOX_SetString()

Description

Sets the content of the given item.

Prototypes

```
void LISTBOX_SetString(LISTBOX_Handle hObj,   const char * s,
                       unsigned int   Index);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
s	Pointer to string containing the new content.
Index	Zero based index of element to be changed.

Table 19.484: LISTBOX_SetString() parameter list

LISTBOX_SetTextAlign()

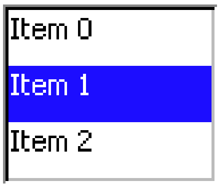
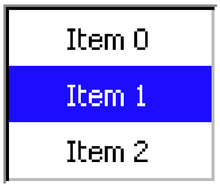
Before	After
	

Table 19.485: LISTBOX_SetTextAlign() before after screenshots

Description

The function sets the text alignment used to display each item of the LISTBOX widget.

Prototype

```
void LISTBOX_SetTextAlign(LISTBOX_Handle hObj, int Align);
```

Parameter	Description
`hObj`	Handle of LISTBOX widget.
`Align`	Text alignment to be used.

Table 19.486: LISTBOX_SetTextAlign() parameter list

Permitted values for parameter `Align` (horizontal and vertical flags are `OR`-combinable)	
Horizontal alignment	
`GUI_TA_LEFT`	Align X-position left (default).
`GUI_TA_HCENTER`	Center X-position.
`GUI_TA_RIGHT`	Align X-position right (default).
Vertical alignment	
`GUI_TA_TOP`	Align Y-position with top of characters (default).
`GUI_TA_VCENTER`	Center Y-position.
`GUI_TA_BOTTOM`	Align Y-position with bottom pixel line of font.

Additional information

The default alignment of list boxes is `GUI_TA_LEFT`. Per default the height of each item depends on the height of the font used to render the LISTBOX items. So vertical text alignment makes only sense if the function `LISTBOX_SetItemSpacing()` is used to set an additional spacing below the items.

LISTBOX_SetTextColor()

Description

Sets the text color of the LISTBOX widget.

Prototype

```
void LISTBOX_SetTextColor(LISTBOX_Handle hObj,   unsigned int Index,
```



```
GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of LISTBOX widget.
Index	Index for text color (see LISTBOX_SetBackColor()).
Color	Color to be set.

Table 19.487: LISTBOX_SetTextColor() parameter list

LISTBOX_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

19.15.6 Examples

The Sample folder contains the following examples which show how the widget can be used:

- WIDGET_SimpleListBox.c
- WIDGET_ListBox.c

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_SimpleListBox.c:

Screenshot(s) of WIDGET_ListBox.c:

19.16 LISTVIEW: Listview widget

LISTVIEW widgets are used to select one element of a list with several columns. To manage the columns a LISTVIEW widget contains a HEADER widget. A LISTVIEW can be created without a surrounding frame window or as a child window of a FRAMEWIN widget. As items in a listview are selected, they appear highlighted. Note that the background color of a selected item depends on whether the LISTVIEW window has input focus. The table below shows the appearance of the LISTVIEW widget:

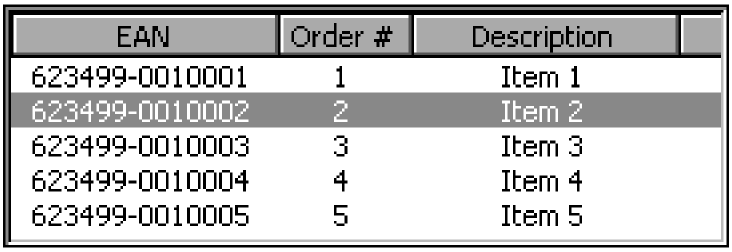
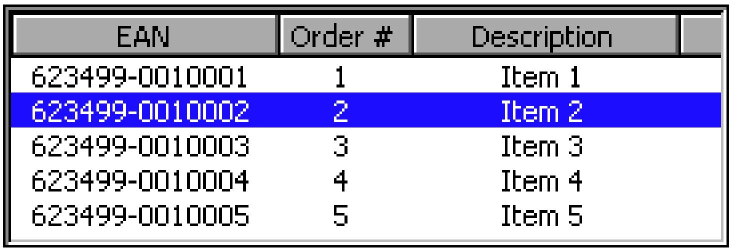
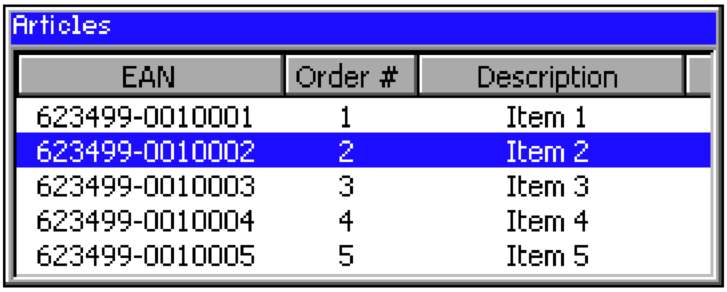
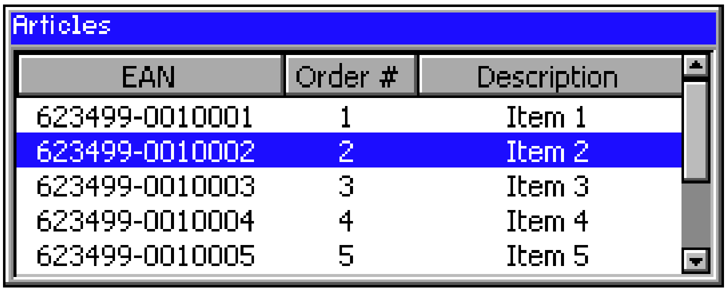
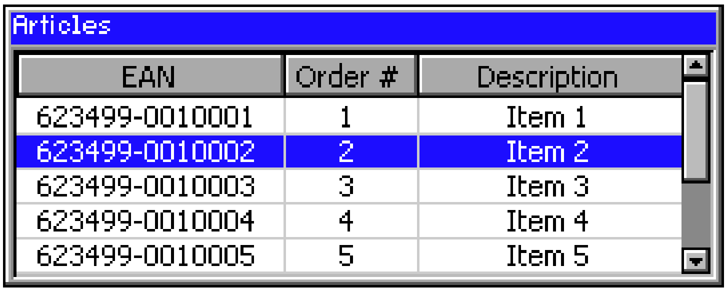
<table>
<tr><th>Description</th><th>LISTVIEW widget</th></tr>
<tr><td>No focus
No surrounding FRAMEWIN
No SCROLLBAR attached
Grid lines not visible</td><td>
</td></tr>
<tr><td>Has input focus
No surrounding FRAMEWIN
No SCROLLBAR attached
Grid lines not visible</td><td>
</td></tr>
<tr><td>Has input focus
With surrounding FRAMEWIN
No SCROLLBAR attached
Grid lines not visible</td><td>
</td></tr>
<tr><td>Has input focus
With surrounding FRAMEWIN
SCROLLBAR attached
Grid lines not visible</td><td>
</td></tr>
<tr><td>Has input focus
With surrounding FRAMEWIN
SCROLLBAR attached
Grid lines visible</td><td>
</td></tr>
</table>

Table 19.488: LISTVIEW appearance

19.16.1 Configuration options

Type	Macro	Default	Description
N	LISTVIEW_ALIGN_DEFAULT	GUI_TA_VCENTER |GUI_TA_HCENTER	Default text alignment.
N	LISTVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	LISTVIEW_BKCOLOR1_DEFAULT	GUI_GRAY	Background color, selected state without focus.
N	LISTVIEW_BKCOLOR2_DEFAULT	GUI_BLUE	Background color, selected state with focus.
N	LISTVIEW_BKCOLOR3_DEFAULT	GUI_LIGHTGRAY	Background color, disabled state.
S	LISTVIEW_FONT_DEFAULT	&GUI_Font13_1	Default font.
N	LISTVIEW_GRIDCOLOR_DEFAULT	GUI_LIGHTGRAY	Color of grid lines (if shown).
N	LISTVIEW_SCROLLSTEP_H_DEFAULT	10	Defines the number of pixels to be scrolled if needed.
N	LISTVIEW_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	LISTVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color, selected state with-out focus.
N	LISTVIEW_TEXTCOLOR2_DEFAULT	GUI_WHITE	Text color, selected state with focus.
N	LISTVIEW_TEXTCOLOR3_DEFAULT	GUI_GRAY	Text color, disabled state.
N	LISTVIEW_WRAPMODE_DEFAULT	GUI_WRAPMODE_NONE	Wrapping mode.

Table 19.489: Configuration options

19.16.2 Predefined IDs

The following symbols define IDs which may be used to make LISTVIEW widgets distinguishable from creation: GUI_ID_LISTVIEW0 - GUI_ID_LISTVIEW3

19.16.3 Notification codes

The following events are sent from a LISTVIEW widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scroll bar has been changed.
WM_NOTIFICATION_SEL_CHANGED	The selection of the list box has changed.

Table 19.490: Notification codes

19.16.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_UP	Moves the selection bar up.
GUI_KEY_DOWN	Moves the selection bar down.
GUI_KEY_RIGHT	If the total amount of the column width is > than the inside area of the listview, the content scrolls to the left.
GUI_KEY_LEFT	If the total amount of the column width is > than the inside area of the listview, the content scrolls to the right.

Table 19.491: Keyboard reaction

19.16.5 LISTVIEW API

The table below lists the available emWin LISTVIEW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
LISTVIEW_AddColumn()	Adds a column to a LISTVIEW.
LISTVIEW_AddRow()	Adds a row to a LISTVIEW.
LISTVIEW_CompareDec()	Compare function for comparing 2 integer values.
LISTVIEW_CompareText()	Compare function for comparing 2 strings.
LISTVIEW_Create()	Creates a LISTVIEW widget. (Obsolete)
LISTVIEW_CreateAttached()	Creates a LISTVIEW widget attached to a window.
LISTVIEW_CreateEx()	Creates a LISTVIEW widget.
LISTVIEW_CreateIndirect()	Creates a LISTVIEW widget from a resource table entry.
LISTVIEW_CreateUser()	Creates a LISTVIEW widget using extra bytes as user data.
LISTVIEW_DecSel()	Decrements selection.
LISTVIEW_DeleteColumn()	Deletes the given column.
LISTVIEW_DeleteRow()	Deletes the given row.
LISTVIEW_DisableRow()	Sets the state of the given row to disabled.
LISTVIEW_DisableSort()	Disables sorting of the LISTVIEW.
LISTVIEW_EnableCellSelect()	Enables/disables cell selection mode of the given widget.
LISTVIEW_EnableRow()	Sets the state of the given row to enabled.
LISTVIEW_EnableSort()	Enables sorting of the LISTVIEW.
LISTVIEW_GetBkColor()	Returns the background color of the LISTVIEW.
LISTVIEW_GetFont()	Returns the font of the LISTVIEW.
LISTVIEW_GetHeader()	Returns the handle of the attached HEADER widget.
LISTVIEW_GetItemRect()	Copies the rectangle coordinates of the specified item.
LISTVIEW_GetItemText()	Copies the text of the specified item to the given buffer.
LISTVIEW_GetNumColumns()	Returns the number of columns.
LISTVIEW_GetNumRows()	Returns the number of rows.
LISTVIEW_GetSel()	Returns the index of the selected row.
LISTVIEW_GetSelCol()	Returns the index of the selected column.
LISTVIEW_GetSelUnsorted()	Returns the index of the selected row in unsorted state.
LISTVIEW_GetTextColor()	Returns the text color of the LISTVIEW widget.
LISTVIEW_GetUserData()	Retrieves the data set with LISTVIEW_SetUserData().
LISTVIEW_GetUserDataRow()	Returns the user data of the given row.
LISTVIEW_GetWrapMode()	Returns the currently used mode for wrapping text.
LISTVIEW_IncSel()	Increments selection.
LISTVIEW_InsertRow()	Inserts a new row at the given position.
LISTVIEW_OwnerDraw()	Default function to be used for drawing a LISTVIEW cell.
LISTVIEW_SetAutoScrollH()	Enables the automatic use of a horizontal scroll bar.
LISTVIEW_SetAutoScrollV()	Enables the automatic use of a vertical scroll bar.
LISTVIEW_SetBkColor()	Sets the background color.
LISTVIEW_SetColumnWidth()	Sets the column width.
LISTVIEW_SetCompareFunc()	Sets the compare function for the given column.
LISTVIEW_SetDefaultBkColor()	Sets the default background color for HEADER widgets.
LISTVIEW_SetDefaultFont()	Sets the default font for HEADER widgets.
LISTVIEW_SetDefaultGridColor()	Sets the default text color for HEADER widgets.
LISTVIEW_SetDefaultTextColor()	Sets the default color of the grid lines for HEADER widgets.
LISTVIEW_SetFixed()	Fixes the given number of columns.
LISTVIEW_SetFont()	Sets the font of the LISTVIEW widget.

Table 19.492: LISTVIEW API list

Routine	Description
LISTVIEW_SetGridVis()	Sets the visibility flag of the grid lines.
LISTVIEW_SetHeaderHeight()	Sets the height of the header.
LISTVIEW_SetItemBitmap()	Sets a bitmap as the background of a LISTVIEW cell
LISTVIEW_SetItemBkColor()	Sets the background color of a LISTVIEW cell
LISTVIEW_SetItemText()	Sets the text of a LISTVIEW cell.
LISTVIEW_SetItemTextColor()	Sets the text color of a LISTVIEW cell
LISTVIEW_SetLBorder()	Sets the number of pixels used for the left border.
LISTVIEW_SetOwnerDraw()	Sets a custom defined function for drawing a cell.
LISTVIEW_SetRBorder()	Sets the number of pixels used for the right border.
LISTVIEW_SetRowHeight()	Sets the row height of the LISTVIEW
LISTVIEW_SetSel()	Sets the current selection.
LISTVIEW_SetSelUnsorted()	Sets the current selection in unsorted state.
LISTVIEW_SetSort()	Sets the column and sorting order to be sorted by.
LISTVIEW_SetTextAlign()	Sets the text alignment of a column.
LISTVIEW_SetTextColor()	Sets the text color.
LISTVIEW_SetUserData()	Sets the extra data of a LISTVIEW widget.
LISTVIEW_SetUserDataRow()	Sets the user data of the given row.
LISTVIEW_SetWrapMode()	Sets the wrapping mode for the given LISTVIEW widget.

Table 19.492: LISTVIEW API list

LISTVIEW_AddColumn()

Description

Adds a new column to a LISTVIEW widget.

Prototype

```
void LISTVIEW_AddColumn(LISTVIEW_Handle   hObj, int Width,
                        const char       * s,    int Align);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Width	Width of the new column
s	Text to be displayed in the HEADER widget
Align	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag. In case of -1 the default alignment for LISTVIEW widgets is used.

Table 19.493: LISTVIEW_AddColumn() parameter list

Permitted values for parameter Align (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left.
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right.
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters.
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

Additional information

The Width-parameter can be 0. If Width = 0 the width of the new column will be calculated by the given text and by the default value of the horizontal spacing.
You can only add columns to an 'empty' LISTVIEW widget. If it contains 1 or more rows you can not add a new column.

LISTVIEW_AddRow()

Description

Adds a new row to a LISTVIEW widget.

Prototype

```
void LISTVIEW_AddRow(LISTVIEW_Handle hObj, const GUI_ConstString * ppText);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
ppText	Pointer to array containing the text of the LISTVIEW cells

Table 19.494: LISTVIEW_AddRow() parameter list

Additional information

The ppText-array should contain one item for each column. If it contains less items the remaining cells left blank.

LISTVIEW_CompareDec()

Description

Compare function for comparing 2 integer values.

Prototype

```
int LISTVIEW_CompareDec(const void * p0, const void * p1);
```

Parameter	Description
p0	Void pointer to first value:
p1	Void pointer to second value.

Table 19.495: LISTVIEW_CompareDec() parameter list

Return value

< 0 if value of cell 0 greater than value of cell 1.
0 if value of cell 0 identical to value of cell 1.
> 0 if value of cell 0 less than value of cell 1.

Additional information

The purpose of this function is to be used by the listviews sorting algorithm if the cell text represents integer values.
For details about how to use this function for sorting, refer also to "LISTVIEW_SetCompareFunc()" on page 666.
The Sample folder contains the example WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_CompareText()

Description

Function for comparison of 2 strings.

Prototype

```
int LISTVIEW_CompareText(const void * p0, const void * p1);
```

Parameter	Description
p0	Void pointer to first text:
p1	Void pointer to second text.

Table 19.496: LISTVIEW_CompareText() parameter list

Return value

> 0 if text of cell 0 greater than text of cell 1.
0 if text of cell 0 identical to text of cell 1.
< 0 if text of cell 0 less than text of cell 1.

Additional information

The purpose of this function is to be used by the listviews sorting algorithm.
For details about how to use this function for sorting, refer also to "LISTVIEW_SetCompareFunc()" on page 666.
The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_Create()

(Obsolete, `LISTVIEW_CreateEx()` should be used instead)

Description

Creates a LISTVIEW widget of a specified size at a specified location.

Prototype

```
LISTVIEW_Handle LISTVIEW_Create(int     x0,      int y0,
                                int     xSize,   int ySize,
                                WM_HWIN hParent, int Id,
                                int     Flags,   int SpecialFlags);
```

Parameter	Description
`x0`	Leftmost pixel of the HEADER widget (in parent coordinates).
`y0`	Topmost pixel of the HEADER widget (in parent coordinates).
`xSize`	Horizontal size of the HEADER widget (in pixels).
`ySize`	Vertical size of the HEADER widget (in pixels).
`hParent`	Handle of the parent window
`Id`	Id of the new HEADER widget
`Flags`	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
`SpecialFlags`	(Reserved for later use)

Table 19.497: LISTVIEW_Create() parameter list

Return value

Handle of the created LISTVIEW widget; 0 if the function fails.

LISTVIEW_CreateAttached()

Description

Creates a LISTVIEW widget which is attached to an existing window.

Prototype

```
LISTVIEW_Handle LISTVIEW_CreateAttached(WM_HWIN hParent,       int Id,
                                        int     SpecialFlags);
```

Parameter	Description
`hObj`	Handle to a LISTVIEW widget
`Id`	Id of the new LISTVIEW widget
`SpecialFlags`	(Not used, reserved for later use)

Table 19.498: LISTVIEW_CreateAttached() parameter list

Return value

Handle of the created LISTVIEW widget; 0 if the function fails.

Additional information

An attached LISTVIEW widget is essentially a child window which will position itself on the parent window and operate accordingly.

LISTVIEW_CreateEx()

Description

Creates a LISTVIEW widget of a specified size at a specified location.

Prototype

```
LISTVIEW_Handle LISTVIEW_CreateEx(int     x0,      int y0,
                                  int     xSize,   int ySize,
                                  WM_HWIN hParent, int WinFlags,
                                  int     ExFlags, int Id);
```

Parameter	Description
`x0`	Leftmost pixel of the widget (in parent coordinates).
`y0`	Topmost pixel of the widget (in parent coordinates).
`xSize`	Horizontal size of the widget (in pixels).
`ySize`	Vertical size of the widget (in pixels).
`hParent`	Handle of parent window. If 0, the new LISTVIEW widget will be a child of the desktop (top-level window).
`WinFlags`	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
`ExFlags`	Not used, reserved for future use.
`Id`	Window ID of the widget.

Table 19.499: LISTVIEW_CreateEx() parameter list

Return value

Handle of the created LISTVIEW widget; 0 if the function fails.

LISTVIEW_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `LISTVIEW_CreateEx()`.

LISTVIEW_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `LISTVIEW_CreateEx()` can be referred to.

LISTVIEW_DecSel()

Description

Decrement the listview selection (moves the selection bar of a specified listview up by one item, if possible).

Prototype

```
void LISTVIEW_DecSel(LISTVIEW_Handle hObj);
```

Parameter	Description
`hObj`	Handle to a LISTVIEW widget

Table 19.500: LISTVIEW_DecSel() parameter list

Additional information

Note that the numbering of items always starts from the top with a value of 0; therefore, decrementing the selection will actually move the selection one row up.

LISTVIEW_DeleteColumn()

Description

Deletes the specified column of the LISTVIEW widget.

Prototype

```
void LISTVIEW_DeleteColumn(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Zero based index of column to be deleted.

Table 19.501: LISTVIEW_DeleteColumn() parameter list

Additional information

Note that the numbering of items always starts from the left with a value of 0.

LISTVIEW_DeleteRow()

Description

Deletes the specified row of the LISTVIEW widget.

Prototype

```
void LISTVIEW_DeleteRow(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of widget
Index	Zero based index of row to be deleted.

Table 19.502: LISTVIEW_DeleteRow() parameter list

Additional information

Note that the numbering of items always starts from the top with a value of 0.

LISTVIEW_DisableRow()

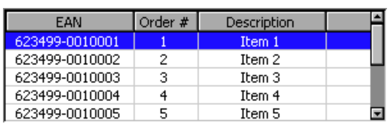
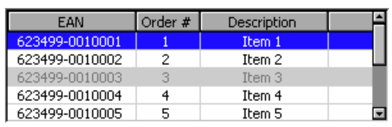
Before	After
	

Table 19.503: LISTVIEW_DisableRow() before after screenshots

Description

The function sets the state of the given row to disabled.

Prototype

```
void LISTVIEW_DisableRow(LISTVIEW_Handle hObj, unsigned Row);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Row	Zero based index of the row to be disabled.

Table 19.504: LISTVIEW_DisableRow() parameter list

Additional information

When scrolling through a LISTVIEW widget disabled items will be skipped. You can not scroll to a disabled listview item.

LISTVIEW_DisableSort()

Description

Disables sorting of the given LISTVIEW widget. After calling this function the content of the LISTVIEW widget will be shown unsorted.

Prototype

```
void LISTVIEW_DisableSort(LISTVIEW_Handle hObj);
```

Parameter	Description
`hObj`	Handle to a LISTVIEW widget

Table 19.505: LISTVIEW_DisableSort() parameter list

Additional information

For details about how to use sorting in LISTVIEW widgets, refer to “LISTVIEW_SetCompareFunc()” on page 666 and “LISTVIEW_SetSort()” on page 675.
The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_EnableCellSelect()

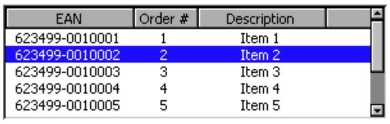
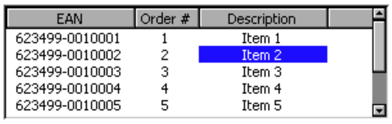
Before	After
	

Table 19.506: LISTVIEW_EnableCellSelect() before after screenshots

Description

Enables or disables cell selection mode of the given LISTVIEW widget. If cell selection is enabled it is possible to select a cell via the keys up, down, left and right.

Prototype

```
void LISTVIEW_EnableCellSelect(LISTVIEW_Handle hObj, unsigned OnOff);
```

Parameter	Description
`hObj`	Handle to a LISTVIEW widget
`OnOff`	1 to enable and 0 to disable cell selection mode.

Table 19.507: LISTVIEW_EnableCellSelect() parameter list

LISTVIEW_EnableRow()

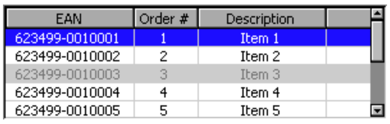
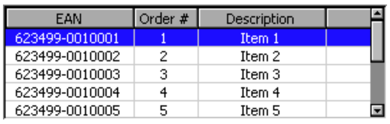
Before	After
	

Table 19.508: LISTVIEW_EnableRow() before after screenshots

Description

The function sets the state of the given row to enabled.

Prototype

```
void LISTVIEW_EnableRow(LISTVIEW_Handle hObj, unsigned Row);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Row	Zero based index of the row to be disabled.

Table 19.509: LISTVIEW_EnableRow() parameter list

Additional information

Refer to "LISTVIEW_DisableRow()" on page 657.

LISTVIEW_EnableSort()

Description

Enables sorting for the given LISTVIEW widget. After calling this function the content of the listview can be rendered sorted after clicking on the header item of the desired column, by which the LISTVIEW widget should sort its data. Note that this works only after a compare function for the desired column has been set.

Prototype

```
void LISTVIEW_EnableSort(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget

Table 19.510: LISTVIEW_EnableSort() parameter list

Additional information

For details about how to set a compare function, refer to "LISTVIEW_SetCompareFunc()" on page 666.
The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_GetBkColor()

Description

Returns the background color of the given LISTVIEW widget.

Prototype

```
GUI_COLOR LISTVIEW_GetBkColor(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of the LISTVIEW widget.
Index	Color index. See table below.

Table 19.511: LISTVIEW_GetBkColor() parameter list

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

Return value

Background color of the given LISTVIEW widget.

LISTVIEW_GetFont()

Description

Returns a pointer to the font used to display the text of the LISTVIEW widget.

Prototype

```
const GUI_FONT * LISTVIEW_GetFont(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.

Table 19.512: LISTVIEW_GetFont() parameter list

Return value

Pointer to the font used to display the text of the LISTVIEW widget.

LISTVIEW_GetHeader()

Description

Returns the handle of the HEADER widget.

Prototype

```
HEADER_Handle LISTVIEW_GetHeader(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle of the LISTVIEW widget.

Table 19.513: LISTVIEW_GetHeader() parameter list

Return value

Handle of the HEADER widget.

Additional information

Each LISTVIEW widget contains a HEADER widget to manage the columns. You can use this handle to change the properties of the LISTVIEW-HEADER, for example to change the text color of the HEADER widget.
Example:

```
LISTVIEW_Handle hListView = LISTVIEW_Create(10, 80, 270, 89, 0, 1234, WM_CF_SHOW, 0);
HEADER_Handle   hHeader   = LISTVIEW_GetHeader(hListView);
HEADER_SetTextColor(hHeader, GUI_GREEN);
```

LISTVIEW_GetItemRect()

Description

Returns the rectangle of the given LISTVIEW cell by copying the coordinates to the given `GUI_RECT` structure.

Prototype

```
void LISTVIEW_GetItemRect(LISTVIEW_Handle   hObj, U32 Col, U32 Row,
```



```
                          GUI_RECT         * pRect);
```

Parameter	Description
hObj	Handle of the LISTVIEW widget.
Col	Zero based index of the cell's column.
Row	Zero based index of the cell's row
pRect	Pointer to a rectangle to be filled with the item coordinates.

Table 19.514: LISTVIEW_GetItemRect() parameter list

LISTVIEW_GetItemText()

Description

Copies the text of the specified item to the given buffer.

Prototype

```
void LISTVIEW_GetItemText(LISTVIEW_Handle hObj,      unsigned   Column,
                          unsigned         Row,       char     * pBuffer,
                          unsigned         MaxSize);
```

Parameter	Description
hObj	Handle of widget
Column	Zero based index of the cell's column.
Row	Zero based index of the cell's row
pBuffer	Pointer to a buffer to be filled by the routine.
MaxSize	Size in bytes of the buffer.

Table 19.515: LISTVIEW_GetItemText() parameter list

Additional information

If the text of the cell does not fit into the buffer, the number of bytes specified by the parameter `MaxSize` will be copied to the buffer.

LISTVIEW_GetNumColumns()

Description

Returns the number of columns of the given LISTVIEW widget.

Prototype

```
unsigned LISTVIEW_GetNumColumns(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle of widget

Table 19.516: LISTVIEW_GetNumColumns() parameter list

Return value

Number of columns of the given LISTVIEW widget.

LISTVIEW_GetNumRows()

Description

Returns the number of rows of the given LISTVIEW widget.

Prototype

```
unsigned LISTVIEW_GetNumRows(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget

Table 19.517: LISTVIEW_GetNumRows() parameter list

Return value

Number of rows of the given LISTVIEW widget.

LISTVIEW_GetSel()

Description

Returns the index of the currently selected row in a specified LISTVIEW widget.

Prototype

```
int LISTVIEW_GetSel(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.

Table 19.518: LISTVIEW_GetSel() parameter list

Return value

Index of the currently selected row.

LISTVIEW_GetSelCol()

Description

Returns the index of the currently selected column.

Prototype

```
int LISTVIEW_GetSel(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.

Table 19.519: LISTVIEW_GetSel() parameter list

Return value

Index of the currently selected column.

LISTVIEW_GetSelUnsorted()

Description

Returns the index of the currently selected row in unsorted state.

Prototype

```
int LISTVIEW_GetSelUnsorted(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.

Table 19.520: LISTVIEW_GetSelUnsorted() parameter list

Return value

Index of the currently selected row in unsorted state.

Additional information

This function returns the actual index of the selected row, whereas the function `LISTVIEW_GetSel()` only returns the index of the sorted row. The actual (unsorted) row index should be used in function calls as row index.
The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_GetTextColor()

Description

Returns the text color of the given LISTVIEW widget.

Prototype

```
GUI_COLOR LISTVIEW_GetTextColor(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Index of color. See table below.

Table 19.521: LISTVIEW_GetTextColor() parameter list

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

Return value

Text color of the given LISTVIEW widget.

LISTVIEW_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

LISTVIEW_GetUserDataRow()

Description

Returns the user data of the given row.

Prototype

```
U32 LISTVIEW_GetUserData(LISTVIEW_Handle hObj, unsigned Row);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Row	Zero based index of row.

Table 19.522: LISTVIEW_GetUserDataRow() parameter list

Return value

User data of the given row.

Additional information

Details on how to set user data for a row can be found in the description of the function “LISTVIEW_SetUserDataRow()” on page 676.

LISTVIEW_GetWrapMode()

Description

Returns the currently used mode for wrapping text.

Prototype

```
GUI_WRAPMODE LISTVIEW_GetWrapMode(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget

Table 19.523: LISTVIEW_GetWrapMode() parameter list

Return value

Currently used wrap mode.

Additional information

Details on how to use text wrapping can be found in the description of the function "LISTVIEW_SetWrapMode()" on page 677.

LISTVIEW_IncSel()

Description

Increment the selection of the LISTVIEW widget (moves the selection bar of a specified LISTVIEW down by one item).

Prototype

```
void LISTVIEW_IncSel(LISTVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget

Table 19.524: LISTVIEW_IncSel() parameter list

LISTVIEW_InsertRow()

Description

Inserts a new row into the LISTVIEW at the given position.

Prototype

```
int LISTVIEW_InsertRow(LISTVIEW_Handle          hObj,    unsigned Index,
                       const GUI_ConstString * ppText);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Index of the new row.
ppText	Pointer to a string array containing the cell data of the new row.

Table 19.525: LISTVIEW_InsertRow() parameter list

Return value

0 if function succeed, 1 if an error occurs.

Additional information

The ppText-array should contain one item for each column. If it contains less items the remaining cells left blank.
If the given index is >= the current number of rows, the function LISTVIEW_AddRow() will be used to add the new row.
The Sample folder contains the example WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_OwnerDraw()

Description

Default function for managing drawing operations of one cell.

Prototype

```
int LISTVIEW_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.

Table 19.526: LISTVIEW_OwnerDraw() parameter list

Return value

Depends on the command in the `Cmd` element of the `WIDGET_ITEM_DRAW_INFO` structure pointed by `pDrawItemInfo`.

Additional information

This function is useful if `LISTVIEW_SetOwnerDraw()` is used. It can be used to retrieve the original size of a data item and/or to draw the text of a data item and should be called for all commands which are not managed by the application defined owner draw function.
The following commands are managed by the default function:

- `WIDGET_ITEM_GET_XSIZE`
- `WIDGET_ITEM_GET_YSIZE`
- `WIDGET_ITEM_DRAW`
- `WIDGET_DRAW_BACKGROUND`

LISTVIEW_SetAutoScrollH()

Description

Enables/disables the automatic use of a horizontal scroll bar.

Prototype

```
void LISTVIEW_SetAutoScrollH(LISTVIEW_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
OnOff	See table below.

Table 19.527: LISTVIEW_SetAutoScrollH() parameter list

Permitted values for parameter OnOff	
0	Disable automatic use of a horizontal scroll bar.
1	Enable automatic use of a horizontal scroll bar.

Additional information

If enabled the LISTVIEW checks if all columns fit into the widgets area. If not a horizontal scroll bar will be added.

LISTVIEW_SetAutoScrollV()

Description

Enables/disables the automatic use of a vertical scroll bar.

Prototype

```
void LISTVIEW_SetAutoScrollV(LISTVIEW_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
OnOff	See table below.

Table 19.528: LISTVIEW_SetAutoScrollV() parameter list

Permitted values for parameter OnOff	
0	Disable automatic use of a vertical scroll bar.
1	Enable automatic use of a vertical scroll bar.

Additional information

If enabled the LISTVIEW checks if all rows fit into the widgets area. If not a vertical scroll bar will be added.

LISTVIEW_SetBkColor()

Description

Sets the background color of the given LISTVIEW widget.

Prototype

```
void LISTVIEW_SetBkColor(LISTVIEW_Handle hObj,   unsigned int Index,
                                   GUI_COLOR       Color);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Index for background color. See table below.
Color	Color to be set.

Table 19.529: LISTVIEW_SetBkColor() parameter list

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

Additional information

To set the background color for a single cell the function LISTVIEW_SetItemBkColor() should be used.
The Sample folder contains the example WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_SetColumnWidth()

Description

Sets the width of the given column.

Prototype

```
void LISTVIEW_SetColumnWidth(LISTVIEW_Handle hObj,   unsigned int Index,
                                        int             Width);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Number of column
Width	New width

Table 19.530: LISTVIEW_SetColumnWidth() parameter list

LISTVIEW_SetCompareFunc()

Description

Sets the compare function for the given column. A compare function needs to be set if the LISTVIEW widget should be sorted by the given column.

Prototype

```
void LISTVIEW_SetCompareFunc(LISTVIEW_Handle hObj, unsigned Column,
                             int (* fpCompare)(const void * p0, const void * p1));
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Column	Index of the desired column for which the compare function should be set.
fpCompare	Function pointer to compare function.

Table 19.531: LISTVIEW_SetCompareFunc() parameter list

Additional information

If the sorting feature of the listview widget is used, the widget uses a compare function to decide if the content of one cell is greater, equal or less than the content of the other cell.
Per default no compare function is set for the LISTVIEW columns. For each column which should be used for sorting, a compare function needs to be set.
The cells of the LISTVIEW widget contain text. But sometimes the text represents data of other types like dates, integers or others. So different compare functions are required for sorting. emWin provides 2 compare functions:

`LISTVIEW_CompareText(): Function` can be used for comparing cells containing text.

`LISTVIEW_CompareDec(): Function` can be used for comparing cells which text, where the content represents integer values.

The compare function should return a value >0, if the content of the second cell is greater than the content of the first cell and <0, if the content of the second cell is less than the content of the first cell or 0 if equal.
Also user defined compare functions can be used. The prototype of a application-defined function should be defined as follows:

Prototype

```
int APPLICATION_Compare(const void * p0, const void * p1);
```

Parameter	Description
p0	Pointer to NULL terminated string data of the first cell.
p1	Pointer to NULL terminated string data of the second cell.

Table 19.532: Parameter list of application defined compare function

Example

```
int APPLICATION_Compare(const void * p0, const void * p1) {
  return strcmp((const char *)p1, (const char *)p0);
}

void SetAppCompareFunc(WM_HWIN hListView, int Column) {
  LISTVIEW_SetCompareFunc(hListView, Column, APPLICATION_Compare);
}
```

The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_SetDefaultBkColor()

Description

Sets the default background color for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultBkColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Description
`Index`	Index of default background color. See table below.
`Color`	Color to be set as default

Table 19.533: LISTVIEW_SetDefaultBkColor() parameter list

Permitted values for parameter `Index`	
`LISTVIEW_CI_UNSEL`	Unselected element.
`LISTVIEW_CI_SEL`	Selected element, without focus.
`LISTVIEW_CI_SELFOCUS`	Selected element, with focus.
`LISTVIEW_CI_DISABLED`	Disabled element.

Return value

Previous default background color.

LISTVIEW_SetDefaultFont()

Description

Sets the default font for new LISTVIEW widgets.

Prototype

```
const GUI_FONT * LISTVIEW_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
`pFont`	Pointer to font used for new LISTVIEW widgets

Table 19.534: LISTVIEW_SetDefaultFont() parameter list

Return value

Pointer to the previous default font.

LISTVIEW_SetDefaultGridColor()

Description

Sets the default color of the grid lines for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultGridColor(GUI_COLOR Color);
```

Parameter	Description
`Color`	New default value

Table 19.535: LISTVIEW_SetDefaultGridColor() parameter list

Return value

Previous default grid color.

LISTVIEW_SetDefaultTextColor()

Description

Sets the default text color for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultTextColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Index of default text color. See table below.
Color	Color to be set as default

Table 19.536: LISTVIEW_SetDefaultTextColor() parameter list

Permitted values for parameter Index	
0	Unselected element.
1	Selected element, without focus.
2	Selected element, with focus.

Return value

Previous default text color.

LISTVIEW_SetFixed()

Description

Fixes the given number of columns at their horizontal positions.

Prototype

```
unsigned LISTVIEW_SetFixed(LISTVIEW_Handle hObj, unsigned Fixed);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
Fixed	Number of columns to be fixed at their horizontal positions.

Table 19.537: LISTVIEW_SetFixed() parameter list

Additional information

Using this function makes sense if one or more columns should remain at their horizontal positions during scrolling operations.

LISTVIEW_SetFont()

Description

Sets the font of the LISTVIEW widget.

Prototype

```
void LISTVIEW_SetFont(LISTVIEW_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
pFont	Pointer to the font.

Table 19.538: LISTVIEW_SetFont() parameter list

LISTVIEW_SetGridVis()

Description

Sets the visibility flag of the grid lines. When creating a LISTVIEW the grid lines are disabled per default.

Prototype

```
int LISTVIEW_SetGridVis(LISTVIEW_Handle hObj, int Show);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Show	Sets the visibility of the grid lines

Table 19.539: LISTVIEW_SetGridVis() parameter list

Permitted values for parameter Show	
0	Not visible.
1	Visible

Return value

Previous value of the visibility flag.

LISTVIEW_SetHeaderHeight()

Description

Sets the height of the attached HEADER widget.

Prototype

```
void LISTVIEW_SetHeaderHeight(LISTVIEW_Handle hObj, unsigned HeaderHeight);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
Show	Height of the attached HEADER widget to be set.

Table 19.540: LISTVIEW_SetHeaderHeight() parameter list

Additional information

Setting the height to 0 causes the HEADER widget not to be displayed.

LISTVIEW_SetItemBitmap()

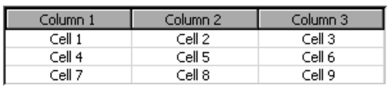
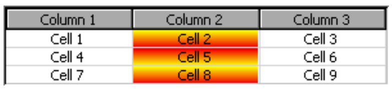
Before	After
	

Table 19.541: LISTVIEW_SetItemBitmap() before after screenshot

Description

Sets a bitmap as background of the given cell.

Prototype

```
void LISTVIEW_SetItemBitmap(LISTVIEW_Handle hObj,
                            unsigned Column, unsigned Row,
                            int xOff, int yOff,
                            const GUI_BITMAP * pBitmap);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Column	Number of column
Row	Number of row

Table 19.542: LISTVIEW_SetItemBitmap() parameter list

Parameter	Description
xOff	Offset for the leftmost pixel of the bitmap to be drawn
yOff	Offset for the topmost pixel of the bitmap to be drawn
pBitmap	Pointer to the bitmap

Table 19.542: LISTVIEW_SetItemBitmap() parameter list

LISTVIEW_SetItemBkColor()

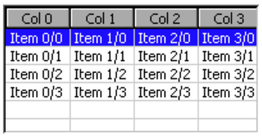
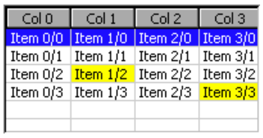
Before	After
	

Table 19.543: LISTVIEW_SetItemBkColor() before after screenshots

Description

Sets the background color of the given cell.

Prototype

```
void LISTVIEW_SetItemBkColor(LISTVIEW_Handle hObj,
                             unsigned         Column, unsigned  Row,
                             unsigned int     Index,  GUI_COLOR Color);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Column	Number of columns.
Row	Number of rows.
Index	Index of background color. See table below.
Color	Color to be used.

Table 19.544: LISTVIEW_SetItemBkColor() parameter list

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

Additional information

This function overwrites the default background color for the given cell set by `LISTVIEW_SetBkColor()`.

LISTVIEW_SetItemText()

Description

Sets the text of one cell of the LISTVIEW widget specified by row and column.

Prototype

```
void LISTVIEW_SetItemText(LISTVIEW_Handle hObj, unsigned     Column,
```



```
                                         unsigned         Row,  const char * s);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
Column	Number of column.
Row	Number of row.
s	Text to be displayed in the table cell.

Table 19.545: LISTVIEW_SetItemText() parameter list

LISTVIEW_SetItemTextColor()

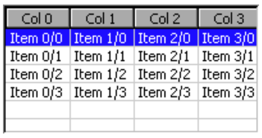
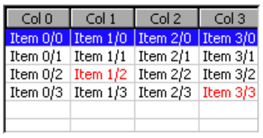
Before	After
	

Table 19.546: LISTVIEW_SetItemTextColor() before after screenshots

Description

Sets the text color of the given cell.

Prototype

```
void LISTVIEW_SetItemTextColor(LISTVIEW_Handle hObj,
                               unsigned        Column, unsigned  Row,
                               unsigned int    Index,  GUI_COLOR Color);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Column	Number of column.
Row	Number of row.
Index	Index of text color. See table below.
Color	Color to be used.

Table 19.547: LISTVIEW_SetItemTextColor() parameter list

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

Additional information

This function overwrites the default text color for the given cell set by `LISTVIEW_SetTextColor()`.

LISTVIEW_SetItemTextSorted()

Description

Sets the text of the given item in a sorted LISTVIEW widget.

Prototype

```
void LISTVIEW_SetItemTextSorted(LISTVIEW_Handle hObj, unsigned Column,
```



```
                                        unsigned        Row,   const char * pText);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
Column	Number of column.
Row	Number of row.
pText	Pointer to the given NULL terminated text.

Table 19.548: LISTVIEW_SetItemTextSorted() parameter list

LISTVIEW_SetLBorder()

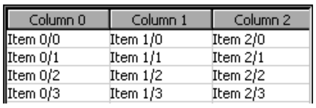
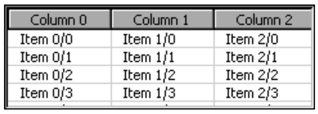
Before	After
	

Table 19.549: LISTVIEW_SetLBorder() before after screenshots

Description

Sets the number of pixels used for the left border within each cell of the LISTVIEW widget.

Prototype

```
void LISTVIEW_SetLBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
BorderSize	Number of pixels to be used.

Table 19.550: LISTVIEW_SetLBorder() parameter list

Additional information

Using this function has no effect to the HEADER widget used by the LISTVIEW widget.

LISTVIEW_SetOwnerDraw()

Description

Sets an application defined owner draw function for the widget which is responsible for drawing a cell.

Prototype

```
void LISTVIEW_SetOwnerDraw(LISTVIEW_Handle hObj,
                           WIDGET_DRAW_ITEM_FUNC * pfOwnerDraw);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
pfOwnerDraw	Pointer to owner draw function.

Table 19.551: LISTVIEW_SetOwnerDraw() parameter list

Additional information

This function sets a pointer to an application defined function which will be called by the widget when a cell has to be drawn or when the x or y size of a item is needed. It gives you the possibility to draw anything as data item, not just plain text. `pfDrawItem` is a pointer to an application-defined function of type `WIDGET_DRAW_ITEM_FUNC` which is explained at the beginning of the chapter.
The following commands are supported: `WIDGET_ITEM_GET_XSIZE`, `WIDGET_ITEM_GET_YSIZE`, `WIDGET_ITEM_DRAW`, and `WIDGET_DRAW_BACKGROUND`.

LISTVIEW_SetRBorder()

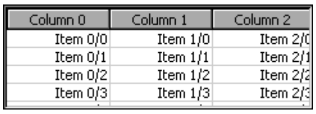
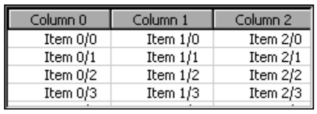
Before	After
	

Table 19.552: LISTVIEW_SetRBorder() before after screenshot

Description

Sets the number of pixels used for the right border within each cell of the LISTVIEW widget.

Prototype

```
void LISTVIEW_SetRBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
BorderSize	Number of pixels to be used.

Table 19.553: LISTVIEW_SetRBorder() parameter list

Additional information

Using this function has no effect to the header widget used by the listview.

LISTVIEW_SetRowHeight()

Description

Sets a constant row height.

Prototype

```
unsigned LISTVIEW_SetRowHeight(LISTVIEW_Handle hObj, unsigned RowHeight);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
RowHeight	Constant row height to set. In case RowHeight == 0, the height of the current font is used instead.

Table 19.554: LISTVIEW_SetRowHeight() parameter list

Return value

Previous value of the row height set by this function.

Additional information

Per default the height of the rows depends on the height of the used font.

LISTVIEW_SetSel()

Description

Sets the selected row of a specified LISTVIEW widget.

Prototype

```
void LISTVIEW_SetSel(LISTVIEW_Handle hObj, int Sel);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Sel	Element to be selected.

Table 19.555: LISTVIEW_SetSel() parameter list

LISTVIEW_SetSelUnsorted()

Description

Sets the index of the currently selected row in unsorted state.

Prototype

```
void LISTVIEW_SetSelUnsorted(LISTVIEW_Handle hObj, int Sel);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
Sel	Zero based selection index in unsorted state.

Table 19.556: LISTVIEW_SetSelUnsorted() parameter list

Additional information

This function sets the actually index of the selected row, whereas the function `LISTVIEW_SetSel()` sets the index of the sorted row. The actual (unsorted) row index should be used in function calls as row index.
The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_SetSort()

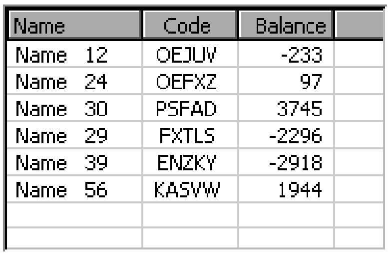
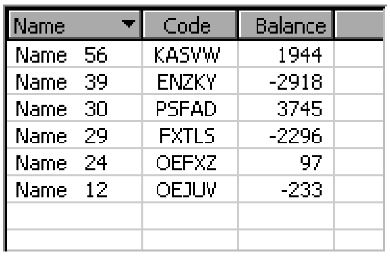
<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 19.557: LISTVIEW_SetSort() before after screenshots

Description

This function sets the column to be sorted by and the sorting order.

Prototype

```
unsigned LISTVIEW_SetSort(LISTVIEW_Handle hObj,     unsigned Column,
                          unsigned        Reverse);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
Column	Column to be sorted by.
Reverse	0 for normal sorting order (greatest element at the top), 1 for reverse order.

Table 19.558: LISTVIEW_SetSort() parameter list

Return value

0 if function was successfully, 1 if not.

Additional information

Before calling this function a compare function needs to be set for the desired column. For details about how to set a compare function, refer to "LISTVIEW_SetCompareFunc()" on page 666.
The `Sample` folder contains the example `WIDGET_SortedListview.c` which shows how to use the function.

LISTVIEW_SetTextAlign()

Description

Sets the alignment for the given column.

Prototype

```
void LISTVIEW_SetTextAlign(LISTVIEW_Handle hObj,   unsigned int Index,
                           int              Align);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Number of column
Align	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

Table 19.559: LISTVIEW_SetTextAlign() parameter list

Permitted values for parameter Align (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
GUI_TA_LEFT	Align X-position left (default).
GUI_TA_HCENTER	Center X-position.
GUI_TA_RIGHT	Align X-position right (default).
Vertical alignment	
GUI_TA_TOP	Align Y-position with top of characters (default).
GUI_TA_VCENTER	Center Y-position.
GUI_TA_BOTTOM	Align Y-position with bottom pixel line of font.

LISTVIEW_SetTextColor()

Description

Sets the text color of the given LISTVIEW widget.

Prototype

```
void LISTVIEW_SetTextColor(LISTVIEW_Handle hObj,   unsigned int Index,
                           GUI_COLOR       Color);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Index	Index for text color. See table below.
Color	Color to be set.

Table 19.560: LISTVIEW_SetTextColor() parameter list

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

LISTVIEW_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

LISTVIEW_SetUserDataRow()

Description

Sets the user data of the given row.

Prototype

```
void LISTVIEW_SetUserData(LISTVIEW_Handle hObj,      unsigned Row,
                          U32              UserData);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget
Row	Row for which the user data should be set
UserData	Value to be associated with the row.

Table 19.561: LISTVIEW_SetUserDataRow() parameter list

Additional information

Sets the 32-bit value associated with the row. Each row has a corresponding 32-bit value intended for use by the application.

LISTVIEW_SetWrapMode()

Description

Sets the wrapping mode which should be used for the cells of the given LISTVIEW widget.

Prototype

```
void LISTVIEW_SetWrapMode(ICONVIEW_Handle hObj, GUI_WRAPMODE WrapMode);
```

Parameter	Description
hObj	Handle to a LISTVIEW widget.
WrapMode	See table below.

Table 19.562: LISTVIEW_SetWrapMode() parameter list

Permitted values for parameter WrapMode	
GUI_WRAPMODE_NONE	No wrapping will be performed.
GUI_WRAPMODE_WORD	Text is wrapped word wise.
GUI_WRAPMODE_CHAR	Text is wrapped char wise.

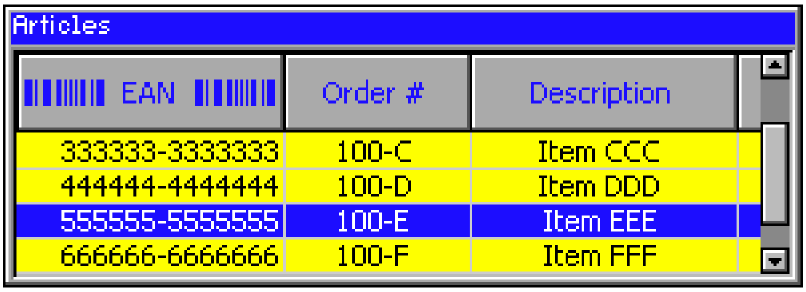
19.16.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_ListView.c`

Note that several other examples also make use of this widget and may also be help-ful to get familiar with the widget.

Screenshot of WIDGET_ListView.c:



19.17 LISTWHEEL: Listwheel widget

This widget is similar to the LISTBOX widget described earlier in this chapter. Whereas the data of a LISTBOX is selected by moving the cursor with the keyboard or by using a SCROLLBAR the LISTWHEEL works completely different: The whole data area can be moved via pointer input device (PID). Striking over the widget from top to bottom or vice versa moves the data up or downwards. When releasing the PID during the data area is moving it slows down its motion and stops by snapping in a new item at the snap position. Further the data is shown in a loop. After the last data item it continues with the first item like in a chain. So the data can be 'rotated' like a wheel:

Description	LISTWHEEL widget
Application example showing three wheels for selecting a date. The example uses the owner draw mechanism to overlay the widget with a customized alpha mask for the shading effect.	

Table 19.563: LISTWHEEL description and screenshot

The table above shows a screenshot of the example `WIDGET_ListWheel.c` located in the example folder `Sample\Tutorial\` of the emWin package.

19.17.1 Configuration options

Type	Macro	Default	Description
S	`LISTWHEEL_FONT_DEFAULT`	GUI_Font13_1	Font used.
N	`LISTWHEEL_BKCOLOR0_DEFAULT`	GUI_WHITE	Background color of normal text.
N	`LISTWHEEL_BKCOLOR1_DEFAULT`	GUI_WHITE	Background color of selected text.
N	`LISTWHEEL_TEXTCOLOR0_DEFAULT`	GUI_BLACK	Text color of normal text.
N	`LISTWHEEL_TEXTCOLOR1_DEFAULT`	GUI_BLUE	Text color of selected text.
N	`LISTWHEEL_TEXTALIGN_DEFAULT`	GUI_TA_LEFT	Text alignment
N	`LISTWHEEL_DECELERATION_DEFAULT`	15	Deceleration value.
N	`LISTWHEEL_TIMER_PERIOD_DEFAULT`	25	Timer period

Table 19.564: Configuration options

19.17.2 Predefined IDs

The following symbols define IDs which may be used to make LISTWHEEL widgets distinguishable from creation: `GUI_ID_LISTWHEEL0` - `GUI_ID_LISTWHEEL3`

19.17.3 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget without releasing.
WM_NOTIFICATION_SEL_CHANGED	An item has been snapped at the snap position.

Table 19.565: Notification codes

19.17.4 Keyboard reaction

This widget currently does not react on keyboard input.

19.17.5 LISTWHEEL API

The table below lists the available emWin LISTWHEEL-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
LISTWHEEL_AddString()	Adds a new string.
LISTWHEEL_CreateEx()	Creates a LISTWHEEL widget.
LISTWHEEL_CreateIndirect()	Creates a LISTWHEEL widget from a resource table entry.
LISTWHEEL_CreateUser()	Creates a LISTWHEEL widget using extra bytes as user data.
LISTWHEEL_GetFont()	Returns the font used to draw the data.
LISTWHEEL_GetItemFromPos()	Returns the index of the item matching the given position.
LISTWHEEL_GetItemText()	Returns the text of the requested item.
LISTWHEEL_GetLBorder()	Returns the size in pixels of the left border.
LISTWHEEL_GetLineHeight()	Returns the height used for one item.
LISTWHEEL_GetNumItems()	Returns the number of data items.
LISTWHEEL_GetPos()	Returns the item index of the currently engaged item.
LISTWHEEL_GetRBorder()	Returns the size in pixels of the right border.
LISTWHEEL_GetSel()	Returns the currently selected item.
LISTWHEEL_GetSnapPosition()	Returns the snap position in pixels of the widget.
LISTWHEEL_GetTextAlign()	Returns the text alignment used to draw the data items.
LISTWHEEL_GetUserData()	Retrieves the data set with LISTWHEEL_SetUserData().
LISTWHEEL_IsMoving()	Returns if a LISTVIEW widget is moving or not.
LISTWHEEL_MoveToPos()	Moves the LISTWHEEL to the given position.
LISTWHEEL_OwnerDraw()	Default function for drawing the widget.
LISTWHEEL_SetBkColor()	Sets the color used for the background.
LISTWHEEL_SetDeceleration()	Sets the deceleration of the LISTWHEEL.
LISTWHEEL_SetFont()	Sets the font used to draw the item text.
LISTWHEEL_SetLBorder()	Sets the size in pixels of the left border.
LISTWHEEL_SetLineHeight()	Sets the height used for drawing one data item.
LISTWHEEL_SetOwnerDraw()	Sets a owner draw function for drawing the widget.
LISTWHEEL_SetPos()	Sets the LISTWHEEL to the given position.
LISTWHEEL_SetRBorder()	Sets the size in pixels of the right border.
LISTWHEEL_SetSel()	Sets the currently selected item.
LISTWHEEL_SetSnapPosition()	Sets the snap position in pixels from the top of the widget.
LISTWHEEL_SetText()	Sets the content of the widget.
LISTWHEEL_SetTextAlign()	Sets the alignment used to draw the data items.

Table 19.566: LISTWHEEL widget API list

Routine	Description
LISTWHEEL_SetTextColor()	Sets the color used to draw the data items.
LISTWHEEL_SetTimerPeriod()	Sets the timer period of the LISTWHEEL widget.
LISTWHEEL_SetUserData()	Sets the extra data of a LISTWHEEL widget.
LISTWHEEL_SetVelocity()	Starts moving the wheel with the given velocity.

Table 19.566: LISTWHEEL widget API list

LISTWHEEL_AddString()

Description

Adds a new data item (typically a string) to the widget.

Prototype

```
void LISTWHEEL_AddString(LISTWHEEL_Handle hObj, const char * s);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
s	Pointer to the string to be added.

Table 19.567: LISTWHEEL_AddString() parameter list

Additional information

The width of the given text should fit into the horizontal widget area. Otherwise the text will be clipped during the drawing operation.

LISTWHEEL_CreateEx()

Description

Creates a LISTWHEEL widget of a specified size at a specified location.

Prototype

```
LISTWHEEL_Handle LISTWHEEL_CreateEx(int      x0,       int y0,
                                    int      xSize,    int ySize,
                                    WM_HWIN hParent,  int WinFlags,
                                    int      ExFlags,  int Id,
                                    const GUI_ConstString * ppText);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new LISTVIEW widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to “WM_CreateWindow()” on page 404 for a list of available parameter values).
ExFlags	Not used, reserved for future use.
Id	Window ID of the widget.
ppText	Pointer to an array of string pointers containing the elements to be displayed.

Table 19.568: LISTWHEEL_CreateEx() parameter list

Return value

Handle of the created LISTWHEEL widget; 0 if the function fails.

Additional information

If the parameter ppText is used the last element of the array needs to be a NULL element.

Example

```
char * apText[] = {
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday",
  "Sunday",
  NULL
};

LISTWHEEL_CreateEx(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW,
                   0, GUI_ID_LISTWHEEL0, apText);
```

LISTWHEEL_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `WinFlags` of the function `LISTWHEEL_CreateEx()`.

LISTWHEEL_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `LISTWHEEL_CreateEx()` can be referred to.

LISTWHEEL_GetFont()

Description

Returns the font which is used to draw the data items of the given LISTWHEEL widget.

Prototype

```
const GUI_FONT * LISTWHEEL_GetFont(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.569: LISTWHEEL_GetFont() parameter list

Return value

Pointer to a `GUI_FONT` structure which is used to draw the data items.

LISTWHEEL_GetItemFromPos()

Description

Returns the index of the item matching the given position.

Prototype

```
int LISTWHEEL_GetItemFromPos(LISTWHEEL_Handle hObj, int yPos);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
yPos	Y-position of the LISTWHEEL widget.

Table 19.570: LISTWHEEL_GetItemFromPos() parameter list

Return value

Index of the item matching the given position.
-1, if an item index could not be determined.

LISTWHEEL_GetItemText()

Description

Returns the text of the requested data item.

Prototype

```
void LISTWHEEL_GetItemText(LISTWHEEL_Handle   hObj,    unsigned Index,
                           char             * pBuffer, int      MaxSize);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Index	Index of the requested item.
pBuffer	Buffer for storing the text.
MaxSize	Size in bytes of the buffer.

Table 19.571: LISTWHEEL_GetItemText() parameter list

Additional information

The function copies the text of the given item into the given buffer. If the size of the buffer is too small the text will be clipped.

LISTWHEEL_GetLBorder()

Description

Returns the size in pixels between the left border of the widget and the beginning of the text.

Prototype

```
int LISTWHEEL_GetLBorder(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.572: LISTWHEEL_GetLBorder() parameter list

Return value

Number of pixels between left border and text.

LISTWHEEL_GetLineHeight()

Description

Returns the height of one data item.

Prototype

```
unsigned LISTWHEEL_GetLineHeight(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.573: LISTWHEEL_GetLineHeight() parameter list

Return value

Height of one data item.

Additional information

This function returns the value set by the function `LISTWHEEL_SetLineHeight()`. A return value of zero means the height of one item depends on the size of the current font. For more details, refer to "LISTWHEEL_SetLineHeight()" on page 688,"LISTWHEEL_GetFont()" on page 681, and "GUI_GetYSizeOfFont()" on page 226.

LISTWHEEL_GetNumItems()

Description

Returns the number of data items of the given LISTWHEEL widget.

Prototype

```
int LISTWHEEL_GetNumItems(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.574: LISTWHEEL_GetNumItems() parameter list

Return value

Number of data items of the given LISTWHEEL widget.

LISTWHEEL_GetPos()

Description

Returns the zero based index of the item which is currently snapped in.

Prototype

```
int LISTWHEEL_GetPos(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.575: LISTWHEEL_GetPos() parameter list

Return value

Index of the item which is currently snapped in.

Additional information

The position at which the items being snapped can be set with the function `LISTWHEEL_SetSnapPosition()`. For more details, refer to "LISTWHEEL_SetSnapPosition()" on page 690.

LISTWHEEL_GetRBorder()

Description

Returns the size in pixels between the right border of the widget and the end of the text.

Prototype

```
int LISTWHEEL_GetRBorder(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.576: LISTWHEEL_GetRBorder() parameter list

Return value

Number of pixels between right border and text.

LISTWHEEL_GetSel()

Description

Returns the zero based index of the currently selected item.

Prototype

```
int LISTWHEEL_GetSel(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.577: LISTWHEEL_GetSel() parameter list

Return value

Index of the currently selected item.

Additional information

For more information, refer to "LISTWHEEL_SetSel()" on page 690.

LISTWHEEL_GetSnapPosition()

Description

Returns the position in pixels from the top of the LISTWHEEL widget at which the data items should be 'snapped in'.

Prototype

```
int LISTWHEEL_GetSnapPosition(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.578: LISTWHEEL_GetSnapPosition() parameter list

Return value

Snap position in pixels from the top edge of the LISTWHEEL widget.

Additional information

The default value is 0.

LISTWHEEL_GetTextAlign()

Description

Returns the text alignment of the given LISTWHEEL widget.

Prototype

```
int LISTWHEEL_GetTextAlign(LISTWHEEL_Handle hObj);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.579: LISTWHEEL_GetTextAlign() parameter list

Return value

Text alignment of the given LISTWHEEL widget.

Additional information

For more information, refer to "LISTWHEEL_SetTextAlign()" on page 692.

LISTWHEEL_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

LISTWHEEL_IsMoving()

Description

Returns the current state (moving or not moving) of the given LISTWHEEL widget.

Prototype

```
int LISTWHEEL_IsMoving(LISTWHEEL_Handle hObj)
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.580: LISTWHEEL_IsMoving() parameter list

Return value

0: not moving
1: moving

LISTWHEEL_MoveToPos()

Description

Moves the data area of the LISTWHEEL widget to the given position.

Prototype

```
void LISTWHEEL_MoveToPos(LISTWHEEL_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Index	Zero based index of the item to which the 'wheel' should move.

Table 19.581: LISTWHEEL_MoveToPos() parameter list

Additional information

The widget starts moving by choosing the shortest way. If for example 7 items are available and item 2 is currently snapped and the widget should move to the last item it begins moving backwards until the seventh item has been reached.
Detailed information on how to set a position can be found in the description of "LISTWHEEL_SetPos()" on page 689.

LISTWHEEL_OwnerDraw()

Description

Default function for managing drawing operations of one data item.

Prototype

```
int LISTWHEEL_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.

Table 19.582: LISTWHEEL_OwnerDraw() parameter list

Return value

Depends on the command in the `Cmd` element of the `WIDGET_ITEM_DRAW_INFO` structure pointed by `pDrawItemInfo`.

Additional information

This function is useful if `LISTWHEEL_SetOwnerDraw()` is used. It can be used to retrieve the original size of a data item and/or to draw the text of a data item and should be called for all commands which are not managed by the application defined owner draw function.
The following commands are managed by the default function:

- `WIDGET_ITEM_GET_XSIZE`
- `WIDGET_ITEM_GET_YSIZE`
- `WIDGET_ITEM_DRAW`

For more information, refer to "User drawn widgets" on page 460,"LISTWHEEL_SetOwnerDraw()" on page 688, and to the provided example.

LISTWHEEL_SetBkColor()

Before	After
	

Table 19.583: LISTWHEEL_SetBkColor() before after screenshots

Description

Sets the specified background color for selected and unselected items.

Prototype

```
void LISTWHEEL_SetBkColor(LISTWHEEL_Handle hObj,   unsigned int Index,
                          GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Index	See element list below.
Color	New background color.

Table 19.584: LISTWHEEL_SetBkColor() parameter list

Permitted values for element Index	
LISTWHEEL_CI_UNSEL	Changes the background color for all unselected items.
LISTWHEEL_CI_SEL	Changes the background color for the selected item.

LISTWHEEL_SetDeceleration()

Description

Sets the deceleration behavior of the LISTWHEEL. The higher the deceleration value, the less time it takes for the LISTWHEEL to stop moving.

Prototype

```
void LISTWHEEL_SetDeceleration(LISTWHEEL_Handle hObj,
                               unsigned int Deceleration)
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Deceleration	Deceleration value

Table 19.585: LISTWHEEL_SetDeceleration() parameter list

Additional information

The default value of the deceleration is 15. This can be change with the configuration option LISTWHEEL_DECELERATION_DEFAULT.

LISTWHEEL_SetFont()

Before	After
	

Table 19.586: LISTWHEEL_SetFont() before after screenshots

Description

Sets the font which should be used to draw the data items.

Prototype

```
void LISTWHEEL_SetFont(LISTWHEEL_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
pFont	Pointer to a GUI_FONT structure.

Table 19.587: LISTWHEEL_SetFont() parameter list

LISTWHEEL_SetLBorder()

Before	After
	

Table 19.588: LISTWHEEL_SetLBorder() before after screenshots

Description

Sets the border size between the left edge of the widget and the beginning of the text.

Prototype

```
void LISTWHEEL_SetLBorder(LISTWHEEL_Handle hObj, unsigned BorderSize);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
BorderSize	Desired border size.

Table 19.589: LISTWHEEL_SetLBorder() parameter list

Additional information

The default value of the border size is 0.

LISTWHEEL_SetLineHeight()

Before	After
	

Table 19.590: LISTWHEEL_SetLineHeight() before after screenshots

Description

Sets the line height used to draw a data item.

Prototype

```
void LISTWHEEL_SetLineHeight(LISTWHEEL_Handle hObj, unsigned LineHeight);
```

Parameter	Description
`hObj`	Handle of LISTWHEEL widget.
`LineHeight`	Desired height. Default is 0 which means the font size determines the height of a line.

Table 19.591: LISTWHEEL_SetLineHeight() parameter list

Additional information

Per default the height of a line depends on the used font. The value set by this function 'overwrites' this default behavior.

LISTWHEEL_SetOwnerDraw()

Before	After
	

Table 19.592: LISTWHEEL_SetOwnerDraw() before after screenshots

Description

Sets an application defined owner draw function for the widget which is responsible for drawing the widget items.

Prototype

```
void LISTWHEEL_SetOwnerDraw(LISTWHEEL_Handle         hObj,
                            WIDGET_DRAW_ITEM_FUNC * pfOwnerDraw);
```

Parameter	Description
`hObj`	Handle of LISTWHEEL widget.
`pfOwnerDraw`	Pointer to owner draw function.

Table 19.593: LISTWHEEL_SetOwnerDraw() parameter list

Additional information

This function sets a pointer to an application defined function which will be called by the widget when a data item has to be drawn or when the x or y size of a item is needed. It gives you the possibility to draw anything as data item, not just plain text. `pfDrawItem` is a pointer to an application-defined function of type `WIDGET_DRAW_ITEM_FUNC` which is explained at the beginning of the chapter.
The following commands are supported: WIDGET_ITEM_GET_YSIZE, WIDGET_ITEM_DRAW, WIDGET_DRAW_BACKGROUND and WIDGET_DRAW_OVERLAY.

Example

The following example routine draws 2 red indicator lines over the widget:

```
static int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_DRAW_OVERLAY:
    GUI_SetColor(GUI_RED);
    GUI_DrawHLine(40, 0, 99);
    GUI_DrawHLine(59, 0, 99);
    break;
  default:
    return LISTWHEEL_OwnerDraw(pDrawItemInfo);
  }
  return 0;
}
```

LISTWHEEL_SetPos()

Description

Sets the data area of the LISTWHEEL widget to the given position.

Prototype

```
void LISTWHEEL_SetPos(LISTWHEEL_Handle hObj, unsigned int Index);
```

Parameter	Description
`hObj`	Handle of the widget.
`Index`	Zero based index of the item to which the ’wheel’ should be set.

Table 19.594: LISTWHEEL_SetPos() parameter list

Additional information

Detailed information on how to move the LISTWHEEL to a position can be found in the description of “LISTWHEEL_MoveToPos()” on page 685.

LISTWHEEL_SetRBorder()

Before	After
	

Table 19.595: LISTWHEEL_SetRBorder() before after screenshots

Description

Sets the border size between the left edge of the widget and the beginning of the text.

Prototype

```
void LISTWHEEL_SetRBorder(LISTWHEEL_Handle hObj, unsigned BorderSize);
```

Parameter	Description
`hObj`	Handle of LISTWHEEL widget.
`BorderSize`	Desired border size.

Table 19.596: LISTWHEEL_SetRBorder() parameter list

Additional information

The default value of the border size is 0.

LISTWHEEL_SetSel()

Before	After
	

Table 19.597: LISTWHEEL_SetSel() before after screenshots

Description

The function sets the selected item.

Prototype

```
void LISTWHEEL_SetSel(LISTWHEEL_Handle hObj, int Sel);
```

Parameter	Description
`hObj`	Handle of LISTWHEEL widget.
`Sel`	Zero based index of item to be selected.

Table 19.598: LISTWHEEL_SetSel() parameter list

Additional information

Only one item can be selected. Per default the item with index 0 is selected.

LISTWHEEL_SetSnapPosition()

Before	After
	

Table 19.599: LISTWHEEL_SetSnapPosition() before after screenshots

Description

The function sets the relative position from the top of the widget at which the items should snap in. Per default the snap position is 0 which means the items are snapped in at the top of the widget.

Prototype

```
void LISTWHEEL_SetSnapPosition(LISTWHEEL_Handle hObj, int SnapPosition);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
SnapPosition	Relative position in pixels from the top of the widget at which the items should be snapped in.

Table 19.600: LISTWHEEL_SetSnapPosition() parameter list

Additional information

The function `LISTWHEEL_GetPos()` can be used to get the zero based index of the current item which has been snapped in.

LISTWHEEL_SetText()

Before	After
	

Table 19.601: LISTWHEEL_SetText() before after screenshots

Description

It removes any existing item and adds the given items passed by the function.

Prototype

```
void LISTWHEEL_SetText(LISTWHEEL_Handle         hObj,
                       const GUI_ConstString * ppText);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
ppText	Pointer to an array of strings. The last item needs to be a NULL pointer.

Table 19.602: LISTWHEEL_SetText() parameter list

Additional information

Note that the last element pointed to by `ppText` needs to be a NULL pointer.

Example

The following should show how the function should be used:

```
static char * _apText[] = {
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday",
  "Sunday",
  NULL
};

static void _SetContent(void) {
  LISTWHEEL_SetText(hWin, _apText);
}
```


LISTWHEEL_SetTextAlign()

Before	After
	

Table 19.603: LISTWHEEL_SetTextAlign() before after screenshots

Description

Sets the text alignment used to draw the items of the LISTWHEEL widget.

Prototype

```
void LISTWHEEL_SetTextAlign(LISTWHEEL_Handle hObj, int Align);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Align	Alignment to be used to draw the items of the widget.

Table 19.604: LISTWHEEL_SetTextAlign() parameter list

Additional information

For details about text alignment, refer to “GUI_SetTextAlign()” on page 105.

LISTWHEEL_SetTextColor()

Before	After
	

Table 19.605: LISTWHEEL_SetTextColor() before after screenshots

Description

Sets the color to be used to draw the text.

Prototype

```
void LISTWHEEL_SetTextColor(LISTWHEEL_Handle hObj,
                            unsigned int     Index, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Index	See table below.
Color	Color to be used.

Table 19.606: LISTWHEEL_SetTextColor() parameter list

Permitted values for parameter Index	
LISTWHEEL_CI_UNSEL	Sets the color of the not selected text.
LISTWHEEL_CI_SEL	Sets the color of the selected text.

LISTWHEEL_SetTimerPeriod()

Description

Sets the time interval after which the LISTWHEEL will be updated in milliseconds. By default this will be every 25ms.

Prototype

```
void LISTWHEEL_SetTimerPeriod(LISTWHEEL_Handle hObj,
                              GUI_TIMER_TIME   TimerPeriod);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
TimerPeriod	Timer period used for updating the LISTWHEEL

Table 19.607: LISTWHEEL_SetTimerPeriod() parameter list

Additional information

The default value of 25 can be set with the configuration option LISTWHEEL_TIMER_PERIOD.

LISTWHEEL_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

LISTWHEEL_SetVelocity()

Description

Starts moving the LISTWHEEL widget using the given velocity.

Prototype

```
void LISTWHEEL_SetVelocity(LISTWHEEL_Handle hObj, int Velocity);
```

Parameter	Description
hObj	Handle of LISTWHEEL widget.
Velocity	Starting speed.

Table 19.608: LISTWHEEL_SetVelocity() parameter list

Additional information

The velocity decreases automatically. The higher the given velocity the longer it takes for the movement to stop.

Example

This function is used in the sample applications MEMDEV_ListWheelEffects.c and WIDGET_ListWheel.c.

19.18 MENU: Menu widget

The MENU widget can be used to create several kinds of MENUs. Each MENU item represents an application command or a submenu. MENUs can be shown horizontally and/or vertically. Menu items can be grouped using separators. Separators are supported for horizontal and vertical MENUs. Selecting a MENU item sends a WM_MENU message to the owner of the MENU or opens a submenu. If mouse support is enabled the MENU widget reacts on moving the mouse over the items of a MENU widget.
The shipment of emWin contains an application example which shows how to use the MENU widget. It can be found under `Sample\Application\Reversi.c`.
The table below shows the appearance of a horizontal MENU widget with a vertical submenu:

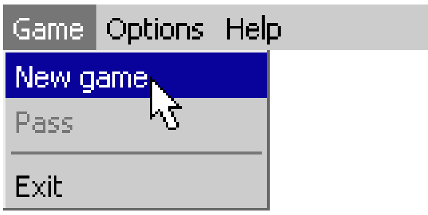
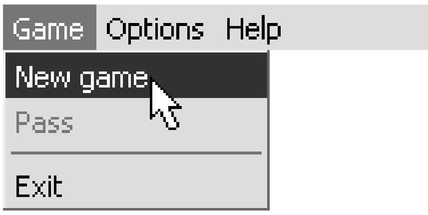
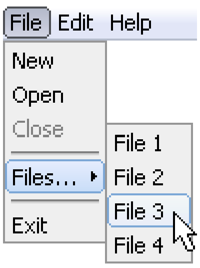
Description	Menu using WIDGET_Effect_3D1L	Menu using WIDGET_Effect_Simple
Color display (8666 mode)		
Monochrome display (16 gray scales)		
Black/white display		

Table 19.609: MENU appearance

Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter "Skinning" on page 871.

19.18.1 Menu messages

To inform its owner about selecting an item or opening a submenu the MENU widget sends a message of type `WM_MENU` to its owner.

WM_MENU

Description

This message is sent to inform the owner of a MENU about selecting an item or opening a submenu. Disabled MENU items will not send this message.

Data

The `Data.p` pointer of the message points to a `MENU_MSG_DATA` structure.

Elements of structure MENU_MSG_DATA

Data type	Element	Description
`U16`	`MsgType`	See table below.
`U16`	`ItemId`	Id of MENU item.

Table 19.610: MENU_MSG_DATA element list

Permitted values for element `MsgType`	
`MENU_ON_INITMENU`	This message is sent to the owner of MENU immediately before the MENU opens. This enables the application to modify the MENU before it is shown.
`MENU_ON_ITEMACTIVATE`	The owner window of a MENU will receive this message after a MENU item has been highlighted. The message is not sent after highlighting a submenu.
`MENU_ON_ITEMPRESSED`	After pressing a MENU item this message will be sent to the owner window of the widget. It will be sent also for disabled MENU items.
`MENU_ON_ITEMSELECT`	This message is sent to the owner of a MENU immediately after a MENU item is selected. The `ItemId` element contains the Id of the pressed MENU item.

Example

The following example shows how to react on a `WM_MENU` message:

```
void Callback(WM_MESSAGE * pMsg) {
  MENU_MSG_DATA * pData;
  WM_HWIN hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
  case WM_MENU:
    pData = (MENU_MSG_DATA *)pMsg->Data.p;
    switch (pData->MsgType) {
    case MENU_ON_ITEMACTIVATE:
      _UpdateStatusbar(pData->ItemId);
      break;
    case MENU_ON_INITMENU:
      _OnInitMenu();
      break;
    case MENU_ON_ITEMSELECT:
      switch (pData->ItemId) {
      case ID_MENU_ITEM0:
        ... /* React on selection of menu item 0 */
        break;
      case ID_MENU_ITEM1:
        ... /* React on selection of menu item 1 */
        break;
      case ...
        ...
      }
      break;
    }
    break;
  default:
    MENU_Callback(pMsg);
  }
}
```


19.18.2 Data structures

The following shows the MENU widget related data structures.

Elements of structure MENU_ITEM_DATA

This structure serves as a container to set or retrieve information about MENU items.

Data type	Element	Description
const char *	pText	MENU item text.
U16	Id	Id of the MENU item.
U16	Flags	See table below.
MENU_Handle	hSubmenu	If the item represents a submenu this element contains the handle of the submenu.

Table 19.611: MENU_ITEM_DATA element list

Permitted values for element Flags	
MENU_IF_DISABLED	Item is disabled.
MENU_IF_SEPARATOR	Item is a separator.

19.18.3 Configuration options

Type	Macro	Default	Description
N	MENU_BKCOLOR0_DEFAULT	GUI_LIGHTGRAY	Background color for enabled and unselected items.
N	MENU_BKCOLOR1_DEFAULT	0x980000	Background color for enabled and selected items.
N	MENU_BKCOLOR2_DEFAULT	GUI_LIGHTGRAY	Background color for disabled items.
N	MENU_BKCOLOR3_DEFAULT	0x980000	Background color for disabled and selected items.
N	MENU_BKCOLOR4_DEFAULT	0x7C7C7C	Background color for active sub-menu items.
N	MENU_BORDER_BOTTOM_DEFAULT	2	Border between item text and item bottom.
N	MENU_BORDER_LEFT_DEFAULT	4	Border between item text and left edge of item.
N	MENU_BORDER_RIGHT_DEFAULT	4	Border between item text and right edge of item.
N	MENU_BORDER_TOP_DEFAULT	2	Border between item text and item top.
S	MENU_EFFECT_DEFAULT	WIDGET_Effect_3D1L	Default effect.
S	MENU_FONT_DEFAULT	GUI_Font13_1	Font used.
N	MENU_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color for enabled and unse-lected items.
N	MENU_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color for enabled and selected items.
N	MENU_TEXTCOLOR2_DEFAULT	0x7C7C7C	Text color for disabled items.
N	MENU_TEXTCOLOR3_DEFAULT	GUI_LIGHTGRAY	Text color for disabled and selected items.
N	MENU_TEXTCOLOR4_DEFAULT	GUI_WHITE	Text color for active submenu items.

Table 19.612: Configuration options

19.18.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	- If the MENU is horizontal, the selection moves one item to the right. - If the MENU is vertical and the current item is a submenu, the submenu opens and the input focus moves to the submenu. - If the MENU is vertical and the current item is not a submenu and the top level MENU is horizontal, the next item of the top level MENU opens and the input focus moves to it.
GUI_KEY_LEFT	- If the MENU is horizontal the selection moves one item to the left. - If the MENU is vertical and the MENU is not the top level MENU, the current MENU closes and the focus moves to the previous MENU. If the previous MENU is horizontal the previous submenu of it opens and the focus moves to the previous submenu.
GUI_KEY_DOWN	- If the MENU is horizontal and the current MENU item is a submenu this submenu opens. - If the MENU is vertical, the selection moves to the next item.
GUI_KEY_UP	- If the MENU is vertical, the selection moves to the previous item.
GUI_KEY_ESCAPE	- If the MENU is not the top level MENU the current MENU will be closed and the focus moves to the previous MENU. - If the MENU is the top level MENU, the current MENU item becomes unselected.
GUI_KEY_ENTER	- If the current MENU item is a submenu, the submenu opens and the focus moves to the submenu. - If the current MENU item is not a submenu, all submenus of the top level MENU closes and a MENU_ON_ITEMSELECT message will be sent to the owner of the MENU.

Table 19.613: Keyboard reaction

19.18.5 MENU API

The table below lists the available emWin MENU-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
MENU_AddItem()	Adds an item to an existing MENU widget.
MENU_Attach()	Attaches a MENU widget with the given size at the given position to a specified window.
MENU_CreateEx()	Creates a MENU widget.
MENU_CreateIndirect()	Creates a MENU widget from a resource table entry.
MENU_CreateUser()	Creates a MENU widget using extra bytes as user data.
MENU_DeleteItem()	Deletes the specified MENU item.
MENU_DisableItem()	Disables the specified MENU item.
MENU_EnableItem()	Enables the specified MENU item.
MENU_GetDefaultBkColor()	Returns the default background color for new MENU widgets.
MENU_GetDefaultBorderSize()	Returns the default border size for new MENU widgets.
MENU_GetDefaultEffect()	Returns the default effect for new MENU widgets.
MENU_GetDefaultFont()	Returns a pointer to the default font used to display the item text of new MENU widget.
MENU_GetDefaultTextColor()	Returns the default text color for new MENU widget.
MENU_GetItem()	Retrieves information about the given MENU item.
MENU_GetItemText()	Returns the text of the given MENU item.
MENU_GetNumItems()	Returns the number of items of the given MENU widget.
MENU_GetOwner()	Returns the owner window of the given MENU widget.
MENU_GetUserData()	Retrieves the data set with MENU_SetUserData().
MENU_InsertItem()	Inserts a MENU item.
MENU_Popup()	Opens a popup MENU at the given position.
MENU_SetBkColor()	Sets the background color of the given MENU widget.

Table 19.614: MENU API list

Routine	Description
MENU_SetBorderSize()	Sets the border size of the given MENU widget.
MENU_SetDefaultBkColor()	Sets the default background color for new MENU widgets.
MENU_SetDefaultBorderSize()	Sets the default border size for new MENU widgets.
MENU_SetDefaultEffect()	Sets the default effect for new MENU widgets.
MENU_SetDefaultFont()	Sets a pointer to the default font used to display the item text of new MENU widget.
MENU_SetDefaultTextColor()	Sets the default text color for new MENU widgets.
MENU_SetFont()	Sets the font used to display the item text of the given MENU widget.
MENU_SetItem()	Changes the information about the given MENU item.
MENU_SetOwner()	Sets the WINDOW to be informed by the MENU widget.
MENU_SetTextColor()	Sets the text color of the given MENU widget.
MENU_SetUserData()	Sets the extra data of a MENU widget.

Table 19.614: MENU API list

MENU_AddItem()

Before	After
	

Table 19.615: MENU_AddItem() before after screenshots

Description

This function adds a new item to the end of the given MENU widget.

Prototype

```
void MENU_AddItem(MENU_Handle hObj, const MENU_ITEM_DATA * pItemData);
```

Parameter	Description
hObj	Handle of MENU widget.
pItemData	Pointer to a MENU_ITEM_DATA structure containing the information of the new item.

Table 19.616: MENU_AddItem() parameter list

Additional information

If using a MENU widget with several submenus the Id of the MENU items should be unique. Different submenus should not contain MENU items with the same IDs.
When adding items to a MENU widget and no fixed sizes are used the size of the MENU will be adapted.
Refer to “Elements of structure MENU_ITEM_DATA” on page 696.

MENU_Attach()

Description

Attaches the given MENU widget at the given position with the given size to a specified WINDOW.

Prototype

```
void MENU_Attach(MENU_Handle hObj,   WM_HWIN hDestWin,
                 int         x,      int     y,
                 int         xSize,  int     ySize,
                 int         Flags);
```

Parameter	Description
hObj	Handle of MENU widget.
hDestWin	Handle to the WINDOW to which the MENU widget should be attached.
x	X position in window coordinates of the MENU widget.
y	Y position in window coordinates of the MENU widget.
xSize	Fixed X size of the MENU. For details, refer to "MENU_CreateEx()" on page 699.
ySize	Fixed Y size of the MENU. For details, refer to "MENU_CreateEx()" on page 699.
Flags	Reserved for future use

Table 19.617: MENU_Attach() parameter list

Additional information

After creating a MENU widget this function can be used to attach the MENU widget to an existing window.

MENU_CreateEx()

Description

Creates a MENU widget of a specified size at a specified location.

Prototype

```
MENU_Handle MENU_CreateEx(int     x0,       int y0,
                          int     xSize,    int ySize,
                          WM_HWIN hParent,  int WinFlags,
                          int     ExFlags,  int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Fixed horizontal size of the widget (in pixels). 0 if MENU should handle the xSize.
ySize	Fixed vertical size of the widget (in pixels). 0 if MENU should handle the ySize.
hParent	Handle of parent window. If 0, the new widget will be a child of the desktop (top-level window). In some cases it can be useful to create the MENU widget in 'unattached' state and attach it later to an existing window. For this case WM_UNATTACHED can be used as parameter.
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to "WM_CreateWindow()" on page 404 for a list of available parameter values).
ExFlags	See table below.
Id	Window ID of the widget.

Table 19.618: MENU_CreateEx() parameter list

Permitted values for parameter ExFlags	
MENU_CF_HORIZONTAL	Creates a horizontal MENU.
MENU_CF_VERTICAL	Creates a vertical MENU.

Return value

Handle of the created MENU widget; 0 if the function fails.

Additional information

The parameters `xSize` and/or `ySize` specifies if a fixed width and/or height should be used for the MENU widget.
If these parameters are > 0, fixed sizes should be used. If for example the MENU should be attached as a horizontal MENU to the top of a WINDOW it can be necessary to use a fixed X size which covers the whole top of the window. In this case the parameter `xSize` can be used to set a fixed X size of the MENU. When attaching or deleting items of a MENU with a fixed size the size of the widget does not change.
If the values are 0, the MENU handles its size itself. That means the size of the MENU depends on the size of the current MENU items of a MENU widget. If items are added or removed the size of the widget will be adapted.

MENU_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `MENU_CreateEx()`.

MENU_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `MENU_CreateEx()` can be referred to.

MENU_DeleteItem()

Before	After
	

Table 19.619: MENU_DeleteItem() before after screenshots

Description

Deletes a given MENU item from a MENU widget.

Prototype

```
void MENU_DeleteItem(MENU_Handle hObj, U16 ItemId);
```

Parameter	Description
`hObj`	Handle of MENU widget.
`ItemId`	Id of the MENU item to be deleted.

Table 19.620: MENU_DeleteItem() parameter list

Additional information

If the item does not exist the function returns immediately.
When deleting items from a MENU widget and no fixed sizes are used the window size will be adapted.

MENU_DisableItem()

Before	After
	

Table 19.621: MENU_DisableItem() before after screenshots

Description

Disables the given MENU item.

Prototype

```
void MENU_DisableItem(MENU_Handle hObj, U16 ItemId);
```

Parameter	Description
`hObj`	Handle of MENU widget.
`ItemId`	Id of the MENU item to be disabled.

Table 19.622: MENU_DisableItem() parameter list

Additional information

If a disabled MENU item is selected, the MENU widget does not send the `WM_MENU` message to the owner. A disabled submenu item can not be opened.

MENU_EnableItem()

Before	After
	

Table 19.623: MENU_EnableItem() before after screenshots

Description

Enables the given MENU item.

Prototype

```
void MENU_EnableItem(MENU_Handle hObj, U16 ItemId);
```

Parameter	Description
`hObj`	Handle of MENU widget.
`ItemId`	Id of the MENU item to be enabled.

Table 19.624: MENU_EnableItem() parameter list

Additional information

For details, refer to “MENU_DisableItem()” on page 701.

MENU_GetDefaultBkColor()

Description

Returns the default background color used to draw new MENU items.

Prototype

```
GUI_COLOR MENU_GetDefaultBkColor(unsigned ColorIndex);
```

Parameter	Description
ColorIndex	Index of color to be returned. See table below.

Table 19.625: MENU_GetDefaultBkColor()

Permitted values for parameter ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Background color of active submenu items.
MENU_CI_DISABLED	Background color of disabled MENU items.
MENU_CI_DISABLED_SEL	Background color of disabled and selected MENU items.
MENU_CI_ENABLED	Background color of enabled and not selected MENU items.
MENU_CI_SELECTED	Background color of enabled and selected MENU items.

Return value

Default background color used to draw new MENU items.

Additional information

For details, refer to “MENU_SetBkColor()” on page 706.

MENU_GetDefaultBorderSize()

Description

Returns the default border size used for new MENU widgets.

Prototype

```
U8 MENU_GetDefaultBorderSize(unsigned BorderIndex);
```

Parameter	Description
BorderIndex	See table below.

Table 19.626: MENU_GetDefaultBorderSize() parameter list

Permitted values for parameter BorderIndex	
MENU_BI_BOTTOM	Border between item text and item bottom.
MENU_BI_LEFT	Border between item text and left edge of item.
MENU_BI_RIGHT	Border between item text and right edge of item
MENU_BI_TOP	Border between item text and item top.

Return value

Default border size used for new MENU widgets.

Additional information

For details, refer to “MENU_SetBorderSize()” on page 707.

MENU_GetDefaultEffect()

Description

Returns the default effect for new MENU widgets.

Prototype

```
const WIDGET_EFFECT * MENU_GetDefaultEffect(void);
```

Return value

The result of the function is a pointer to a `WIDGET_EFFECT` structure.

Additional information

For more information, refer to “WIDGET_SetDefaultEffect()” on page 459.

MENU_GetDefaultFont()

Description

Returns a pointer to the default font used to display the MENU item text of new MENU widgets.

Prototype

```
const GUI_FONT * MENU_GetDefaultFont(void);
```

Return value

Pointer to the default font used to display the MENU item text of new MENU widgets.

MENU_GetDefaultTextColor()

Description

Returns the default text color for new MENU widgets.

Prototype

```
GUI_COLOR MENU_GetDefaultTextColor(unsigned ColorIndex);
```

Parameter	Description
`ColorIndex`	Index of color to be returned. See table below.

Table 19.627: MENU_GetDefaultTextColor() parameter list

Permitted values for parameter `ColorIndex`	
`MENU_CI_ACTIVE_SUBMENU`	Text color of active submenu items.
`MENU_CI_DISABLED`	Text color of disabled MENU items.
`MENU_CI_DISABLED_SEL`	Text color of disabled and selected MENU items.
`MENU_CI_ENABLED`	Text color of enabled and not selected MENU items.
`MENU_CI_SELECTED`	Text color of enabled and selected MENU items.

Return value

Default text color for new MENU widgets.

Additional information

For details, refer to “MENU_SetDefaultTextColor()” on page 709.

MENU_GetItem()

Description

Retrieves information about the given MENU item.

Prototype

```
void MENU_GetItem(MENU_Handle hObj, U16 ItemId, MENU_ITEM_DATA * pItemData);
```

Parameter	Description
hObj	Handle of MENU widget.
ItemId	Id of the requested MENU item.
pItemData	Pointer to a MENU_ITEM_DATA structure to be filled by the function.

Table 19.628: MENU_GetItem() parameter list

Additional information

If using a MENU widget with several submenus the handle of the widget needs to be the handle of the menu/submenu containing the requested item or the handle of a higher menu/submenu.
The function sets the element `pText` of the `MENU_ITEM_INFO` data structure to 0. To retrieve the MENU item text the function `MENU_GetItemText()` should be used.
Refer to the beginning of the MENU chapter for details about the `MENU_ITEM_INFO` data structure.

MENU_GetItemText()

Description

Returns the text of the given MENU item.

Prototype

```
void MENU_GetItemText(MENU_Handle   hObj,     U16      ItemId,
                      char        * pBuffer, unsigned BufferSize);
```

Parameter	Description
hObj	Handle of MENU widget.
ItemId	Id of the requested MENU item.
pBuffer	Buffer to be filled by the function.
BufferSize	Maximum number of bytes to be retrieved.

Table 19.629: MENU_GetItemText() parameter list

MENU_GetNumItems()

Description

Returns the number of items of the given MENU widget.

Prototype

```
unsigned MENU_GetNumItems(MENU_Handle hObj);
```

Parameter	Description
hObj	Handle of MENU widget.

Table 19.630: MENU_GetNumItems() parameter list

Return value

Number of items of the given MENU widget.

MENU_GetOwner()

Description

Returns the owner WINDOW of the given MENU widget.

Prototype

```
WM_HWIN MENU_GetOwner(MENU_Handle hObj);
```

Parameter	Description
hObj	Handle of MENU widget.

Table 19.631: MENU_GetOwner() parameter list

Return value

Owner WINDOW of the given MENU widget.

MENU_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

MENU_InsertItem()

Before	After
	

Table 19.632: MENU_InsertItem() before after screenshots

Description

Inserts a MENU item at the given position.

Prototype

```
void MENU_InsertItem(MENU_Handle hObj, U16 ItemId,
                          const MENU_ITEM_DATA * pItemData);
```

Parameter	Description
hObj	Handle of MENU widget.
ItemId	Id of the MENU item the new item should be inserted before.
pItemData	Pointer to a MENU_ITEM_DATA structure containing the information of the new item.

Table 19.633: MENU_InsertItem() parameter list

Additional information

Refer to the beginning of the MENU chapter for details about the `MENU_ITEM_INFO` data structure.

MENU_Popup()

Description

Opens the given MENU at the given position. After selecting a MENU item or after touching the display outside the MENU the popup MENU will be closed.

Prototype

```
void MENU_Popup(MENU_Handle hObj,   WM_HWIN hDestWin,
                int         x,      int     y,
                int         xSize,  int     ySize,
```

```
int         Flags);
```

Parameter	Description
hObj	Handle of MENU widget.
hDestWin	Handle to the window to which the MENU should be attached.
x	X position in window coordinates of the MENU widget.
y	Y position in window coordinates of the MENU widget.
xSize	Fixed X size of the MENU. For details, refer to "MENU_CreateEx()" on page 699.
ySize	Fixed Y size of the MENU. For details, refer to "MENU_CreateEx()" on page 699.
Flags	Reserved for future use

Table 19.634: MENU_Popup() parameter list

Additional information

After selecting a MENU item or after touching the display outside the popup MENU the MENU will be closed. Note that the MENU will not be deleted automatically.
The `Sample` folder contains the example `WIDGET_PopupMenu.c` which shows how to use the function.

MENU_SetBkColor()

Before	After
	

Table 19.635: MENU_SetBkColor() before after screenshots

Description

Sets the background color of the given MENU widget.

Prototype

```
void MENU_SetBkColor(MENU_Handle hObj,   unsigned ColorIndex,
                     GUI_COLOR   Color);
```

Parameter	Description
hObj	Handle of MENU widget.
ColorIndex	Index of color. See table below.
Color	Color to be used.

Table 19.636: MENU_SetBkColor() parameter list

Permitted values for parameter ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Background color of active submenu items.
MENU_CI_DISABLED	Background color of disabled MENU items.
MENU_CI_DISABLED_SEL	Background color of disabled and selected MENU items.
MENU_CI_ENABLED	Background color of enabled and not selected MENU items.
MENU_CI_SELECTED	Background color of enabled and selected MENU items.

MENU_SetBorderSize()

Before	After
	

Table 19.637: MENU_SetBorderSize() before after screenshots with vertical MENU

The following code is executed between the screenshots above:

```
MENU_SetBorderSize(hMenuGame, MENU_BI_LEFT, 20);
```

Before	After
	

Table 19.638: MENU_SetBorderSize() before after screenshots with horizontal MENU

The following code is executed between the screenshots above:

```
MENU_SetBorderSize(hMenu, MENU_BI_LEFT, 10);
MENU_SetBorderSize(hMenu, MENU_BI_RIGHT, 10);
```

Description

Sets the border size of the given MENU widget.

Prototype

```
void MENU_SetBorderSize(MENU_Handle hObj,        unsigned BorderIndex,
                        U8          BorderSize);
```

Parameter	Description
hObj	Handle of MENU widget.
BorderIndex	See table below.
BorderSize	Size to be used.

Table 19.639: MENU_SetBorderSize() parameter list

Permitted values for parameter BorderIndex	
MENU_BI_BOTTOM	Border between item text and item bottom.
MENU_BI_LEFT	Border between item text and left edge of item.
MENU_BI_RIGHT	Border between item text and right edge of item
MENU_BI_TOP	Border between item text and item top.

MENU_SetDefaultBkColor()

Description

Sets the default background color used to draw new MENU items.

Prototype

```
void MENU_SetDefaultBkColor(unsigned ColorIndex, GUI_COLOR Color);
```

Parameter	Description
ColorIndex	Index of color to be returned. See table below.
Color	Color to be used.

Table 19.640: MENU_SetDefaultBkColor() parameter list

Permitted values for parameter ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Background color of active submenu items.
MENU_CI_DISABLED	Background color of disabled MENU items.
MENU_CI_DISABLED_SEL	Background color of disabled and selected MENU items.
MENU_CI_ENABLED	Background color of enabled and not selected MENU items.
MENU_CI_SELECTED	Background color of enabled and selected MENU items.

Additional information

For details, refer to “MENU_SetBkColor()” on page 706.

MENU_SetDefaultBorderSize()

Description

Sets the default border size used for new MENU widgets.

Prototype

```
void MENU_SetDefaultBorderSize(unsigned BorderIndex, U8 BorderSize);
```

Parameter	Description
BorderIndex	See table below.
BorderSize	Border size to be used.

Table 19.641: MENU_SetDefaultBorderSize() parameter list

Permitted values for parameter BorderIndex	
MENU_BI_BOTTOM	Border between item text and item bottom.
MENU_BI_LEFT	Border between item text and left edge of item.
MENU_BI_RIGHT	Border between item text and right edge of item
MENU_BI_TOP	Border between item text and item top.

Additional information

For details, refer to “MENU_SetBorderSize()” on page 707.

MENU_SetDefaultEffect()

Description

Sets the default effect for new MENU widgets.

Prototype

```
void MENU_SetDefaultEffect(const WIDGET_EFFECT * pEffect);
```

Parameter	Description
pEffect	Pointer to a WIDGET_EFFECT structure.

Table 19.642: MENU_SetDefaultEffect() parameter list

Additional information

For more information, refer to “WIDGET_SetDefaultEffect()” on page 459.

MENU_SetDefaultFont()

Description

Sets the pointer to the default font used to display the MENU item text of new MENU widgets.

Prototype

```
void MENU_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
`pFont`	Pointer to the `GUI_FONT` structure to be used.

Table 19.643: MENU_SetDefaultFont() parameter list

Additional information

For details, refer to “MENU_SetFont()” on page 709.

MENU_SetDefaultTextColor()

Description

Sets the default text color for new MENU widgets.

Prototype

```
void MENU_SetDefaultTextColor(unsigned ColorIndex, GUI_COLOR Color);
```

Parameter	Description
`ColorIndex`	Index of color to be used. See table below.
`Color`	Color to be used

Table 19.644: MENU_SetDefaultTextColor() parameter list

Permitted values for parameter `ColorIndex`	
`MENU_CI_ACTIVE_SUBMENU`	Text color of active submenu items.
`MENU_CI_DISABLED`	Text color of disabled MENU items.
`MENU_CI_DISABLED_SEL`	Text color of disabled and selected MENU items.
`MENU_CI_ENABLED`	Text color of enabled and not selected MENU items.
`MENU_CI_SELECTED`	Text color of enabled and selected MENU items.

Additional information

For details, refer to “MENU_SetTextColor()” on page 711.

MENU_SetFont()

Before	After
	

Table 19.645: MENU_SetFont() before after screenshots

Description

Sets the pointer to the font used to display the item text of the MENU widget.

Prototype

```
void MENU_SetFont(MENU_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of MENU widget.
pFont	Pointer to the GUI_FONT structure to be used.

Table 19.646: MENU_SetFont() parameter list

MENU_SetItem()

Before	After
	

Table 19.647: MENU_SetItem() before after screenshots

Description

Sets the item information for the given MENU item.

Prototype

```
void MENU_SetItem(MENU_Handle             hObj,       U16 ItemId,
                  const MENU_ITEM_DATA * pItemData);
```

Parameter	Description
hObj	Handle of MENU widget.
ItemId	Id of the MENU item to be changed.
pItemData	Pointer to a MENU_ITEM_DATA structure containing the new information.

Table 19.648: MENU_SetItem() parameter list

MENU_SetOwner()

Description

Sets the owner WINDOW of the MENU widget that will be informed with WM_MENU messages.

Prototype

```
void MENU_SetOwner(MENU_Handle hObj, WM_HWIN hOwner);
```

Parameter	Description
hObj	Handle of MENU widget.
hOwner	Handle of the owner WINDOW which should receive the WM_MENU messages of the MENU widget.

Table 19.649: MENU_SetOwner() parameter list

Additional information

If no owner is set the parent WINDOW of the MENU widget will receive `WM_MENU` messages. In case the `WM_MENU` messages are not intended to be sent to the parent WINDOW, this function can be used to set another recipient for the messages.

MENU_SetSel()

Before	After
	

Table 19.650: MENU_SetSel() before after screenshots

Description

Sets the selected item of the given MENU widget.

Prototype

```
void MENU_SetSel(MENU_Handle hObj, int Sel);
```

Parameter	Description
`hObj`	Handle of MENU widget.
`Sel`	Zero based index of MENU item to be selected.

Table 19.651: MENU_SetSel() parameter list

Return value

The function returns the zero based index of the previous selected MENU item.

Additional information

A value <0 for parameter `Sel` deselects the MENU items.

MENU_SetTextColor()

Before	After
	

Table 19.652: MENU_SetTextColor() before after screenshots

Description

Sets the text color of the given MENU widget.

Prototype

```
void MENU_SetTextColor(MENU_Handle hObj,   unsigned ColorIndex,
                       GUI_COLOR    Color);
```

Parameter	Description
`hObj`	Handle of MENU widget.
`ColorIndex`	Index of color to be used. See table below.
`Color`	Color to be used.

Table 19.653: MENU_SetTextColor() parameter list

Permitted values for parameter ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Text color of active submenu items.
MENU_CI_DISABLED	Text color of disabled MENU items.
MENU_CI_DISABLED_SEL	Text color of disabled and selected MENU items.
MENU_CI_ENABLED	Text color of enabled and not selected MENU items.
MENU_CI_SELECTED	Text color of enabled and selected MENU items.

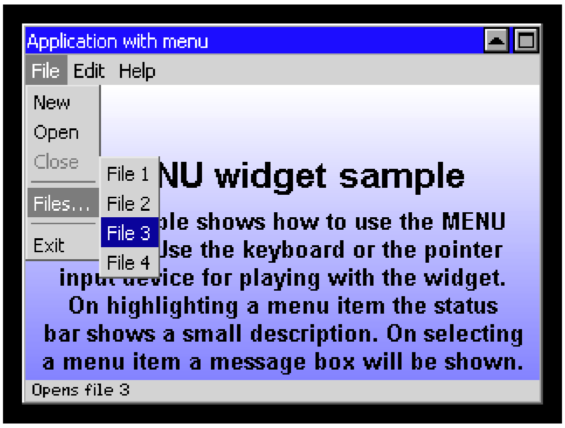
MENU_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

19.18.6 Example

The Sample folder contains the sample WIDGET_Menu.c which shows how the widget can be used. Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Menu.c:



19.19 MULTIEDIT: Multi line text widget

The MULTIEDIT widget enables you to edit text with multiple lines. You can use it as a simple text editor or to display static text. The widget supports scrolling with and without scroll bars. All MULTIEDIT-related routines are in the file(s) MULTIEDIT*.c, MULTIEDIT.h. All identifiers are prefixed MULTIEDIT. The table below shows the appearance of the MULTIEDIT widget:

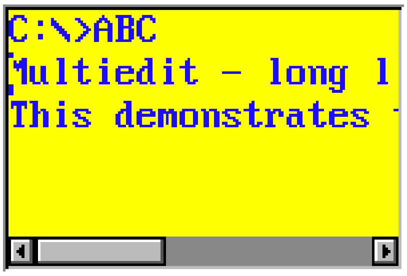
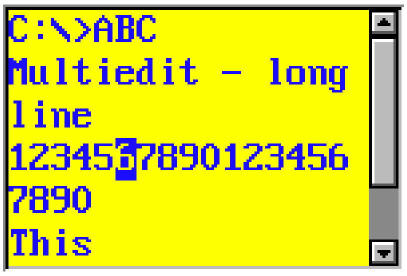
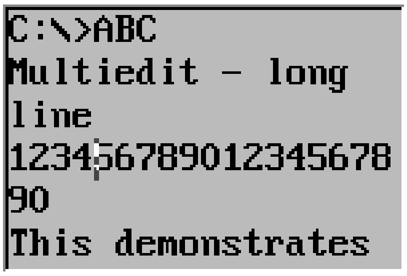
Description	Frame window
edit mode, automatic horizontal scroll bar, non wrapping mode, insert mode,	
edit mode, automatic vertical scroll bar, word wrapping mode, overwrite mode,	
read only mode, word wrapping mode	

Table 19.654: MULTIEDIT appearance

19.19.1 Configuration options

Type	Macro	Default	Description
S	MULTIEDIT_FONT_DEFAULT	GUI_Font13_1	Font used.
N	MULTIEDIT_BKCOLOR0_DEFAULT	GUI_WHITE	Background color.
N	MULTIEDIT_BKCOLOR2_DEFAULT	0xC0C0C0	Background color read only mode.
N	MULTIEDIT_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color.
N	MULTIEDIT_TEXTCOLOR2_DEFAULT	GUI_BLACK	Text color read only mode.

Table 19.655: Configuration options

19.19.2 Predefined IDs

The following symbols define IDs which may be used to make MULTIEDIT widgets distinguishable from creation: GUI_ID_MULTIEDIT0 - GUI_ID_MULTIEDIT3

19.19.3 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scroll bar has been changed.
WM_NOTIFICATION_VALUE_CHANGED	The text of the widget has been changed.

Table 19.656: Notification codes

19.19.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_UP	Moves the cursor one line up.
GUI_KEY_DOWN	Moves the cursor one line down.
GUI_KEY_RIGHT	Moves the cursor one character to the right.
GUI_KEY_LEFT	Moves the cursor one character to the left.
GUI_KEY_END	Moves the cursor to the end of the current row.
GUI_KEY_HOME	Moves the cursor to the begin of the current row.
GUI_KEY_BACKSPACE	If the widget works in read/write mode this key deletes the character before the cursor.
GUI_KEY_DELETE	If the widget works in read/write mode this key deletes the character below the cursor.
GUI_KEY_INSERT	Toggles between insert and overwrite mode.
GUI_KEY_ENTER	If the widget works in read/write mode this key inserts a new line ('\n') at the current position. If the widget works in read only mode the cursor will be moved to the beginning of the next line.

Table 19.657: Keyboard reaction

19.19.5 MULTIEDIT API

The table below lists the available emWin MULTIEDIT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
MULTIEDIT_AddKey()	Key input routine.
MULTIEDIT_AddText()	Adds additional text at the current cursor position.
MULTIEDIT_Create()	Creates a MULTIEDIT widget. (Obsolete)
MULTIEDIT_CreateEx()	Creates a MULTIEDIT widget.
MULTIEDIT_CreateIndirect()	Creates a MULTIEDIT widget from a resource table entry.
MULTIEDIT_CreateUser()	Creates a MULTIEDIT widget using extra bytes as user data.
MULTIEDIT_EnableBlink()	Enables/disables a blinking cursor.
MULTIEDIT_GetCursorCharPos()	Returns the number of the character at the cursor position.
MULTIEDIT_GetCursorPixelPos()	Returns the pixel position of the cursor.

Table 19.658: MULTIEDIT API list

Routine	Description
MULTIEDIT_GetPrompt()	Returns the text of the prompt.
MULTIEDIT_GetText()	Returns the text.
MULTIEDIT_GetTextSize()	Returns the buffer size used by the current text.
MULTIEDIT_GetUserData()	Retrieves the data set with MULTIEDIT_SetUserData().
MULTIEDIT_SetAutoScrollH()	Activates automatic use of a horizontal scroll bar.
MULTIEDIT_SetAutoScrollV()	Activates automatic use of a vertical scroll bar.
MULTIEDIT_SetBkColor()	Sets the background color.
MULTIEDIT_SetBufferSize()	Sets the buffer size used for text and prompt.
MULTIEDIT_SetCursorOffset()	Sets the cursor to the given character.
MULTIEDIT_SetFocussable()	Sets the focussability of the given MULTIEDIT widget.
MULTIEDIT_SetFont()	Sets the font.
MULTIEDIT_SetInsertMode()	Enables/disables the insert mode.
MULTIEDIT_SetMaxNumChars()	Sets the maximum number of characters including the prompt.
MULTIEDIT_SetPasswordMode()	Enables/disables password mode.
MULTIEDIT_SetPrompt()	Sets the prompt text.
MULTIEDIT_SetReadOnly()	Enables/disables the read only mode.
MULTIEDIT_SetText()	Sets the text.
MULTIEDIT_SetTextAlign()	Sets the text alignment.
MULTIEDIT_SetTextColor()	Sets the text color,
MULTIEDIT_SetUserData()	Sets the extra data of a MULTIEDIT widget.
MULTIEDIT_SetWrapWord()	Enables/disables word wrapping.
MULTIEDIT_SetWrapNone()	Enables/disables the non wrapping mode.

Table 19.658: MULTIEDIT API list

MULTIEDIT_AddKey()

Description

Adds user input to a specified MULTIEDIT widget.

Prototype

```
void MULTIEDIT_AddKey(MULTIEDIT_HANDLE hObj, int Key);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
Key	Character to be added.

Table 19.659: MULTIEDIT_AddKey() parameter list

Additional information

The specified character is added to the user input of the MULTIEDIT widget. If the maximum count of characters has been reached, another character will not be added.

MULTIEDIT_AddText()

Description

Adds the given text at the current cursor position.

Prototype

```
int MULTIEDIT_AddText(MULTIEDIT_HANDLE hObj, const char * s);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
s	Pointer to a NULL terminated text to be added.

Table 19.660: MULTIEDIT_AddText() parameter list

Additional information

If the number of characters exceeds the limit set with the function MULTIEDIT_SetMaxNumChars() the function will add only the characters of the text which fit into the widget respecting the limit.

MULTIEDIT_Create()

(Obsolete, MULTIEDIT_CreateEx() should be used instead)

Description

Creates a MULTIEDIT widget of a specified size at a specified location.

Prototype

```
MULTIEDIT_HANDLE MULTIEDIT_Create(int          x0,      int y0,
                                  int          xSize,   int ySize,
                                  WM_HWIN      hParent, int Id,
                                  int          Flags,   int ExFlags,
                                  const char * pText,   int MaxLen);
```

Parameter	Description
x0	Leftmost pixel of the MULTIEDIT widget (in parent coordinates).
y0	Topmost pixel of the MULTIEDIT widget (in parent coordinates).
xSize	Horizontal size of the MULTIEDIT widget (in pixels).
ySize	Vertical size of the MULTIEDIT widget (in pixels).
hParent	Parent window of the MULTIEDIT widget.
Id	ID of the MULTIEDIT widget.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	See table below.
pText	Text to be used.
MaxLen	Maximum number of bytes for text and prompt.

Table 19.661: MULTIEDIT_Create() parameter list

Permitted values for parameter ExFlags	
MULTIEDIT_CF_AUTOSCROLLBAR_H	Automatic use of a horizontal scroll bar.
MULTIEDIT_CF_AUTOSCROLLBAR_V	Automatic use of a vertical scroll bar.
MULTIEDIT_CF_INSERT	Enables insert mode.
MULTIEDIT_CF_READONLY	Enables read only mode.

Return value

Handle of the created MULTIEDIT widget; 0 if the function fails.

MULTIEDIT_CreateEx()

Description

Creates a MULTIEDIT widget of a specified size at a specified location.

Prototype

```
MULTIEDIT_HANDLE MULTIEDIT_CreateEx(int          x0,        int y0,
                                    int          xSize,     int ySize,
                                    WM_HWIN      hParent,   int WinFlags,
                                    int          ExFlags,   int Id,
                                    int          BufferSize,
                                    const char * pText);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new MULTIEDIT widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter “The Window Manager (WM)” on page 375 for a list of available parameter values).
ExFlags	See table below.
Id	Window ID of the widget.
BufferSize	Initial text buffer size of the widget. Use MULTIEDIT_SetMaxNumChars to set the maximum number of characters.
pText	Text to be used.

Table 19.662: MULTIEDIT_CreateEx() parameter list

Permitted values for parameter ExFlags	
MULTIEDIT_CF_AUTOSCROLLBAR_H	Automatic use of a horizontal scroll bar.
MULTIEDIT_CF_AUTOSCROLLBAR_V	Automatic use of a vertical scroll bar.
MULTIEDIT_CF_INSERT	Enables insert mode.
MULTIEDIT_CF_READONLY	Enables read only mode.

Return value

Handle of the created MULTIEDIT widget; 0 if the function fails.

MULTIEDIT_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The element Para of the according GUI_WIDGET_CREATE_INFO structure is used according to the parameter BufferSize of the function MULTIEDIT_CreateEx(). The element Flags is used according to the parameter ExFlags of the function MULTIEDIT_CreateEx().

MULTIEDIT_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function MULTIEDIT_CreateEx() can be referred to.

MULTIEDIT_EnableBlink()

Description

Enables/disables a blinking cursor.

Prototype

```
void MULTIEDIT_EnableBlink(MULTIEDIT_Handle hObj, int Period, int OnOff);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
Period	Blinking period
OnOff	1 enables blinking, 0 disables blinking

Table 19.663: MULTIEDIT_EnableBlink() parameter list

Additional information

This function calls `GUI_X_GetTime()`.

MULTIEDIT_GetCursorCharPos()

Description

Returns the number of the character at the cursor position.

Prototype

```
int MULTIEDIT_GetCursorCharPos(MULTIEDIT_Handle hObj);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.

Table 19.664: MULTIEDIT_GetCursorCharPos() parameter list

Return value

Number of the character at the cursor position.

Additional information

The widget returns the character position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

MULTIEDIT_GetCursorPixelPos()

Description

Returns the pixel position of the cursor in window coordinates.

Prototype

```
void MULTIEDIT_GetCursorPixelPos(MULTIEDIT_Handle   hObj,
                                 int              * pxPos, int * pyPos);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
pxPos	Pointer to integer variable for the X-position in window coordinates.
pyPos	Pointer to integer variable for the Y-position in window coordinates.

Table 19.665: MULTIEDIT_GetCursorPixelPos() parameter list

Additional information

The widget returns the pixel position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

MULTIEDIT_GetPrompt()

Description

Returns the current prompt text.

Prototype

```
void MULTIEDIT_GetPrompt(MULTIEDIT_HANDLE hObj, char * sDest, int MaxLen);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
sDest	Buffer for the prompt text to be returned.
MaxLen	Maximum number of bytes to be copied to sDest.

Table 19.666: MULTIEDIT_GetPrompt() parameter list

Additional information

The function copies the current prompt text to the buffer given by sDest. The maximum number of bytes copied to the buffer is given by MaxLen.

MULTIEDIT_GetText()

Description

Returns the current text.

Prototype

```
void MULTIEDIT_GetText(MULTIEDIT_HANDLE hObj, char * sDest, int MaxLen);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
sDest	Buffer for the text to be returned.
MaxLen	Maximum number of bytes to be copied to sDest.

Table 19.667: MULTIEDIT_GetText() parameter list

Additional information

The function copies the current text to the buffer given by sDest. The maximum number of bytes copied to the buffer is given by MaxLen.

MULTIEDIT_GetTextSize()

Description

Returns the buffer size used to store the current text (and prompt).

Prototype

```
int MULTIEDIT_GetTextSize(MULTIEDIT_HANDLE hObj);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.

Table 19.668: MULTIEDIT_GetTextSize() parameter list

Return value

Buffer size used to store the current text (and prompt).

MULTIEDIT_GetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_GetUserData().

MULTIEDIT_SetAutoScrollH()

Description

Enables/disables the automatic use of a horizontal scroll bar.

Prototype

```
void MULTIEDIT_SetAutoScrollH(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
OnOff	See table below.

Table 19.669: MULTIEDIT_SetAutoScrollH() parameter list

Permitted values for parameter OnOff	
0	Disables automatic use of a horizontal scroll bar.
1	Enables automatic use of a horizontal scroll bar.

Additional information

Enabling the use of a automatic horizontal scroll bar only makes sense with the non wrapping mode explained later in this chapter. If enabled the MULTIEDIT widget checks if the width of the non wrapped text fits into the client area. If not a horizontal scroll bar will be attached to the window.

MULTIEDIT_SetAutoScrollV()

Description

Enables/disables the automatic use of a vertical scroll bar.

Prototype

```
void MULTIEDIT_SetAutoScrollV(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
OnOff	See table below.

Table 19.670: MULTIEDIT_SetAutoScrollV() parameter list

Permitted values for parameter OnOff	
0	Disables automatic use of a vertical scroll bar.
1	Enables automatic use of a vertical scroll bar.

Additional information

If enabled the MULTIEDIT widget checks if the height of the text fits into the client area. If not a vertical scroll bar will be attached to the window.

MULTIEDIT_SetBkColor()

Description

Sets the background color of the given MULTIEDIT widget.

Prototype

```
void MULTIEDIT_SetBkColor(MULTIEDIT_HANDLE hObj,   unsigned int Index,
                          GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
Index	See table below.
Color	Background color to be used.

Table 19.671: MULTIEDIT_SetBkColor() parameter list

Permitted values for parameter `Index`	
`MULTIEDIT_CI_EDIT`	Edit mode.
`MULTIEDIT_CI_READONLY`	Read only mode.

MULTIEDIT_SetBufferSize()

Description

Sets the maximum number of bytes used by text and prompt.

Prototype

```
void MULTIEDIT_SetBufferSize(MULTIEDIT_HANDLE hObj, int BufferSize);
```

Parameter	Description
`hObj`	Handle of MULTIEDIT widget.
`BufferSize`	Maximum number of bytes.

Table 19.672: MULTIEDIT_SetBufferSize() parameter list

Additional information

The function clears the current content of the MULTIEDIT widget and allocates the given number of bytes for the text and for the prompt.

MULTIEDIT_SetCursorOffset()

Description

Sets the cursor position to the given character.

Prototype

```
void MULTIEDIT_SetCursorOffset(MULTIEDIT_HANDLE hObj, int Offset);
```

Parameter	Description
`hObj`	Handle of MULTIEDIT widget.
`Offset`	New cursor position.

Table 19.673: MULTIEDIT_SetCursorOffset() parameter list

Additional information

The number of characters used for the prompt has to be added to the parameter Offset. If a prompt is used the value for parameter Offset should not be smaller than the number of characters used for the prompt.

MULTIEDIT_SetFocussable()

Description

Sets the focussability of the given MULTIEDIT widget.

Prototype

```
void MULTIEDIT_SetFocussable(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Description
`hObj`	Handle of MULTIEDIT widget.
`OnOff`	See table below.

Table 19.674: MULTIEDIT_SetFocussable() parameter list

Permitted values for parameter `OnOff`	
0	Disables focussability.
1	Enables focussability.

Additional information

The text can not be aligned to the center if the widget is focusable. To change text alignment, the function `MULTIEDIT_SetTextAlign()` (page 724) can be used.

MULTIEDIT_SetFont()

Description

Sets the font used to display the text and the prompt.

Prototype

```
void MULTIEDIT_SetFont(MULTIEDIT_HANDLE hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
pFont	Pointer to font to be used.

Table 19.675: MULTIEDIT_SetFont() parameter list

MULTIEDIT_SetInsertMode()

Description

Enables/disables the insert mode. The default behaviour is overwrite mode.

Prototype

```
void MULTIEDIT_SetInsertMode(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
OnOff	See table below.

Table 19.676: MULTIEDIT_SetInsertMode() parameter list

Permitted values for parameter OnOff	
0	Disables insert mode.
1	Enables insert mode.

MULTIEDIT_SetMaxNumChars()

Description

Sets the maximum number of characters used by text and prompt.

Prototype

```
void MULTIEDIT_SetMaxNumChars(MULTIEDIT_HANDLE hObj, unsigned MaxNumChars);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
MaxNumChars	Maximum number of characters.

Table 19.677: MULTIEDIT_SetMaxNumChars() parameter list

MULTIEDIT_SetPasswordMode()

Description

Enables/disables the password mode.

Prototype

```
void MULTIEDIT_SetPasswordMode(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
OnOff	See table below.

Table 19.678: MULTIEDIT_SetPasswordMode() parameter list

Permitted values for parameter OnOff	
0	Disables password mode.
1	Enables password mode.

Additional information

The password mode enables you to conceal the user input.

MULTIEDIT_SetPrompt()

Description

Sets the prompt text.

Prototype

```
void MULTIEDIT_SetPrompt(MULTIEDIT_HANDLE hObj, const char * sPrompt);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
sPrompt	Pointer to the new prompt text.

Table 19.679: MULTIEDIT_SetPrompt() parameter list

Additional information

The prompt text is displayed first. The cursor can not be moved into the prompt.

MULTIEDIT_SetReadOnly()

Description

Enables/disables the read only mode.

Prototype

```
void MULTIEDIT_SetReadOnly(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
OnOff	See table below.

Table 19.680: MULTIEDIT_SetReadOnly() parameter list

Permitted values for parameter OnOff	
0	Disables read only mode.
1	Enables read only mode.

Additional information

If the read only mode has been set the widget does not change the text. Only the cursor will be moved.

MULTIEDIT_SetText()

Description

Sets the text to be handled by the MULTIEDIT widget.

Prototype

```
void MULTIEDIT_SetText(MULTIEDIT_HANDLE hObj, const char * s);
```

Parameter	Description
hObj	Handle of the MULTIEDIT widget.
s	Pointer to the text to be handled by the MULTIEDIT widget.

Table 19.681: MULTIEDIT_SetText() parameter list

Additional information

The function copies the given text to the buffer allocated when creating the widget or by MULTIEDIT_SetMaxSize(). The current text can be retrieved by MULTIEDIT_GetText().

MULTIEDIT_SetTextAlign()

Description

Sets the text alignment for the given MULTIEDIT widget.

Prototype

```
void MULTIEDIT_SetTextAlign(MULTIEDIT_HANDLE hObj, int Align);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
Align	See table below.

Table 19.682: MULTIEDIT_SetTextAlign() parameter list

Permitted values for parameter Align	
GUI_TA_LEFT	Left text align.
GUI_TA_HCENTER	Horizontally centered text align.
GUI_TA_RIGHT	Right text align.

Additional information

The text can not be horizontally centered if the widget is focusable. To change focussability of the widget, the function MULTIEDIT_SetFocussable() (page 721) can be used.

MULTIEDIT_SetTextColor()

Description

Sets the text color.

Prototype

```
void MULTIEDIT_SetTextColor(MULTIEDIT_HANDLE hObj,   unsigned int Index,
                            GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.
Index	See table below.
Color	Text color to be used.

Table 19.683: MULTIEDIT_SetTextColor() parameter list

Permitted values for parameter Index	
MULTIEDIT_CI_EDIT	Edit mode.
MULTIEDIT_CI_READONLY	Read only mode.

MULTIEDIT_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

MULTIEDIT_SetWrapWord()

Description

Enables the word wrapping mode.

Prototype

```
void MULTIEDIT_SetWrapWord(MULTIEDIT_HANDLE hObj);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.

Table 19.684: MULTIEDIT_SetWrapWord() parameter list

Additional information

If the word wrapping mode has been set the text at the end of a line will be wrapped at the beginning of the last word (if possible).

MULTIEDIT_SetWrapNone()

Description

Enables the non wrapping mode.

Prototype

```
void MULTIEDIT_SetWrapNone(MULTIEDIT_HANDLE hObj);
```

Parameter	Description
hObj	Handle of MULTIEDIT widget.

Table 19.685: MULTIEDIT_SetWrapNone() parameter list

Additional information

'Non wrapping' means line wrapping would be done only at new lines. If the horizontal size of the text exceeds the size of the client area the text will be scrolled.

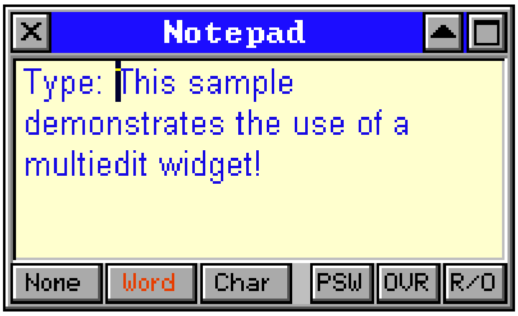
19.19.6 Example

The Sample folder contains the following example which shows how the widget can be used:

- WIDGET_MultiEdit.c

Note that several other examples also make use of this widget and may also be helpful to get familiar with it.

Screenshot of WIDGET_Multiedit.c:



19.20 MULTIPAGE: Multiple page widget

A MULTIPAGE widget is analogous to the dividers in a notebook or the labels in a file cabinet. By using a MULTIPAGE widget, an application can define multiple pages for the same area of a window or dialog box. Each page consists of a certain type of information or a group of widgets that the application displays when the user selects the corresponding page. To select a page the tab of the page has to be clicked. If not all tabs can be displayed, the MULTIPAGE widget automatically shows a small scroll bar at the edge to scroll the pages.
The `Sample` folder contains the file `WIDGET_Multipage.c` which shows how to create and use the MULTIPAGE widget.
The table below shows the appearance of the MULTIPAGE widget:

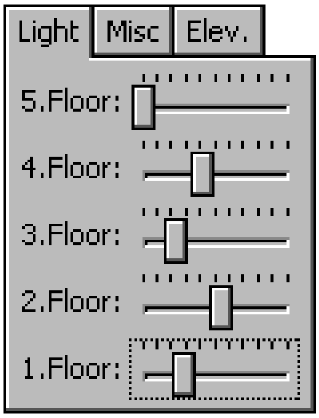
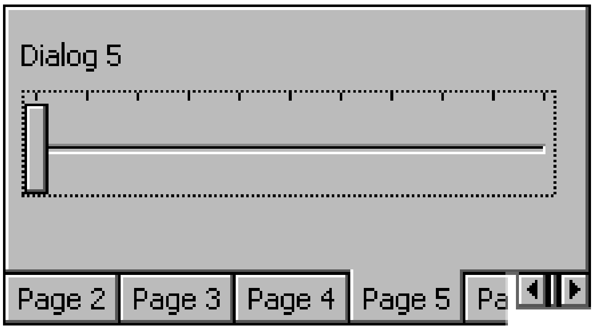
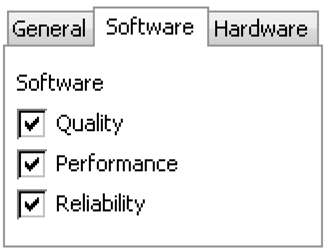
Description	MULTIPAGE widget
MULTIPAGE widget with 3 pages, alignment top/left.	
MULTIPAGE widget with 6 pages, alignment bottom/right.	

Table 19.686: MULTIPAGE appearance

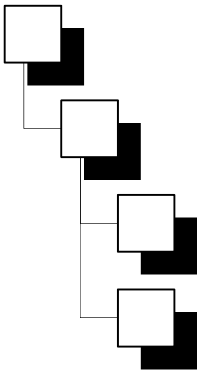
Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter “Skinning” on page 871.

Structure of MULTIPAGE widget

A MULTIPAGE widget with n pages consists of n+2 windows:



- 1 MULTIPAGE window
- 1 Client window
- n Page windows

The page windows will be added to the client window of the widget. The diagram at the right side shows the structure of the widget.

19.20.1 Configuration options

Type	Macro	Default	Description
N	MULTIPAGE_ALIGN_DEFAULT	MULTIPAGE_ALIGN_LEFT | MULTIPAGE_ALIGN_TOP	Default alignment.
N	MULTIPAGE_BKCOLOR0_DEFAULT	0xD0D0D0	Default background color of pages in disabled state.
N	MULTIPAGE_BKCOLOR1_DEFAULT	0xC0C0C0	Default background color of pages in enabled state.
S	MULTIPAGE_FONT_DEFAULT	&GUI_Font13_1	Default font used by the widget.
N	MULTIPAGE_TEXTCOLOR0_DEFAULT	0x808080	Default text color of pages in disabled state.
N	MULTIPAGE_TEXTCOLOR1_DEFAULT	0x000000	Default text color of pages in enabled state.

Table 19.687: Configuration options

19.20.2 Predefined IDs

The following symbols define IDs which may be used to make MULTIPAGE widgets distinguishable from creation: GUI_ID_MULTIPAGE0 - GUI_ID_MULTIPAGE3

19.20.3 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_MOVED_OUT	Widget has been clicked and pointer has been moved out of the widget without releasing.
WM_NOTIFICATION_VALUE_CHANGED	The text of the widget has been changed.

Table 19.688: Notification codes

19.20.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_PGUP	Switches to the next page.
GUI_KEY_PGDOWN	Switches to the previous page.

Table 19.689: Keyboard reaction

19.20.5 MULTIPAGE API

The table below lists the available emWin MULTIPAGE-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
MULTIPAGE_AddEmptyPage()	Adds a page to a MULTIPAGE widget without the requirement to attach a window.
MULTIPAGE_AddPage()	Adds a page to a MULTIPAGE widget.
MULTIPAGE_AttachWindow()	Attaches a window to a certain page of the MULTIPAGE widget.
MULTIPAGE_CreateEx()	Creates a MULTIPAGE widget.
MULTIPAGE_CreateIndirect()	Creates a MULTIPAGE widget from a resource table entry.
MULTIPAGE_CreateUser()	Creates a MULTIPAGE widget using extra bytes as user data.
MULTIPAGE_DeletePage()	Deletes a page from a MULTIPAGE widget.
MULTIPAGE_DisablePage()	Disables a page from a MULTIPAGE widget.
MULTIPAGE_EnablePage()	Enables a page from a MULTIPAGE widget.
MULTIPAGE_EnableScrollbar()	Configures the given MULTIPAGE widget to make use of an automatic scroll bar.
MULTIPAGE_GetDefaultAlign()	Returns the default alignment for MULTIPAGE widgets.
MULTIPAGE_GetDefaultBkColor()	Returns the default background color for MULTIPAGE widgets.
MULTIPAGE_GetDefaultFont()	Returns the default font used for MULTIPAGE widgets.
MULTIPAGE_GetDefaultTextColor()	Returns the default text color used for MULTIPAGE widgets.
MULTIPAGE_GetPageText()	Returns the text of the given page.
MULTIPAGE_GetSelection()	Returns the current selection.
MULTIPAGE_GetUserData()	Retrieves the data set with MULTIPAGE_SetUserData().
MULTIPAGE_GetWindow()	Returns the window handle of a given page.
MULTIPAGE_IsPageEnabled()	Returns if a given page is enabled or not.
MULTIPAGE_SelectPage()	Selects the given page.
MULTIPAGE_SetAlign()	Sets the alignment for the tabs.
MULTIPAGE_SetBitmap()	Sets a bitmap to be displayed on the given tab.
MULTIPAGE_SetBitmapEx()	Sets a bitmap to be displayed at the specified position on the given tab.
MULTIPAGE_SetBkColor()	Sets the background color.
MULTIPAGE_SetDefaultAlign()	Sets the default alignment for new MULTIPAGE widgets.
MULTIPAGE_SetDefaultBkColor()	Sets the default background color for new MULTIPAGE widgets.
MULTIPAGE_SetDefaultBorderSizeX()	Sets the default border size on the x-axis.
MULTIPAGE_SetDefaultBorderSizeY()	Sets the default border size on the y-axis.
MULTIPAGE_SetDefaultFont()	Sets the default font used by new MULTIPAGE widgets.
MULTIPAGE_SetDefaultTextColor()	Sets the default text color used by new MULTIPAGE widgets.
MULTIPAGE_SetFont()	Selects the font for the widget.

Table 19.690: MULTIPAGE API list

Routine	Description
MULTIPAGE_SetRotation()	Sets the rotation mode for the widget.
MULTIPAGE_SetTabHeight()	Sets the height for the given MULTIPAGE widget.
MULTIPAGE_SetTabWidth()	Sets the width for the given tab.
MULTIPAGE_SetText()	Sets the text displayed in a tab of a MULTIPAGE widget.
MULTIPAGE_SetTextAlign()	Sets the text alignment for the given MULTIPAGE widget.
MULTIPAGE_SetTextColor()	Sets the text color.
MULTIPAGE_SetUserData()	Sets the extra data of a MULTIPAGE widget.

Table 19.690: MULTIPAGE API list

MULTIPAGE_AddEmptyPage()

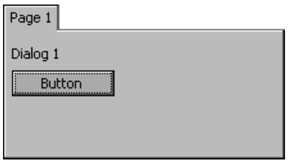
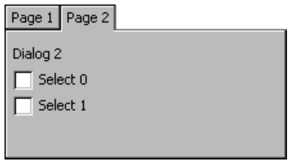
Before	After
	

Table 19.691: MULTIPAGE_AddEmptyPage() before after screenshots

Description

Adds a new page to a given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_AddEmptyPage(MULTIPAGE_Handle   hObj,   WM_HWIN hWin,
                            const char       * pText);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
hWin	Handle of window to be shown in the given page.
pText	Pointer to text to be displayed in the tab of the page.

Table 19.692: MULTIPAGE_AddEmptyPage() parameter list

Additional information

It is recommended, that all windows added to a MULTIPAGE widget handle the complete client area of the MULTIPAGE widget when processing the WM_PAINT message. If no window has to be added, hWin can be specified with 0.

MULTIPAGE_AddPage()

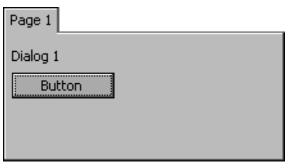
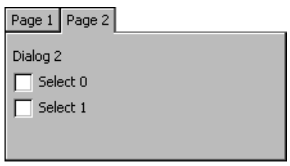
Before	After
	

Table 19.693: MULTIPAGE_AddPage() before after screenshots

Description

Adds a new page to a given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_AddPage(MULTIPAGE_Handle   hObj,   WM_HWIN hWin ,
                       const char        * pText);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
hWin	Handle of window to be shown in the given page.
pText	Pointer to text to be displayed in the tab of the page.

Table 19.694: MULTIPAGE_AddPage() parameter list

Additional information

It is recommended, that all windows added to a MULTIPAGE widget handle the complete client area of the MULTIPAGE widget when processing the `WM_PAINT` message.

MULTIPAGE_AttachWindow()

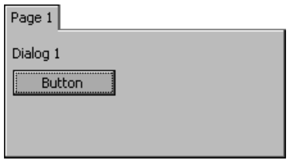
Before	After
	

Table 19.695: MULTIPAGE_AttachWindow() before after screenshots

Description

Attaches a window to a certain page of the MULTIPAGE widget.

Prototype

```
WM_HWIN MULTIPAGE_AttachWindow(MULTIPAGE_Handle hObj,  unsigned Index,
                               WM_HWIN          hWin);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Index	Index of the page the window has to be added to.
pText	Pointer to text to be displayed in the tab of the page.

Table 19.696: MULTIPAGE_AttachWindow() parameter list

Additional information

It is recommended, that all windows added to a MULTIPAGE widget handle the complete client area of the MULTIPAGE widget when processing the `WM_PAINT` message.

MULTIPAGE_CreateEx()

Description

Creates a MULTIPAGE widget of a specified size at a specified position.

Prototype

```
MULTIPAGE_Handle MULTIPAGE_CreateEx(int     x0,      int y0,
                                    int     xSize,   int ySize,
                                    WM_HWIN hParent, int WinFlags,
```



```
                                  int      ExFlags, int Id);
```

Parameter	Description
x0	X-position of the MULTIPAGE widget (in parent coordinates).
y0	Y-position of the MULTIPAGE widget (in parent coordinates).
xSize	Horizontal size of the MULTIPAGE widget (in pixels).
ySize	Vertical size of the MULTIPAGE widget (in pixels).
hParent	Handle of parent window. If 0, the new widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to “WM_CreateWindow()” on page 404 for a list of available parameter values).
ExFlags	Not used yet, reserved for future use.
Id	Window ID of the MULTIPAGE widget.

Table 19.697: MULTIPAGE_CreateEx() parameter list

Return value

Handle of the new MULTIPAGE widget.

Additional information

The size of the tabs depends on the size of the font used for the MULTIPAGE widget.

MULTIPAGE_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `MULTIPAGE_CreateEx()`.

MULTIPAGE_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `MULTIPAGE_CreateEx()` can be referred to.

MULTIPAGE_DeletePage()

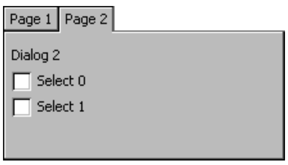
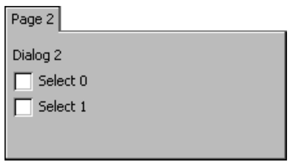
Before	After
	

Table 19.698: MULTIPAGE_DeletePage() before after screenshots

Description

Removes a page from a MULTIPAGE widget and optional deletes the window.

Prototype

```
void MULTIPAGE_DeletePage(MULTIPAGE_Handle hObj,    unsigned Index,
```



```
                              int              Delete);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of the page to be removed from the MULTIPAGE widget.
Delete	If >0 the window attached to the page will be deleted.

Table 19.699: MULTIPAGE_DeletePage() parameter list

MULTIPAGE_DisablePage()

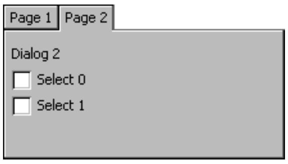
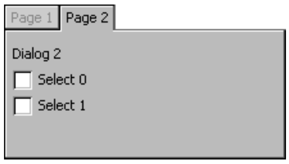
Before	After
	

Table 19.700: MULTIPAGE_DisablePage() before after screenshots

Description

Disables a page from a MULTIPAGE widget.

Prototype

```
void MULTIPAGE_DisablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of the page to be disabled.

Table 19.701: MULTIPAGE_DisablePage() parameter list

Additional information

A disabled page of a window can not be selected by clicking the tab of the page. The default state of MULTIEDIT pages is 'enabled'.

MULTIPAGE_EnablePage()

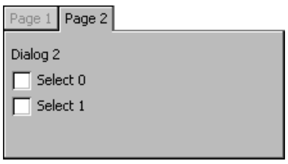
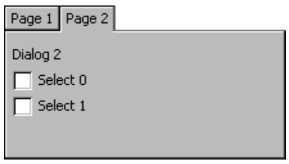
Before	After
	

Table 19.702: MULTIPAGE_EnablePage() before after screenshots

Description

Enables a page of a MULTIPAGE widget.

Prototype

```
void MULTIPAGE_EnablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.

Table 19.703: MULTIPAGE_EnablePage() parameter list

Additional information

The default state of MULTIEDIT pages is 'enabled'.

MULTIPAGE_EnableScrollbar()

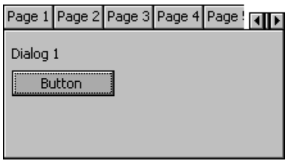
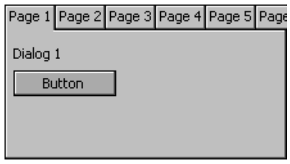
<table>
<tr><th>With scroll bar</th><th>Without scroll bar</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 19.704: MULTIPAGE_EnableScrollbar() screenshots

Description

Configures the given MULTIPAGE widget to make use of an automatic scroll bar.

Prototype

```
void MULTIPAGE_EnableScrollbar(MULTIEDIT_HANDLE hObj, unsigned OnOff);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`OnOff`	1 = Enable automatic scroll bar. 0 = Disable automatic scroll bar.

Table 19.705: MULTIPAGE_EnableScrollbar() parameter list

Additional information

Calling this function takes effect only before the first page has been added to the widget. The automatic scroll bar is enabled by default.

MULTIPAGE_GetDefaultAlign()

Description

Returns the default tab alignment for new MULTIPAGE widgets.

Prototype

```
unsigned MULTIPAGE_GetDefaultAlign(void);
```

Return value

Default tab alignment for new MULTIPAGE widgets.

Additional information

The following table shows the alignment values returned by this function:

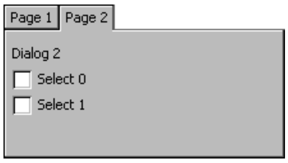
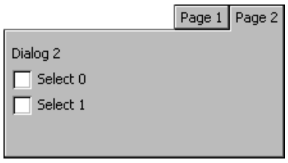
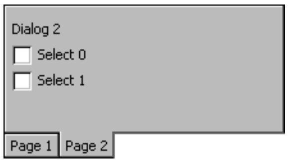
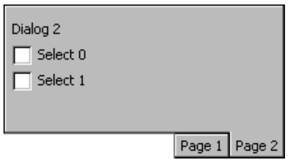
Alignment	Appearance of MULTIPAGE widget
MULTIPAGE_ALIGN_LEFT | MULTIPAGE_ALIGN_TOP	
MULTIPAGE_ALIGN_RIGHT | MULTIPAGE_ALIGN_TOP	
MULTIPAGE_ALIGN_LEFT | MULTIPAGE_ALIGN_BOTTOM	
MULTIPAGE_ALIGN_RIGHT | MULTIPAGE_ALIGN_BOTTOM	

Table 19.706: MULTIPAGE_GetDefaultAlign() additional information

MULTIPAGE_GetDefaultBkColor()

Description

Returns the default background color for new MULTIPAGE widgets.

Prototype

```
GUI_COLOR MULTIPAGE_GetDefaultBkColor(unsigned Index);
```

Parameter	Description
Index	See table below.

Table 19.707: MULTIPAGE_GetDefaultBkColor() parameter list

Permitted values for parameter Index	
0	Returns the default background color for pages in disabled state.
1	Returns the default background color for pages in enabled state.

Return value

Default background color for new MULTIPAGE widgets.

MULTIPAGE_GetDefaultFont()

Description

Returns a pointer to the font used to display the text in the tabs of new MULTIPAGE widgets.

Prototype

```
const GUI_FONT * MULTIPAGE_GetDefaultFont(void);
```


Return value

Pointer to the font used to display the text in the tabs of new MULTIPAGE widgets.

MULTIPAGE_GetDefaultTextColor()

Description

Returns the default text color used to display the text in the tabs of new MULTIPAGE widgets.

Prototype

```
GUI_COLOR MULTIPAGE_GetDefaultTextColor(unsigned Index);
```

Parameter	Description
Index	See table below.

Table 19.708: MULTIPAGE_GetDefaultTextColor() parameter list

Permitted values for parameter Index	
0	Returns the default text color for pages in disabled state.
1	Returns the default text color for pages in enabled state.

Return value

Default text color used to display the text in the tabs of new MULTIPAGE widgets.

MULTIPAGE_GetPageText()

Description

Returns the text of the given page.

Prototype

```
int MULTIPAGE_GetPageText(MULTIPAGE_Handle   hObj,     unsigned Index,
                          char             * pBuffer,  int      MaxLen);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Index	Index of the page.
pBuffer	User defined buffer which is filled with the page text.
MaxLen	Maximum length of the text to be copied.

Table 19.709: MULTIPAGE_GetPageText() parameter list

Return value

Length of the copied text.

MULTIPAGE_GetSelection()

Description

Returns the zero based index of the currently selected page of a MULTIPAGE widget.

Prototype

```
int MULTIPAGE_GetSelection(MULTIPAGE_Handle hObj);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.

Table 19.710: MULTIPAGE_GetSelection() parameter list

Return value

Zero based index of the currently selected page of a MULTIPAGE widget.

MULTIPAGE_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

MULTIPAGE_GetWindow()

Description

Returns the handle of the window displayed in the given page.

Prototype

```
WM_HWIN MULTIPAGE_GetWindow(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Index`	Zero based index of page.

Table 19.711: MULTIPAGE_GetWindow() parameter list

Return value

Handle of the window displayed in the given page.

MULTIPAGE_IsPageEnabled()

Description

Returns if the given page of a MULTIEDIT widget is enabled or not.

Prototype

```
int MULTIPAGE_IsPageEnabled (MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Index`	Zero based index of requested page.

Table 19.712: MULTIPAGE_IsPageEnabled() parameter list

Return value

1 if the given page is enabled, otherwise 0.

MULTIPAGE_SelectPage()

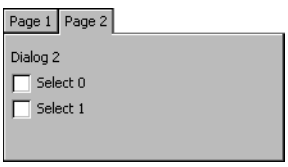
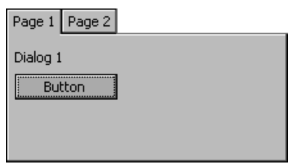
Before	After
	

Table 19.713: MULTIPAGE_SelectPage() before after screenshots

Description

Sets the currently selected page of a MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SelectPage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Index`	Zero based index of page to be selected.

Table 19.714: MULTIPAGE_SelectPage() parameter list

MULTIPAGE_SetAlign()

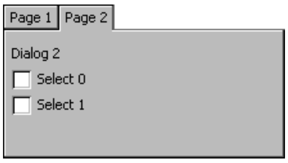
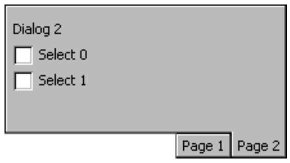
Before	After
	

Table 19.715: MULTIPAGE_SetAlign() before after screenshots

Description

Sets the tab alignment for the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetAlign(MULTIPAGE_Handle hObj, unsigned Align);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Align`	See table below.

Table 19.716: MULTIPAGE_SetAlign() parameter list

Permitted values for parameter `Index` (horizontal and vertical flags are OR-combinable)	
`MULTIPAGE_ALIGN_BOTTOM`	Aligns the tabs at the right side.
`MULTIPAGE_ALIGN_LEFT`	Aligns the tabs at the left side.
`MULTIPAGE_ALIGN_RIGHT`	Aligns the tabs at the top of the widget.
`MULTIPAGE_ALIGN_TOP`	Aligns the tabs at the bottom of the widget.

Additional information

For more information, refer to "MULTIPAGE_GetDefaultAlign()" on page 734.

MULTIPAGE_SetBitmap()

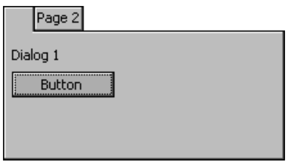
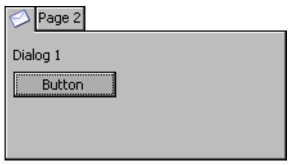
Before	After
	

Table 19.717: MULTIPAGE_SetBitmap() before after screenshots

Description

Sets a bitmap to be displayed on the given tab. The bitmap is horizontally and vertically aligned to the center of the tab.

Prototype

```
int MULTIPAGE_SetBitmap(MULTIPAGE_Handle hObj,  const GUI_BITMAP * pBitmap,
```



```
                    int                Index, int                State);
```

Parameter	Description
hObj	Handle of the MULTIPAGE widget.
pBitmap	Pointer to the bitmap to be used for the given tab.
Index	Index of the tab.
State	State of the tab for which the bitmap has to be used. See table below.

Table 19.718: MULTIPAGE_SetBitmap() parameter list

Permitted values for parameter State (horizontal and vertical flags are OR-combinable)	
MULTIPAGE_BI_SELECTED	Selected state.
MULTIPAGE_BI_UNSELECTED	Unselected state.
MULTIPAGE_BI_DISABLED	Disabled state.

Return value

0 on success, 1 on error.

MULTIPAGE_SetBitmapEx()

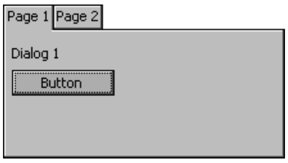
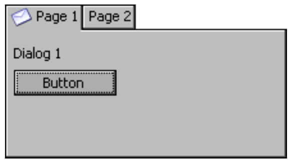
Before	After
	

Table 19.719: MULTIPAGE_SetBitmapEx() before after screenshots

Description

Sets a bitmap to be displayed on the given tab adjusting the position from the center to the right, left, top and bottom according to the x and y values.

Prototype

```
int MULTIPAGE_SetBitmapEx(MULTIPAGE_Handle hObj, const GUI_BITMAP * pBitmap,
                          int x, int y, int Index, int State);
```

Parameter	Description
hObj	Handle of the MULTIPAGE widget.
pBitmap	Pointer to the bitmap to be used for the given tab.
x	Adjustment value for the x position of the bitmap. 0 means horizontally centered.
y	Adjustment value for the y position of the bitmap. 0 means vertically centered.
Index	Index of the tab.
State	State of the tab for which the bitmap has to be used. See table below.

Table 19.720: MULTIPAGE_SetBitmapEx() parameter list

Permitted values for parameter State (horizontal and vertical flags are OR-combinable)	
MULTIPAGE_BI_SELECTED	Selected state.
MULTIPAGE_BI_UNSELECTED	Unselected state.
MULTIPAGE_BI_DISABLED	Disabled state.

Return value

0 on success, 1 on error.

MULTIPAGE_SetBkColor()

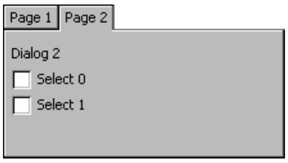
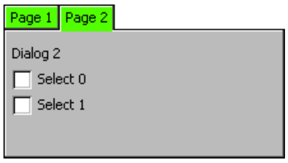
Before	After
	

Table 19.721: MULTIPAGE_SetBkColor() before after screenshots

Description

Sets the background color of the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetBkColor(MULTIPAGE_Handle hObj,   GUI_COLOR Color,
                          unsigned         Index);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Color`	Color to be used.
`Index`	See table below.

Table 19.722: MULTIPAGE_SetBkColor() parameter list

Permitted values for parameter `Index`	
`MULTIPAGE_CI_DISABLED`	Sets the default text color for disabled pages.
`MULTIPAGE_CI_ENABLED`	Sets the default text color for enabled pages.

Additional information

The function only sets the background color for the MULTIPAGE widget. The child windows added to the widget are not affected. That means if the complete client area is drawn by windows added to the widget, only the background color of the tabs changes.

MULTIPAGE_SetDefaultAlign()

Description

Sets the default tab alignment for new MULTIPAGE widgets.

Prototype

```
void MULTIPAGE_SetDefaultAlign(unsigned Align);
```

Parameter	Description
`Align`	Tab alignment used for new MULTIPAGE widgets.

Table 19.723: MULTIPAGE_SetDefaultAlign() parameter list

Additional information

For more information about the tab alignment, refer to “MULTIPAGE_GetDefaultAlign()” on page 734 and “MULTIPAGE_SetAlign()” on page 738.

MULTIPAGE_SetDefaultBkColor()

Description

Sets the default background color used for new MULTIPAGE widgets.

Prototype

```
void MULTIPAGE_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Description
`Color`	Color to be used.
`Index`	See table below.

Table 19.724: MULTIPAGE_SetDefaultBkColor() parameter list

Permitted values for parameter `Index`	
0	Sets the default background color for pages in disabled state.
1	Sets the default background color for pages in enabled state.

MULTIPAGE_SetDefaultBorderSizeX()

Description

Sets the default border size on the x-axis.

Prototype

```
void MULTIPAGE_SetDefaultBorderSizeX(unsigned Size);
```

Parameter	Description
`Size`	Border size to be used.

Table 19.725: MULTIPAGE_SetDefaultBorderSizeX() parameter list

MULTIPAGE_SetDefaultBorderSizeY()

Description

Sets the default border size on the y-axis.

Prototype

```
void MULTIPAGE_SetDefaultBorderSizeY(unsigned Size);
```

Parameter	Description
`Size`	Border size to be used.

Table 19.726: MULTIPAGE_SetDefaultBorderSizeY() parameter list

MULTIPAGE_SetDefaultFont()

Description

Sets the default font used to display the text in the tabs of new MULTIPAGE widgets.

Prototype

```
void MULTIPAGE_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
`pFont`	Pointer to GUI_FONT structure to be used.

Table 19.727: MULTIPAGE_SetDefaultFont() parameter list

Additional information

The horizontal and vertical size of the tabs depends on the size of the used font.

MULTIPAGE_SetDefaultTextColor()

Description

Sets the default text color used to display the text in the tabs of new MULTIPAGE widgets.

Prototype

```
void MULTIPAGE_SetDefaultTextColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Description
`Color`	Color to be used.
`Index`	See table below.

Table 19.728: MULTIPAGE_SetDefaultTextColor() parameter list

MULTIPAGE_SetFont()

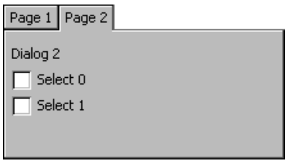
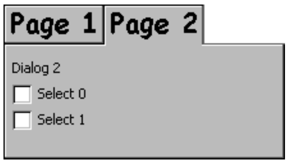
Before	After
	

Table 19.729: MULTIPAGE_SetFont() before after screenshots

Description

Sets the font used to display the text in the tabs of a given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetFont(MULTIPAGE_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`pFont`	Pointer to GUI_FONT structure used to display the text in the tabs.

Table 19.730: MULTIPAGE_SetFont() parameter list

Additional information

The vertical and horizontal size of the tabs depend on the size of the used font and the text shown in the tabs.

MULTIPAGE_SetRotation()

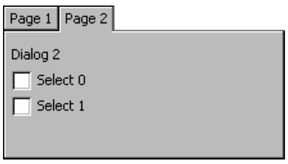
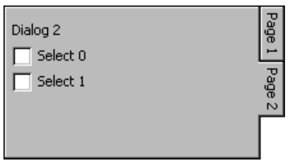
Before	After
	

Table 19.731: MULTIPAGE_SetRotation() before after screenshots

Description

Sets the rotation mode of the given widget.

Prototype

```
void MULTIPAGE_SetRotation(MULTIPAGE_Handle hObj, unsigned Rotation);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Rotation`	Rotation mode. See table below.

Table 19.732: MULTIPAGE_SetRotation() parameter list

Permitted values for parameter Index	
MULTIPAGE_CF_ROTATE_CW	Arranges the tabs at the vertical side and rotates the tab text by 90 degrees clockwise.
0	Default horizontal mode.

MULTIPAGE_SetTabHeight()

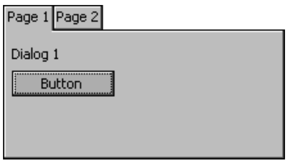
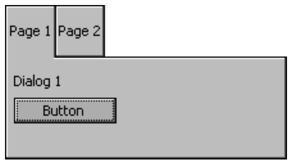
Before	After
	

Table 19.733: MULTIPAGE_SetTabHeight() before after screenshots

Description

Sets the height for all tabs.

Prototype

```
void MULTIPAGE_SetTabHeight(MULTIPAGE_Handle hObj, int Height);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Height	Height to be set.

Table 19.734: MULTIPAGE_SetTabHeight() parameter list

MULTIPAGE_SetTabWidth()

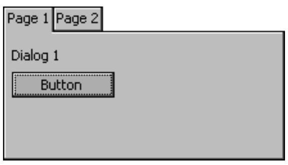
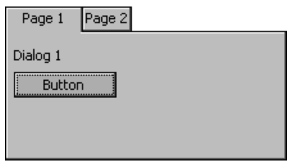
Before	After
	

Table 19.735: MULTIPAGE_SetTabWidth() before after screenshots

Description

Sets the width for the given tab.

Prototype

```
void MULTIPAGE_SetTabWidth(MULTIPAGE_Handle hObj, int Width, int Index);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Width	Width to be set.
Index	Index of the tab.

Table 19.736: MULTIPAGE_SetTabWidth() parameter list

MULTIPAGE_SetText()

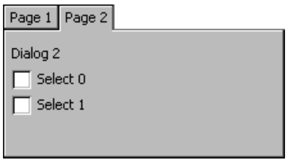
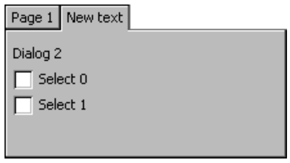
Before	After
	

Table 19.737: MULTIPAGE_SetText() parameter list

Description

Sets the text displayed in the tab of a given page.

Prototype

```
void MULTIPAGE_SetText(MULTIPAGE_Handle hObj,   const char * pText,
                       unsigned          Index);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
pText	Pointer to the text to be displayed.
Index	Zero based index of the page.

Table 19.738: MULTIPAGE_SetText() parameter list

MULTIPAGE_SetTextAlign()

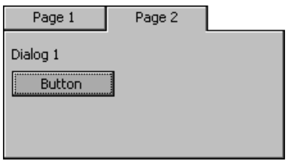
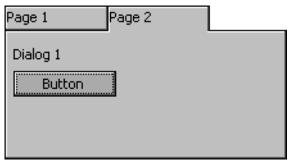
Before	After
	

Table 19.739: MULTIPAGE_SetTextAlign() before after screenshots

Description

Sets the text alignment for the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetTextAlign(MULTIPAGE_Handle hObj, unsigned Align);
```

Parameter	Description
hObj	Handle of MULTIPAGE widget.
Align	Text alignment. See table below.

Table 19.740: MULTIPAGE_SetTextAlign() parameter list

Additional information

Setting the text alignment can have a visual impact only in case the tab width was changed. Otherwise the width is set according to the width of the text.

MULTIPAGE_SetTextColor()

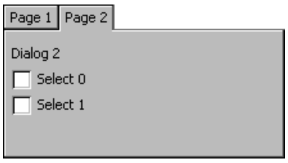
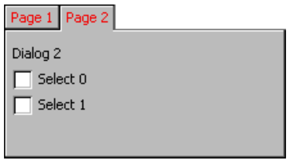
Before	After
	

Table 19.741: MULTIPAGE_SetTextColor() before after screenshots

Description

Sets the color used to display the text in the tabs of a MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetTextColor(MULTIPAGE_Handle hObj,   GUI_COLOR Color,
                            unsigned          Index);
```

Parameter	Description
`hObj`	Handle of MULTIPAGE widget.
`Color`	Color to be used.
`Index`	See table below.

Table 19.742: MULTIPAGE_SetTextColor() parameter list

Permitted values for parameter `Index`	
0	Sets the text color for pages in disabled state.
1	Sets the text color for pages in enabled state.

MULTIPAGE_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

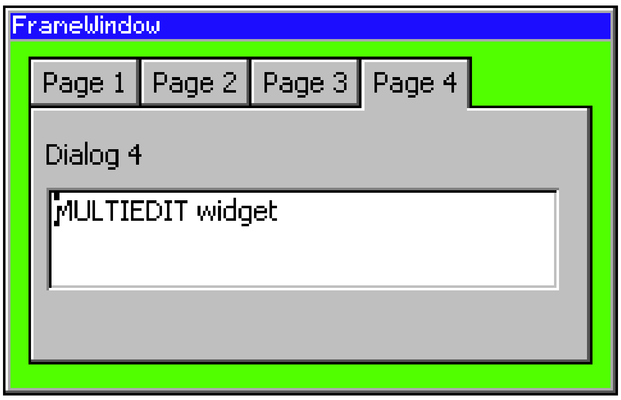
19.20.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_Multipage.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Multipage.c:



19.21 PROGBAR: Progress bar widget

PROGBAR widgets are commonly used in applications for visualization; for example, a tank fill-level indicator or an oil-pressure indicator. Example screenshots can be found at the beginning of the chapter and at end of this section. All PROGBAR-related routines are in the file(s) `PROGBAR*.c`, `PROGBAR.h`. All identifiers are prefixed PROGBAR.

Skinning...

...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter "Skinning" on page 871.

19.21.1 Configuration options

Type	Macro	Default	Description
S	PROGBAR_DEFAULT_FONT	GUI_DEFAULT_FONT	Font used.
N	PROGBAR_DEFAULT_BARCOLOR0	0x555555 (dark gray)	Left bar color.
N	PROGBAR_DEFAULT_BARCOLOR1	0xAAAAAA (light gray)	Right bar color.
N	PROGBAR_DEFAULT_TEXTCOLOR0	0xFFFFFF	Text color, left bar.
N	PROGBAR_DEFAULT_TEXTCOLOR1	0x000000	Text color, right bar.

Table 19.743: Configuration options

19.21.2 Predefined IDs

The following symbols define IDs which may be used to make PROGBAR widgets distinguishable from creation: `GUI_ID_PROGBAR0` - `GUI_ID_PROGBAR3`

19.21.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

19.21.4 PROGBAR API

The table below lists the available emWin PROGBAR-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
PROGBAR_Create()	Creates a PROGBAR widget. (Obsolete)
PROGBAR_CreateAsChild()	Creates a PROGBAR widget as a child window. (Obsolete)
PROGBAR_CreateEx()	Creates a PROGBAR widget.
PROGBAR_CreateIndirect()	Creates a PROGBAR widget from resource table entry.
PROGBAR_CreateUser()	Creates a PROGBAR widget using extra bytes as user data.
PROGBAR_GetMinMax()	Returns the minimum and maximum of the PROGBAR widget.
PROGBAR_GetUserData()	Retrieves the data set with PROGBAR_SetUserData().
PROGBAR_GetValue()	Returns the current value.
PROGBAR_SetBarColor()	Sets the color(s) for the bar.
PROGBAR_SetFont()	Select the font for the text.
PROGBAR_SetMinMax()	Set the minimum and maximum values used for the bar.
PROGBAR_SetText()	Set the (optional) text for the bar graph.
PROGBAR_SetTextAlign()	Set text alignment (default is centered).
PROGBAR_SetTextColor()	Set the color(s) for the text.

Table 19.744: PROGBAR API list

Routine	Description
PROGBAR_SetTextPos()	Set the text position (default 0,0).
PROGBAR_SetUserData()	Sets the extra data of a PROGBAR widget.
PROGBAR_SetValue()	Set the value for the bar graph (and percentage if no text has been assigned).

Table 19.744: PROGBAR API list

PROGBAR_Create()

(Obsolete, PROGBAR_CreateEx() should be used instead)

Description

Creates a PROGBAR widget of a specified size at a specified location.

Prototype

```
PROGBAR_Handle PROGBAR_Create(int x0,     int y0,
                              int xSize, int ySize, int Flags);
```

Parameter	Description
x0	Leftmost pixel of the PROGBAR widget (in parent coordinates).
y0	Topmost pixel of the PROGBAR widget (in parent coordinates).
xSize	Horizontal size of the PROGBAR widget (in pixels).
ySize	Vertical size of the PROGBAR widget (in pixels).
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).

Table 19.745: PROGBAR_Create() parameter list

Return value

Handle of the created PROGBAR widget; 0 if the function fails.

PROGBAR_CreateAsChild()

(Obsolete, PROGBAR_CreateEx should be used instead)

Description

Creates a PROGBAR widget as a child window.

Prototype

```
PROGBAR_Handle PROGBAR_CreateAsChild(int      x0,      int y0,
                                     int      xSize,   int ySize,
                                     WM_HWIN hParent, int Id,
                                     int      Flags);
```

Parameter	Description
x0	X-position of the PROGBAR widget relative to the parent window.
y0	Y-position of the PROGBAR widget relative to the parent window.
xSize	Horizontal size of the PROGBAR widget (in pixels).
ySize	Vertical size of the PROGBAR widget (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
Flags	Window create flags (see PROGBAR_Create()).

Table 19.746: PROGBAR_CreateAsChild() parameter list

Return value

Handle of the created PROGBAR widget; 0 if the function fails.

PROGBAR_CreateEx()

Description

Creates a PROGBAR widget of a specified size at a specified location.

Prototype

```
PROGBAR_Handle PROGBAR_CreateEx(int      x0,       int y0,
                                int      xSize,    int ySize,
                                WM_HWIN  hParent,  int WinFlags,
                                int      ExFlags,  int Id);
```

Parameter	Description
`x0`	Leftmost pixel of the PROGBAR widget (in parent coordinates).
`y0`	Topmost pixel of the PROGBAR widget (in parent coordinates).
`xSize`	Horizontal size of the PROGBAR widget (in pixels).
`ySize`	Vertical size of the PROGBAR widget (in pixels).
`hParent`	Handle of parent window. If 0, the new PROGBAR widget will be a child of the desktop (top-level window).
`WinFlags`	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
`ExFlags`	See table below.
`Id`	Window ID of the PROGBAR widget.

Table 19.747: PROGBAR_CreateEx() parameter list

Permitted values for parameter `ExFlags`	
`PROGBAR_CF_VERTICAL`	A vertical PROGBAR widget will be created. Vertical PROGBAR widgets do not show any text.
`PROGBAR_CF_HORIZONTAL`	A horizontal PROGBAR widget will be created.

Return value

Handle of the created PROGBAR widget; 0 if the function fails.

PROGBAR_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element Para of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `PROGBAR_CreateEx()`.

PROGBAR_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `PROGBAR_CreateEx()` can be referred to.

PROGBAR_GetMinMax()

Description

Copies the minimum value and maximum value of the given PROGBAR widget to the given integer pointers.

Prototype

```
void PROGBAR_GetMinMax(PROGBAR_Handle hObj, int * pMin, int * pMax);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
pMin	Minimum value (Range: -16383 < Min <= 16383).
pMax	Maximum value (Range: -16383 < Max <= 16383)

Table 19.748: PROGBAR_SetMinMax() parameter list

PROGBAR_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

PROGBAR_GetValue()

Description

Returns the current value of the given PROGBAR widget.

Prototype

```
int PROGBAR_GetValue(PROGBAR_Handle hObj);
```

Parameter	Description
hObj	Handle of PROGBAR widget.

Table 19.749:

Return value

Current value of the PROGBAR widget.

PROGBAR_SetBarColor()

Description

Sets the color(s) of the PROGBAR widget.

Prototype

```
void PROGBAR_SetBarColor(PROGBAR_Handle hObj,   unsigned int Index,
                         GUI_COLOR      Color);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
Index	See table below. Other values are not permitted.
Color	Color to set (24-bit RGB value).

Table 19.750: PROGBAR_SetBarColor() parameter list

Permitted values for parameter Index	
0	Left/lower portion of the PROGBAR widget.
1	Right/upper portion of the PROGBAR widget.

PROGBAR_SetFont()

Description

Selects the font for the text display inside the PROGBAR widget.

Prototype

```
void PROGBAR_SetFont(PROGBAR_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
pFont	Pointer to the font.

Table 19.751: PROGBAR_SetFont() parameter list

Additional information

If this function is not called, the default font for PROGBAR widgets (the GUI default font) will be used. However, the default font of the PROGBAR widget may be changed in the GUIConf.h file.
Simply #define the default font as follows (example):

```
#define PROGBAR_DEFAULT_FONT &GUI_Font13_ASCII
```

PROGBAR_SetMinMax()

Description

Sets the minimum and maximum values used for the PROGBAR widget.

Prototype

```
void PROGBAR_SetMinMax(PROGBAR_Handle hObj, int Min, int Max);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
Min	Minimum value (Range: -16383 < Min <= 16383).
Max	Maximum value (Range: -16383 < Max <= 16383)

Table 19.752: PROGBAR_SetMinMax() parameter list

Additional information

If this function is not called, the default values of Min = 0, Max = 100 will be used.

PROGBAR_SetText()

Description

Sets the text displayed inside the PROGBAR widget.

Prototype

```
void PROGBAR_SetText(PROGBAR_Handle hObj, const char * s);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
s	Text to display. A NULL pointer is permitted; in this case a percentage value will be displayed.

Table 19.753: PROGBAR_SetText() parameter list

Additional information

If this function is not called, a percentage value will be displayed as the default. If you do not want to display any text at all, you should set an empty string.

PROGBAR_SetTextAlign()

Description

Sets the text alignment.

Prototype

```
void PROGBAR_SetTextAlign(PROGBAR_Handle hObj, int Align);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
Align	Horizontal alignment attribute for the text. See table below.

Table 19.754: PROGBAR_SetTextAlign() parameter list

Permitted values for parameter Align	
GUI_TA_HCENTER	Centers the title (default).
GUI_TA_LEFT	Displays the title to the left.
GUI_TA_RIGHT	Displays the title to the right.

Additional information

If this function is not called, the default behavior is to display the text centered.

PROGBAR_SetTextColor()

Description

Sets the text color of the PROGBAR widget.

Prototype

```
void PROGBAR_SetTextColor(PROGBAR_Handle hObj,   unsigned int Index,
                          GUI_COLOR      Color);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
Index	See table below. Other values are not permitted.
Color	Color to set (24-bit RGB value).

Table 19.755: PROGBAR_SetTextColor() parameter list

Permitted values for parameter Index	
0	Left portion of the text.
1	Right portion of the text.

PROGBAR_SetTextPos()

Description

Sets the text position in pixels.

Prototype

```
void PROGBAR_SetTextPos(PROGBAR_Handle hObj, int XOff, int YOff);
```

Parameter	Description
hObj	Handle of PROGBAR widget.
XOff	Number of pixels to move text in horizontal direction. Positive number will move text to the right.
YOff	Number of pixels to move text in vertical direction. Positive number will move text down.

Table 19.756: PROGBAR_SetTextPos() parameter list

Additional information

The values move the text the specified number of pixels within the widget. Normally, the default of (0,0) should be sufficient.

PROGBAR_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

PROGBAR_SetValue()

Description

Sets the value of the PROGBAR widget.

Prototype

```
void PROGBAR_SetValue(PROGBAR_Handle hObj, int v);
```

Parameter	Description
`hObj`	Handle of PROGBAR widget.
`v`	Value to set.

Table 19.757: PROGBAR_SetValue() parameter list

Additional information

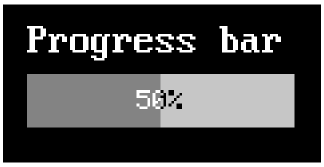
The bar indicator will be calculated with regard to the max/min values. If a percentage is automatically displayed, the percentage will also be calculated using the given min/max values as follows:
p = 100% * (v-Min)/(Max-Min)
The default value after creation of the widget is 0.

19.21.5 Examples

The `Sample` folder contains the following examples which show how the widget can be used:
- WIDGET_SimpleProgbar.c
- WIDGET_Progbar.c

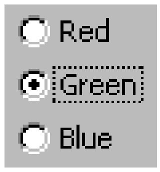
Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_SimpleProgbar.c:



Screenshot of WIDGET_Progbar.c:

19.22 RADIO: Radio button widget

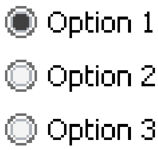


Radio buttons, like check boxes, are used for selecting choices. A dot appears when a RADIO button is turned on or selected. The difference from check boxes is that the user can only select one RADIO button at a time. When a button is selected, the other buttons in the widget are turned off, as shown to the right. One RADIO button widget may contain any number of buttons, which are always arranged vertically.
All RADIO-related routines are located in the file(s) `RADIO*.c`, `RADIO.h`.
All identifiers are prefixed RADIO. The table below shows the default appearances of a RADIO button:

	Selected	Unselected
Enabled		
Disabled		

Table 19.758: RADIO appearance

Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter "Skinning" on page 871.

19.22.1 Configuration options

Type	Macro	Default	Description
S	`RADIO_IMAGE0_DEFAULT`	(see table above)	Default outer image used to show a disabled RADIO button.
S	`RADIO_IMAGE1_DEFAULT`	(see table above)	Default outer image used to show a enabled RADIO button.
S	`RADIO_IMAGE_CHECK_DEFAULT`	(see table above)	Default inner image used to mark the selected item.
N	`RADIO_FONT_DEFAULT`	&GUI_Font13_1	Default font used to render the text of the RADIO widget.
N	`RADIO_DEFAULT_TEXT_COLOR`	GUI_BLACK	Default text color of RADIO widget.
N	`RADIO_DEFAULT_BKCOLOR`	0xC0C0C0	Default background color of RADIO buttons if no transparency is used.
N	`RADIO_FOCUSCOLOR_DEFAULT`	GUI_BLACK	Default color for rendering the focus rectangle.

Table 19.759: Configuration options

19.22.2 Predefined IDs

The following symbols define IDs which may be used to make RADIO widgets distinguishable from creation: GUI_ID_RADIO0 - GUI_ID_RADIO7

19.22.3 Notification codes

The following events are sent from a RADIO widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	RADIO button has been clicked.
WM_NOTIFICATION_RELEASED	RADIO button has been released.
WM_NOTIFICATION_MOVED_OUT	RADIO button has been clicked and pointer has been moved out of the button without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Value (selection) of the RADIO widget has changed.

Table 19.760: Notification codes

19.22.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	Increments the selection by 1.
GUI_KEY_DOWN	Increments the selection by 1.
GUI_KEY_LEFT	Decrements the selection by 1.
GUI_KEY_UP	Decrements the selection by 1.

Table 19.761: Keyboard reaction

19.22.5 RADIO API

The table below lists the available emWin RADIO-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
RADIO_Create()	Creates a RADIO widget. (Obsolete)
RADIO_CreateEx()	Creates a RADIO widget.
RADIO_CreateIndirect()	Creates a RADIO widget from resource table entry.
RADIO_CreateUser()	Creates a RADIO widget using extra bytes as user data.
RADIO_Dec()	Decrement the button selection by a value of 1.
RADIO_GetDefaultFont()	Returns the default font used to show the text of new RADIO widgets
RADIO_GetDefaultTextColor()	Returns the default text color used to show the text of new RADIO widgets
RADIO_GetText()	Returns the text of a RADIO item.
RADIO_GetUserData()	Retrieves the data set with RADIO_SetUserData().
RADIO_GetValue()	Return the currently selected button.
RADIO_Inc()	Increment the button selection by a value of 1.
RADIO_SetBkColor()	Sets the background color of the RADIO widget.
RADIO_SetDefaultFocusColor()	Sets the default focus rectangle color for new RADIO widgets.
RADIO_SetDefaultFont()	Sets the default font used to show the text of new RADIO widgets.
RADIO_SetDefaultImage()	Sets the images to be used for new RADIO buttons.
RADIO_SetDefaultTextColor()	Sets the default text color used to show the text of new RADIO widgets.
RADIO_SetFocusColor()	Sets the color of the focus rectangle.
RADIO_SetFont()	Sets the font used to show the text of the RADIO widget.

Table 19.762: RADIO widget API list

Routine	Description
RADIO_SetGroupId()	Sets the group Id of the given RADIO widget.
RADIO_SetImage()	Sets the images used to display the RADIO button.
RADIO_SetText()	Sets the text
RADIO_SetTextColor()	Sets the text color used to show the text of the RADIO widget.
RADIO_SetUserData()	Sets the extra data of a RADIO widget.
RADIO_SetValue()	Set the button selection.

Table 19.762: RADIO widget API list

RADIO_Create()

(Obsolete, `RADIO_CreateEx()` should be used instead)

Description

Creates a RADIO widget of a specified size at a specified location.

Prototype

```
RADIO_Handle RADIO_Create(int      x0,       int      y0,
                          int      xSize,    int      ySize,
                          WM_HWIN  hParent,  int      Id,
                          int      Flags,    unsigned Para);
```

Parameter	Description
x0	Leftmost pixel of the RADIO widget (in parent coordinates).
y0	Topmost pixel of the RADIO widget (in parent coordinates).
xSize	Horizontal size of the RADIO widget (in pixels).
ySize	Vertical size of the RADIO widget (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
Flags	Window create flags. Typically `WM_CF_SHOW` in order to make the widget visible immediately (refer to `WM_CreateWindow()` in the chapter “The Window Manager (WM)” on page 375 for a list of available parameter values).
Para	Number of buttons in the group.

Table 19.763: RADIO_Create() parameter list

Return value

Handle of the created RADIO widget; 0 if the function fails.

RADIO_CreateEx()

Description

Creates a RADIO widget of a specified size at a specified location.

Prototype

```
RADIO_Handle RADIO_CreateEx(int      x0,        int y0,
                            int      xSize,     int ySize,
                            WM_HWIN  hParent,   int WinFlags,
                            int      ExFlags,   int Id,
                            int      NumItems,  int Spacing);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).

Table 19.764: RADIO_CreateEx() parameter list

Parameter	Description
hParent	Handle of parent window. If 0, the new RADIO widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	Not used, reserved for future use.
Id	Window ID of the widget.
NumItems	Number of items contained by the RADIO widget. (default is 2)
Spacing	Number of vertical pixels used for each item of the RADIO widget.

Table 19.764: RADIO_CreateEx() parameter list

Return value

Handle of the created RADIO widget; 0 if the function fails.

Additional information

If creating a RADIO widget make sure, that the given ySize is enough to show all items. The value should be at least NumItems * Spacing. If the given value of NumItems is <= 0 a default value of 2 is used.

RADIO_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. For details the function <WIDGET>_CreateIndirect() should be referred to. The element Flags of the according GUI_WIDGET_CREATE_INFO structure is not used. The following table shows the use of the element Para:

Bits	Description
0 - 7	Number of items of the RADIO widget. If 0, a default value of 2 items is used.
8 - 15	Number of vertical pixels used for each item. If 0 the height of the default image is used.
16 - 23	Not used, reserved for future use.
24 - 31	Not used, reserved for future use.

Table 19.765: Bit differentiation of the Para element

RADIO_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function RADIO_CreateEx() can be referred to.

RADIO_Dec()

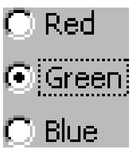
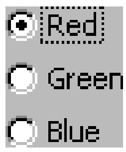
Before	After
	

Table 19.766: RADIO_Dec() before after screenshots

Description

Decrements the selection by 1.

Prototype

```
void RADIO_Dec(RADIO_Handle hObj);
```

Parameter	Description
hObj	Handle of RADIO widget.

Table 19.767: RADIO_Dec() parameter list

Additional information

Note that the numbering of the buttons always starts from the top with a value of 0; therefore, decrementing the selection will actually move the selection one button up.

RADIO_GetDefaultFont()

Description

Returns the default font used to display the optional text next to new RADIO buttons.

Prototype

```
const GUI_FONT * RADIO_GetDefaultFont(void);
```

Return value

Default font used to display the optional text next to the RADIO buttons.

Additional information

For information about how to add text to a RADIO widget, refer to “RADIO_SetText()” on page 763.

RADIO_GetDefaultTextColor()

Description

Returns the default text color used to display the optional text next to new RADIO buttons.

Prototype

```
GUI_COLOR RADIO_GetDefaultTextColor (void);
```

Return value

Default text color used to display the optional text next to new RADIO buttons.

Additional information

For information about how to add text to a RADIO widget, refer to “RADIO_SetText()” on page 763.

RADIO_GetText()

Description

Returns the optional text of the given RADIO widget.

Prototype

```
int RADIO_GetText(RADIO_Handle   hObj,     unsigned Index,
                  char         * pBuffer,  int      MaxLen);
```

Parameter	Description
hObj	Handle of RADIO widget.
Index	Index of the desired item.
pBuffer	Pointer to buffer to which the text will be copied.
MaxLen	Buffer size in bytes.

Table 19.768: RADIO_GetText() parameter list

Return value

Length of the text copied into the buffer.

Additional information

If the desired item of the RADIO widget contains no text the function returns 0 and the buffer remains unchanged.

RADIO_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

RADIO_GetValue()

Description

Returns the currently selected button.

Prototype

```
int RADIO_GetValue(RADIO_Handle hObj);
```

Parameter	Description
hObj	Handle of RADIO widget.

Table 19.769: RADIO_GetValue() parameter list

Return value

The value of the currently selected button. If no button is selected (in case of using a RADIO button group) the return value is -1.

Additional information

For information about how to use groups of RADIO buttons, refer to "RADIO_SetGroupId()" on page 762.

RADIO_Inc()

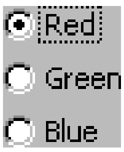
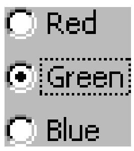
Before	After
	

Table 19.770: RADIO_Inc() before after screenshots

Description

Increments the selection by a value of 1.

Prototype

```
void RADIO_Inc(RADIO_Handle hObj);
```

Parameter	Description
hObj	Handle of RADIO widget.

Table 19.771: RADIO_Inc() parameter list

Additional information

Note that the numbering of the buttons always starts from the top with a value of 0; therefore, incrementing the selection will actually move the selection one button down.

RADIO_SetBkColor()

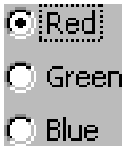
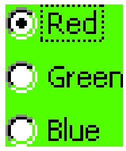
Before	After
	

Table 19.772: RADIO_SetBkColor() before after screenshots

Description

Sets the background color of the RADIO widget.

Prototype

```
void RADIO_SetBkColor(RADIO_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of RADIO widget.
Color	Color to be used for the background. (range 0x000000 and 0xFFFFFF or a valid color define) GUI_INVALID_COLOR to make background transparent

Table 19.773: RADIO_SetBkColor() parameter list

Additional information

The background of this widget can either be filled with any available color or transparent. If a valid RGB color is specified, the background is filled with the color, otherwise the background (typically the content of the parent window) is visible. If the background is transparent, the widget is treated as transparent window, otherwise as non-transparent window. Note that using a background color allows more efficient (faster) rendering. If skinning is active this function should not be used.

RADIO_SetDefaultFocusColor()

Description

Sets the default focus rectangle color for new RADIO buttons.

Prototype

```
GUI_COLOR RADIO_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Description
Color	Default color to be used for new RADIO buttons.

Table 19.774: RADIO_SetDefaultFocusColor() parameter list

Return value

Previous default focus rectangle color.

Additional information

For more information, refer to “RADIO_SetFocusColor()” on page 761 .

RADIO_SetDefaultFont()

Description

Sets the default font used to display the optional text next to new RADIO buttons.

Prototype

```
void RADIO_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to GUI_FONT structure used to show the text of new RADIO widgets.

Table 19.775: RADIO_SetDefaultFont() parameter list

Additional information

For information about how to add text to a RADIO widget, refer to “RADIO_SetText()” on page 763.

RADIO_SetDefaultImage()

Description

Sets the images used to draw new RADIO buttons.

Prototype

```
void RADIO_SetDefaultImage(const GUI_BITMAP * pBitmap, unsigned int Index);
```

Parameter	Description
pBitmap	Pointer to the bitmap.
Index	See table below.

Table 19.776: RADIO_SetDefaultImage() parameter list

Permitted values for parameter Index	
RADIO_BI_INACTIV	Outer image used to show a disabled RADIO button.
RADIO_BI_ACTIV	Outer image used to show a enabled RADIO button.
RADIO_BI_CHECK	Inner image used to mark the selected item.

Additional information

Two images are used to display a RADIO button. One image is used to draw the outer frame used to display a unselected RADIO button. In dependence of the current state it will be the bitmap referenced by RADIO_BI_ACTIV (default) or by RADIO_BI_ACTIV. The second image (referenced by RADIO_BI_CHECK) is used to mark the currently selected button.

RADIO_SetDefaultTextColor()

Description

Sets the default text color used to display the optional text next to new RADIO buttons.

Prototype

```
void RADIO_SetDefaultTextColor(GUI_COLOR TextColor);
```

Parameter	Description
TextColor	New color to be used.

Table 19.777: RADIO_SetDefaultTextColor() parameter list

Additional information

For information about how to add text to a RADIO widget, refer to “RADIO_SetText()” on page 763.

RADIO_SetFocusColor()

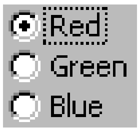
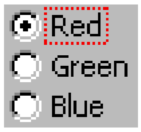
Before	After
	

Table 19.778: RADIO_SetFocusColor() before after screenshots

Description

Sets the color used to render the focus rectangle of the RADIO button.

Prototype

```
GUI_COLOR RADIO_SetFocusColor(RADIO_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of RADIO widget.
Color	Color to be used for the focus rectangle.

Table 19.779: RADIO_SetFocusColor() parameter list

Return value

Previous color of the focus rectangle.

Additional information

The focus rectangle is only visible if the widget has the input focus.

RADIO_SetFont()

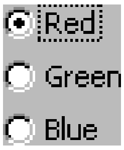
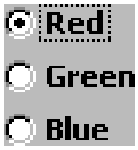
Before	After
	

Table 19.780: RADIO_SetFont() before after screenshots

Description

Sets the font used to display the optional text next to the RADIO buttons.

Prototype

```
void RADIO_SetFont(RADIO_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of RADIO widget.
pFont	Pointer to GUI_FONT structure to be used to display the text.

Table 19.781: RADIO_SetFont() parameter list

Additional information

For information about how to add text to a RADIO widget, refer to “RADIO_SetText()” on page 763.

RADIO_SetGroupId()

<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 19.782: RADIO_SetGroupId() before after screenshots

Description

Sets the group ID of the RADIO widget.

Prototype

```
void RADIO_SetGroupId(RADIO_Handle hObj, U8 GroupId);
```

Parameter	Description
`hObj`	Handle of RADIO widget.
`GroupId`	ID of the RADIO button group. Must be between 1 and 255. If the value is 0 the RADIO widget is not assigned to a RADIO button group.

Table 19.783: RADIO_SetGroupId() parameter list

Additional information

This command can be used to create groups of RADIO buttons. The behavior of one group is the same as the behavior of one RADIO button. This makes it possible to create for example 2 RADIO widgets side by side with 3 buttons each and build one group of them.

Example

The following example shows how to create a group of 2 RADIO widgets as shown in the screenshot at the beginning of the function description:

```
hRadio_0 = RADIO_CreateEx(10, 10, 60, 0, WM_HBKWIN, WM_CF_SHOW, 0, 1234, 3, 20);
RADIO_SetText(hRadio_0, "Red", 0);
RADIO_SetText(hRadio_0, "Green", 1);
RADIO_SetText(hRadio_0, "Blue", 2);
hRadio_1 = RADIO_CreateEx(65, 10, 60, 0, WM_HBKWIN, WM_CF_SHOW, 0, 1234, 3, 20);
RADIO_SetText(hRadio_1, "Magenta", 0);
RADIO_SetText(hRadio_1, "Cyan", 1);
RADIO_SetText(hRadio_1, "Yellow", 2);
RADIO_SetGroupId(hRadio_0, 1);
RADIO_SetGroupId(hRadio_1, 1);
```

RADIO_SetImage()

Description

Sets the images used to draw the RADIO button.

Prototype

```
void RADIO_SetImage(RADIO_Handle hObj,   const GUI_BITMAP * pBitmap,
                    unsigned int Index);
```

Parameter	Description
`hObj`	Handle of RADIO widget.
`pBitmap`	Pointer to the bitmap.
`Index`	(see table shown under `RADIO_SetDefaultImage`)

Table 19.784: RADIO_SetImage() parameter list

Additional information

(see `RADIO_SetDefaultImage`).

RADIO_SetText()

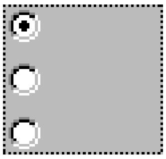
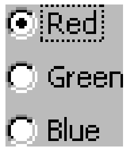
Before	After
	

Table 19.785: RADIO_SetText() before after screenshots

Description

Sets the optional text shown next to the RADIO buttons.

Prototype

```
void RADIO_SetText(RADIO_Handle hObj, const char * pText, unsigned Index);
```

Parameter	Description
`hObj`	Handle of RADIO widget.
`pText`	Pointer to the text to be shown next to the specified RADIO button.
`Index`	Zero based index of the RADIO button.

Table 19.786: RADIO_SetText() parameter list

Additional information

If using a RADIO widget without text (old style) the focus rectangle is drawn around the buttons of the widget. If using RADIO button text the focus rectangle is shown around the text of the currently selected RADIO button of the widget.

Example

The following example shows how to add the text shown in the screenshot above:

```
RADIO_SetText(hRadio_0, "Red",   0);
RADIO_SetText(hRadio_0, "Green", 1);
RADIO_SetText(hRadio_0, "Blue",  2);
```

RADIO_SetTextColor()

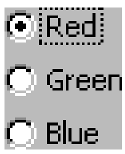
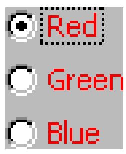
Before	After
	

Table 19.787: RADIO_SetTextColor() before after screenshots

Description

Sets the text color used to show the optional text beside the RADIO buttons.

Prototype

```
void RADIO_SetTextColor(RADIO_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of RADIO widget.
Color	Color used to show the text.

Table 19.788: RADIO_SetTextColor() parameter list

Additional information

For information about how to add text to a RADIO widget, refer to “RADIO_SetText()” on page 763.

RADIO_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

RADIO_SetValue()

Description

Sets the current button selection.

Prototype

```
void RADIO_SetValue(RADIO_Handle hObj, int v);
```

Parameter	Description
hObj	Handle of RADIO widget.
v	Value to be set.

Table 19.789: RADIO_SetValue() parameter list

Additional information

The topmost RADIO button in a RADIO widget always has the 0 value, the next button down is always 1, the next is 2, etc.

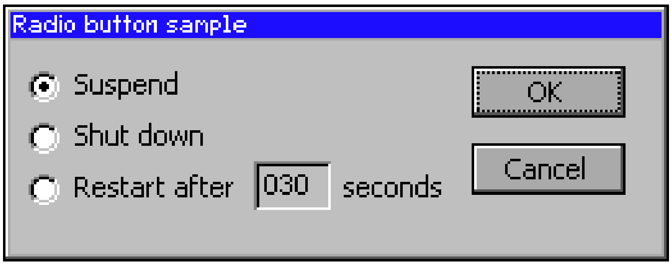
19.22.6 Examples

The `Sample` folder contains the following example which shows how the widget can be used:

- `DIALOG_Radio.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

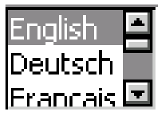
Screenshot of DIALOG_Radio.c:



19.23 SCROLLBAR: Scroll bar widget

Scroll bars are used for scrolling through list boxes or any other type of window. They may be created horizontally, as shown below, or vertically.

A scroll bar is typically attached to an existing window, for example the list box shown below:



All SCROLLBAR-related routines are located in the file(s) `SCROLLBAR*.c`, `SCROLLBAR.h`. All identifiers are prefixed SCROLLBAR.

Skinning...

...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter "Skinning" on page 871.

19.23.1 Configuration options

Type	Macro	Default	Description
N	SCROLLBAR_COLOR_SHAFT_DEFAULT	0x808080	Color of the shaft.
N	SCROLLBAR_COLOR_ARROW_DEFAULT	GUI_BLACK	Color of the arrows.
N	SCROLLBAR_COLOR_THUMB_DEFAULT	0xc0c0c0	Color of the thumb area.
N	SCROLLBAR_THUMB_SIZE_MIN_DEFAULT	4	Minimum thumb size.

Table 19.790: Configuration options

19.23.2 Predefined IDs

The following symbols define IDs which may be used to make SCROLLBAR widgets distinguishable from creation: `GUI_ID_SCROLLBAR0` - `GUI_ID_SCROLLBAR3`

19.23.3 Notification codes

The following events are sent from a scroll bar widget to its parent window as part of a `WM_NOTIFY_PARENT` message:

Message	Description
WM_NOTIFICATION_CLICKED	SCROLLBAR has been clicked.
WM_NOTIFICATION_RELEASED	SCROLLBAR has been released.
WM_NOTIFICATION_SCROLLBAR_ADDED	SCROLLBAR has just been added (attached) to an existing window. The window needs to be informed so that it can initialize the scroll bar.
WM_NOTIFICATION_VALUE_CHANGED	Value of SCROLLBAR has changed, either by moving the thumb or by pressing the arrow buttons.

Table 19.791: Notification codes

19.23.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	Increments the current value of the SCROLLBAR widget by 1.
GUI_KEY_DOWN	Increments the current value of the SCROLLBAR widget by 1.
GUI_KEY_PGDOWN	Increments the current value of the SCROLLBAR widget by a value which represents 1 page.
GUI_KEY_LEFT	Decrements the current value of the SCROLLBAR widget by 1.
GUI_KEY_UP	Decrements the current value of the SCROLLBAR widget by 1.
GUI_KEY_PGUP	Decrements the current value of the SCROLLBAR widget by a value which represents 1 page.

Table 19.792: Keyboard reaction

19.23.5 SCROLLBAR API

The table below lists the available emWin SCROLLBAR-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
SCROLLBAR_AddValue()	Increment or decrement the value of the SCROLLBAR widget by a specified value.
SCROLLBAR_Create()	Creates a SCROLLBAR widget. (Obsolete)
SCROLLBAR_CreateAttached()	Creates a SCROLLBAR widget attached to a window.
SCROLLBAR_CreateEx()	Creates a SCROLLBAR widget.
SCROLLBAR_CreateIndirect()	Creates a SCROLLBAR widget from resource table entry.
SCROLLBAR_CreateUser()	Creates a SCROLLBAR widget using extra bytes as user data.
SCROLLBAR_Dec()	Decrements the value of the SCROLLBAR widget by a value of 1.
SCROLLBAR_GetDefaultWidth()	Returns the default width of a SCROLLBAR widget.
SCROLLBAR_GetNumItems()	Returns the number of items.
SCROLLBAR_GetPageSize()	Returns the page size (in number of items).
SCROLLBAR_GetThumbSizeMin()	Returns the minimal thumb size in pixels.
SCROLLBAR_GetUserData()	Retrieves the data set with SCROLLBAR_SetUserData().
SCROLLBAR_GetValue()	Returns the current item value.
SCROLLBAR_Inc()	Increments the value of the scroll bar by a value of 1.
SCROLLBAR_SetColor()	Sets the color of a SCROLLBAR widget.
SCROLLBAR_SetDefaultColor()	Sets the default colors for new SCROLLBAR widgets.
SCROLLBAR_SetDefaultWidth()	Sets the default width of SCROLLBAR widgets.
SCROLLBAR_SetNumItems()	Sets the number of items for scrolling.
SCROLLBAR_SetPageSize()	Sets the page size (in number of items).
SCROLLBAR_SetState()	Sets the state of a SCROLLBAR widget.
SCROLLBAR_SetThumbSizeMin()	Sets the minimal thumb size in pixels.
SCROLLBAR_SetUserData()	Sets the extra data of a SCROLLBAR widget.
SCROLLBAR_SetValue()	Sets the current value of the SCROLLBAR widget.
SCROLLBAR_SetWidth()	Sets the width of the SCROLLBAR widget.

Table 19.793: SCROLLBAR API list

SCROLLBAR_AddValue()

Definition

Increments or decrements the value of the SCROLLBAR widget by a specified value.

Prototype

```
void SCROLLBAR_AddValue(SCROLLBAR_Handle hObj, int Add);
```

Parameter	Description
hObj	Handle of SCROLLBAR.
Add	Number of items to increment or decrement at one time.

Table 19.794: SCROLLBAR_AddValue() parameter list

Additional information

The SCROLLBAR widget cannot exceed the value set in SCROLLBAR_SetNumItems(). For example, if a window contains 200 items and the scroll bar is currently at value 195, incrementing the bar by 3 items will move it to value 198. However, incrementing by 10 items will only move the bar as far as value 200, which is the maximum value for this particular window.

SCROLLBAR_Create()

(Obsolete, SCROLLBAR_CreateEx() should be used instead)

Description

Creates a SCROLLBAR widget of a specified size at a specified location.

Prototype

```
SCROLLBAR_Handle SCROLLBAR_Create(int     x0,       int y0,
                                  int     xSize,    int ySize
                                  WM_HWIN hParent,  int Id,
                                  int     WinFlags, int SpecialFlags);
```

Parameter	Description
x0	Leftmost pixel of the SCROLLBAR widget (in parent coordinates).
y0	Topmost pixel of the SCROLLBAR widget (in parent coordinates).
xSize	Horizontal size of the SCROLLBAR widget (in pixels).
ySize	Vertical size of the SCROLLBAR widget (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter “The Window Manager (WM)” on page 375 for a list of available parameter values).
SpecialFlags	Special creation flags. Permitted values are listed under “SCROLLBAR_CreateEx()” .

Table 19.795: SCROLLBAR_Create() parameter list

Return value

Handle of the created SCROLLBAR widget; 0 if the function fails.

SCROLLBAR_CreateAttached()

Description

Creates a scroll bar which is attached to an existing window.

Prototype

```
SCROLLBAR_Handle SCROLLBAR_CreateAttached(WM_HWIN hParent,
                                          int     SpecialFlags);
```


Parameter	Description
hParent	Handle of parent window.
SpecialFlags	Special creation flags. Permitted values are listed under "SCROLLBAR_CreateEx()" .

Table 19.796: SCROLLBAR_CreateAttached() parameter list

Return value

Handle of the created SCROLLBAR widget; 0 if the function fails.

Additional information

An attached SCROLLBAR widget is essentially a child window which will position itself on the parent window and operate accordingly.
Vertical attached SCROLLBAR widgets will be automatically placed on the right side of the parent window; horizontal SCROLLBAR widgets on the bottom. Since no more than one horizontal and one vertical SCROLLBAR widget can be attached to a parent window, no ID needs to be passed as parameter. The following fixed ID's will automatically be assigned when an attached SCROLLBAR widget is created:
`GUI_ID_HSCROLL` for a horizontal SCROLLBAR widget, and
`GUI_ID_VSCROLL` for a vertical SCROLLBAR widget.

Example

Creates a list box with an attached SCROLLBAR widget:

```
LISTBOX_Handle hListBox;
hListBox = LISTBOX_Create(ListBox, 50, 50, 100, 100, WM_CF_SHOW);
SCROLLBAR_CreateAttached(hListBox, SCROLLBAR_CF_VERTICAL);
```

Screenshots of above example

The picture on the left shows the list box as it appears after creation. On the right it is shown with the attached vertical scroll bar:

Before	After
	

Table 19.797: SCROLLBAR_CreateAttached() before after screenshots

SCROLLBAR_CreateEx()

Description

Creates a SCROLLBAR widget of a specified size at a specified location.

Prototype

```
SCROLLBAR_Handle SCROLLBAR_CreateEx(int      x0,       int y0,
                                    int      xSize,    int ySize,
                                    WM_HWIN hParent,  int WinFlags,
                                    int      ExFlags,  int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).

Table 19.798: SCROLLBAR_CreateEx() parameter list

Parameter	Description
hParent	Handle of parent window. If 0, the new SCROLLBAR widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	OR-combination of special creation flags. See permitted values below.
Id	Window ID of the widget.

Table 19.798: SCROLLBAR_CreateEx() parameter list

Permitted values for parameter ExFlags	
SCROLLBAR_CF_VERTICAL	Creates a vertical SCROLLBAR widget.
SCROLLBAR_CF_FOCUSSABLE	Creates a focusable SCROLLBAR widget.

Return value

Handle of the created SCROLLBAR widget; 0 if the function fails.

SCROLLBAR_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Para` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The element `Flags` is used according to the parameter `ExFlags` of the function `SCROLLBAR_CreateEx()`.

SCROLLBAR_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `SCROLLBAR_CreateEx()` can be referred to.

SCROLLBAR_Dec()

Description

Decrements the current value of the given SCROLLBAR widget by a value of 1.

Prototype

```
void SCROLLBAR_Dec(SCROLLBAR_Handle hObj);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.

Table 19.799: SCROLLBAR_Dec() parameter list

Additional information

The definition of an "item" is application-specific, although in most cases it is equal to one line. Items are numbered top to bottom or left to right, beginning with a value of 0.

SCROLLBAR_GetDefaultWidth()

Description

Returns the default width used to create a SCROLLBAR widget.

Prototype

```
int SCROLLBAR_GetDefaultWidth(void);
```

Return value

Default width used to create a SCROLLBAR widget.

SCROLLBAR_GetNumItems()

Description

Returns the number of SCROLLBAR items.

Prototype

```
int SCROLLBAR_GetNumItems(SCROLLBAR_Handle hObj);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget

Table 19.800: SCROLLBAR_GetNumItems() parameter list

Return value

The number of SCROLLBAR items.

SCROLLBAR_GetPageSize()

Description

Returns the page size.

Prototype

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.

Table 19.801: SCROLLBAR_GetPageSize() parameter list

Return value

The number of items specified to be one page.

SCROLLBAR_GetThumbSizeMin()

Description

Returns the minimum thumb size in pixels.

Prototype

```
int SCROLLBAR_GetThumbSizeMin(void);
```

Return value

Minimum thumb size in pixels.

SCROLLBAR_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

SCROLLBAR_GetValue()

Description

Returns the value of the current item.

Prototype

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.

Table 19.802: SCROLLBAR_GetValue() parameter list

Return value

The value of the current item.

SCROLLBAR_Inc()

Description

Increments the current value of the given SCROLLBAR widget by a value of 1.

Prototype

```
void SCROLLBAR_Inc(SCROLLBAR_Handle hObj);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.

Table 19.803: SCROLLBAR_Inc() parameter list

Additional information

The definition of an "item" is application-specific, although in most cases it is equal to one line. Items are numbered top to bottom or left to right, beginning with a value of 0.

SCROLLBAR_SetColor()

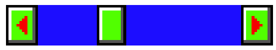
Before	After
	

Table 19.804: SCROLLBAR_SetColor() before after screenshots

Description

Sets the color attribute of the given SCROLLBAR widget.

Prototype

```
GUI_COLOR SCROLLBAR_SetColor(SCROLLBAR_Handle hObj, int Index,
                             GUI_COLOR        Color);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.
Index	See table below.
Color	Color to be used.

Table 19.805: SCROLLBAR_SetColor() parameter list

Permitted values for parameter Index	
SCROLLBAR_CI_THUMB	Color of thumb area.
SCROLLBAR_CI_SHAFT	Color of shaft.
SCROLLBAR_CI_ARROW	Color of arrows.

Return value

Previous color used for the given index.

SCROLLBAR_SetDefaultColor()

Description

Sets the default color attributes for new SCROLLBAR widgets.

Prototype

```
GUI_COLOR SCROLLBAR_SetDefaultColor(GUI_COLOR Color, unsigned int Index);
```

Parameter	Description
Color	Color used as default for newly created SCROLLBAR widgets.
Index	(see table under SCROLLBAR_SetColor())

Table 19.806: SCROLLBAR_SetDefaultColor() parameter list

Return value

Previous default color.

SCROLLBAR_SetDefaultWidth()

Description

Sets the default width used to create a SCROLLBAR widget.

Prototype

```
int SCROLLBAR_SetDefaultWidth(int DefaultWidth);
```

Parameter	Description
DefaultWidth	Default width to use for new SCROLLBAR widgets.

Table 19.807: SCROLLBAR_SetDefaultWidth() parameter list

Return value

Previous default width.

SCROLLBAR_SetNumItems()

Description

Sets the number of items for scrolling.

Prototype

```
void SCROLLBAR_SetNumItems(SCROLLBAR_Handle hObj, int NumItems);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.
NumItems	Number of items to be set.

Table 19.808: SCROLLBAR_SetNumItems() parameter list

Additional information

The definition of an "item" is application-specific, although in most cases it is equal to one line.
The number of items is the maximum value. The SCROLLBAR widget can not go beyond this value.

SCROLLBAR_SetPageSize()

Description

Sets the page size.

Prototype

```
void SCROLLBAR_SetPageSize(SCROLLBAR_Handle hObj, int PageSize);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.
PageSize	Page size (in number of items).

Table 19.809: SCROLLBAR_SetPageSize() parameter list

Additional information

Page size is specified as the number of items to one page. If the user pages up or down, either with the keyboard or by mouse-clicking in the SCROLLBAR area, the window will be scrolled up or down by the number of items specified as one page.

SCROLLBAR_SetState()

Description

Sets the state of a SCROLLBAR widget.

Prototype

```
void SCROLLBAR_SetState(SCROLLBAR_Handle         hObj,
                        const WM_SCROLL_STATE * pState);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.
pState	Pointer to a data structure of type WM_SCROLL_STATE.

Table 19.810: SCROLLBAR_SetState() parameter list

Additional information

The data structure is defined as follows:

```
typedef struct {
  int NumItems;
  int v;
  int PageSize;
} WM_SCROLL_STATE;
```

SCROLLBAR_SetThumbSizeMin()

Description

Sets the minimum thumb size in pixels.

Prototype

```
int SCROLLBAR_SetThumbSizeMin(int ThumbSizeMin);
```

Parameter	Description
ThumbSizeMin	Minimum thumb size to be set.

Table 19.811: SCROLLBAR_SetThumbSizeMin() parameter list

Return value

Old minimum thumb size in pixels.

SCROLLBAR_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

SCROLLBAR_SetValue()

Description

Sets the value of the given SCROLLBAR widget.

Prototype

```
void SCROLLBAR_SetValue(SCROLLBAR_Handle hObj, int v);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.
v	Value to be set.

Table 19.812: SCROLLBAR_SetValue() parameter list

SCROLLBAR_SetWidth()

Description

Sets the width of the given SCROLLBAR widget.

Prototype

```
void SCROLLBAR_SetWidth(SCROLLBAR_Handle hObj, int Width);
```

Parameter	Description
hObj	Handle of a SCROLLBAR widget.
Width	Width to be set.

Table 19.813: SCROLLBAR_SetWidth() parameter list

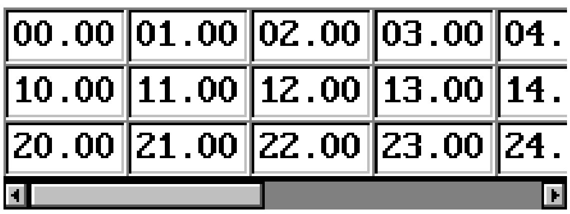
19.23.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_ScrollbarMove.c`

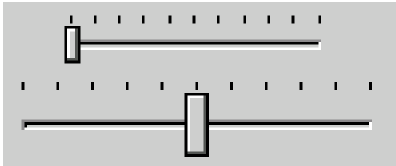
Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_ScrollbarMove.c:



19.24 SLIDER: Slider widget

SLIDER widgets are commonly used for modifying values through the use of a slider bar. The widget consists of a slider bar and tick marks beside the bar. These tick marks can be used to snap the slider bar while dragging it. For details about how to use the tick marks for snapping refer to the function `SLIDER_SetRange()`.



All SLIDER-related routines are located in the file(s) `SLIDER*.c`, `SLIDER.h`. All identifiers are prefixed SLIDER.

Skinning...

...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter “Skinning” on page 871.

19.24.1 Configuration options

Type	Macro	Default	Description
N	`SLIDER_BKCOLOR0_DEFAULT`	0xc0c0c0	Background color.
N	`SLIDER_COLOR0_DEFAULT`	0xc0c0c0	Slider (thumb) color.
N	`SLIDER_FOCUSCOLOR_DEFAULT`	GUI_BLACK	Default color for rendering the focus rectangle.

Table 19.814: Configuration options

19.24.2 Predefined IDs

The following symbols define IDs which may be used to make SLIDER widgets distinguishable from creation: `GUI_ID_SLIDER0` - `GUI_ID_SLIDER9`

19.24.3 Notification codes

The following events are sent from a SLIDER widget to its parent window as part of a `WM_NOTIFY_PARENT` message:

Message	Description
`WM_NOTIFICATION_CLICKED`	SLIDER widget has been clicked.
`WM_NOTIFICATION_RELEASED`	SLIDER widget has been released.
`WM_NOTIFICATION_VALUE_CHANGED`	Value of the SLIDER widget has changed by moving the thumb.

Table 19.815: Notification codes

19.24.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
`GUI_KEY_RIGHT`	Increments the current value of the SLIDER widget by one item.
`GUI_KEY_LEFT`	Decrements the current value of the SLIDER widget by one item.

Table 19.816: Keyboard reaction

19.24.5 SLIDER API

The table below lists the available emWin SLIDER-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
SLIDER_Create()	Creates a SLIDER widget. (Obsolete)
SLIDER_CreateEx()	Creates a SLIDER widget.
SLIDER_CreateIndirect()	Creates a SLIDER widget from resource table entry.
SLIDER_CreateUser()	Creates a SLIDER widget using extra bytes as user data.
SLIDER_Dec()	Decrement the value of the SLIDER widget.
SLIDER_GetUserData()	Retrieves the data set with SLIDER_SetUserData().
SLIDER_GetValue()	Return the current value of the SLIDER widget.
SLIDER_Inc()	Increment the value of the SLIDER widget.
SLIDER_SetBkColor()	Sets the background color of the SLIDER widget.
SLIDER_SetDefaultFocusColor()	Sets the default focus rectangle color for new SLIDER widgets.
SLIDER_SetFocusColor()	Sets the color of the focus rectangle.
SLIDER_SetNumTicks()	Sets the number of tick marks of the SLIDER widget.
SLIDER_SetRange()	Set the range of the SLIDER widget.
SLIDER_SetUserData()	Sets the extra data of a SLIDER widget.
SLIDER_SetValue()	Set the current value of the SLIDER widget.
SLIDER_SetWidth()	Set the width of the SLIDER widget.

Table 19.817: SLIDER API list

SLIDER_Create()

(Obsolete, SLIDER_CreateEx() should be used instead)

Description

Creates a SLIDER widget of a specified size at a specified location.

Prototype

```
SLIDER_Handle SLIDER_Create(int      x0,       int y0,
                            int      xSize,    int ySize,
                            WM_HWIN  hParent,  int Id,
                            int      WinFlags, int SpecialFlags);
```

Parameter	Description
x0	Leftmost pixel of the slider (in parent coordinates).
y0	Topmost pixel of the slider (in parent coordinates).
xSize	Horizontal size of the slider (in pixels).
ySize	Vertical size of the slider (in pixels).
hParent	Handle of the parent window.
Id	Id to be returned
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
SpecialFlags	Special creation flag (see indirect creation flag under SLIDER_CreateIndirect()).

Table 19.818: SLIDER_Create() parameter list

Return value

Handle of the created SLIDER widget; 0 if the function fails.

SLIDER_CreateEx()

Description

Creates a SLIDER widget of a specified size at a specified location.

Prototype

```
SLIDER_Handle SLIDER_CreateEx(int      x0,       int y0,
                              int      xSize,    int ySize,
                              WM_HWIN hParent, int WinFlags,
                              int      ExFlags, int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of the parent window. If 0, the new SLIDER widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	Special creation flags. See table below.
Id	Window ID of the widget.

Table 19.819: SLIDER_CreateEx() parameter list

Permitted values for parameter ExFlags	
SLIDER_CF_VERTICAL	Create a vertical slider (default is horizontal).

Return value

Handle of the created SLIDER widget; 0 if the function fails.

SLIDER_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The element Para of the according GUI_WIDGET_CREATE_INFO structure is not used. The element Flags is used according to the parameter ExFlags of the function SLIDER_CreateEx().

SLIDER_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function SLIDER_CreateEx() can be referred to.

SLIDER_Dec()

Description

Decrements the current value of the SLIDER widget by one item.

Prototype

```
void SLIDER_Dec(SLIDER_Handle hObj);
```

Parameter	Description
hObj	Handle of SLIDER widget.

Table 19.820: SLIDER_Dec() parameter list

SLIDER_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

SLIDER_GetValue()

Description

Returns the current value of the SLIDER widget.

Prototype

```
int SLIDER_GetValue(SLIDER_Handle hObj);
```

Parameter	Description
hObj	Handle of SLIDER widget.

Table 19.821: SLIDER_GetValue() parameter list

Return value

The current value of the SLIDER widget.

SLIDER_Inc()

Description

Increments the current value of the SLIDER widget by one item.

Prototype

```
void SLIDER_Inc(SLIDER_Handle hObj);
```

Parameter	Description
hObj	Handle of SLIDER widget.

Table 19.822: SLIDER_Inc() parameter list

SLIDER_SetBkColor()

Description

Sets the background color of the SLIDER widget.

Prototype

```
void SLIDER_SetBkColor(SLIDER_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of SLIDER widget.
Color	Color to be used for the background. (range 0x000000 and 0xFFFFFF or a valid color define) GUI_INVALID_COLOR to make background transparent

Table 19.823: SLIDER_SetBkColor() parameter list

Additional information

The background of this widget can either be filled with any available color or transparent. If a valid RGB color is specified, the background is filled with the color, otherwise the background (typically the content of the parent window) is visible. If the background is transparent, the widget is treated as transparent window, otherwise as non-transparent window. Note that using a background color allows more efficient (faster) rendering.

This widget is per default a transparent window. The appearance of a transparent windows background depends on the appearance of the parent window. When a transparent window needs to be redrawn first the background will be drawn by sending a `WM_PAINT` message to the parent window.
If using this function with a valid color the status of the window will be changed from

transparent to non transparent and if the window needs to be redrawn the background will be filled with the given color.
If `GUI_INVALID_COLOR` is passed to the function the status will be changed from non transparent to transparent.

SLIDER_SetDefaultFocusColor()

Description

Sets the default focus rectangle color for new SLIDER widgets.

Prototype

```
GUI_COLOR SLIDER_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Description
Color	Default color to be used for new slider bars.

Table 19.824: SLIDER_SetDefaultFocusColor() parameter list

Return value

Previous default focus rectangle color.

Additional information

For more information, refer to "SLIDER_SetFocusColor()" on page 779.

SLIDER_SetFocusColor()

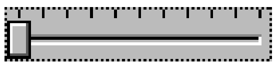
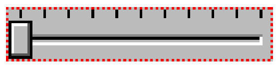
Before	After
	

Table 19.825: SLIDER_SetFocusColor() before after screenshots

Description

Sets the color used to render the focus rectangle of the SLIDER widget.

Prototype

```
GUI_COLOR SLIDER_SetFocusColor(SLIDER_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of SLIDER widget.
Color	Color to be used for the focus rectangle.

Table 19.826: SLIDER_SetFocusColor() parameter list

Return value

Previous color of the focus rectangle.

Additional information

The focus rectangle is only visible if the widget has the input focus.

SLIDER_SetNumTicks()

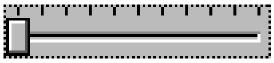
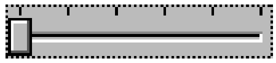
Before	After
	

Table 19.827: SLIDER_SetNumTicks() before after screenshots

Description

Sets the number of tick marks of the SLIDER widget.

Prototype

```
void SLIDER_SetNumTicks(SLIDER_Handle hObj, int NumTicks);
```

Parameter	Description
hObj	Handle of SLIDER widget.
NumTicks	Number of tick marks drawn.

Table 19.828: SLIDER_SetNumTicks() parameter list

Additional information

After creating a SLIDER widget the default number of tick marks is 10. The tick marks have no effect to snap the slider bar while dragging it.

SLIDER_SetRange()

Description

Sets the range of the SLIDER widget.

Prototype

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

Parameter	Description
hObj	Handle of SLIDER widget.
Min	Minimum value.
Max	Maximum value.

Table 19.829: SLIDER_SetRange() parameter list

Additional information

After creating a SLIDER widget the default range is set to 0 - 100.

Examples

If a value should be modified in the range of 0 - 2499 set the range as follows:

```
SLIDER_SetRange(hSlider, 0, 2499);
```

If a value should be modified in the range of 100 - 499 set the range as follows:

```
SLIDER_SetRange(hSlider, 100, 499);
```

If a value should be modified in the range of 0 to 5000 and the slider bar should change the value in steps of 250 set the range and the tick marks as follows. The result returned by `SLIDER_GetValue()` should be multiplied with 250:

```
SLIDER_SetRange(hSlider, 0, 20);
SLIDER_SetNumTicks(hSlider, 21);
```


SLIDER_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

SLIDER_SetValue()

Description

Sets the current value of the SLIDER widget.

Prototype

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

Parameter	Description
`hObj`	Handle of SLIDER widget.
`v`	Value to be set.

Table 19.830: SLIDER_SetValue() parameter list

SLIDER_SetWidth()

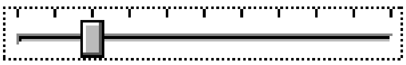
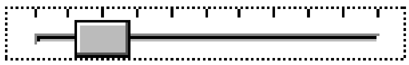
Before	After
	

Table 19.831: SLIDER_SetWidth() before after screenshots

Description

Sets the width of the SLIDER widget.

Prototype

```
void SLIDER_SetWidth(SLIDER_Handle hObj, int Width);
```

Parameter	Description
`hObj`	Handle of SLIDER widget.
`Width`	Width to be set.

Table 19.832: SLIDER_SetWidth() parameter list

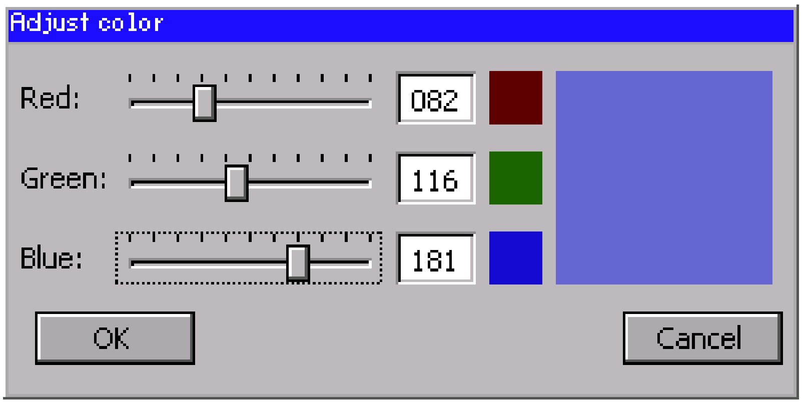
19.24.6 Example

The `Sample` folder contains the following example which shows how the widget can be used:

- `DIALOG_SliderColor.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of DIALOG_SliderColor.c:

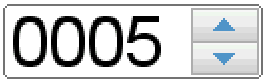


19.25 SPINBOX: Spinning box widget

SPINBOX widgets are used to manage values which need to be adjustable in a fast but still precise manner. A SPINBOX consists of 2 buttons and an embedded EDIT widget.

All SPINBOX-related routines are located in the file(s) `SPINBOX*.c` and `SPINBOX.h`. All identifiers are prefixed SPINBOX.

Skinning...



...is available for this widget. The screenshot above shows the widget using the default skin. Details can be found in the chapter “Skinning” on page 871.

19.25.1 Configuration options

Type	Macro	Default	Description
N	SPINBOX_DEFAULT_BUTTON_BKCOLOR0	0xAAAAAA	Background color for the button state disabled.
N	SPINBOX_DEFAULT_BUTTON_BKCOLOR1	GUI_WHITE	Background color for the button state pressed.
N	SPINBOX_DEFAULT_BUTTON_BKCOLOR2	GUI_LIGHTGRAY	Background color for the button state unpressed.
N	SPINBOX_DEFAULT_BUTTON_UCOLOR0	0xAAAAAA	Background color for the button state disabled.
N	SPINBOX_DEFAULT_BUTTON_UCOLOR1	GUI_WHITE	Background color for the button state pressed.
N	SPINBOX_DEFAULT_BUTTON_UCOLOR2	GUI_LIGHTGRAY	Background color for the button state unpressed.
N	SPINBOX_DEFAULT_BUTTON_LCOLOR0	0xAAAAAA	Background color for the button state disabled.
N	SPINBOX_DEFAULT_BUTTON_LCOLOR1	GUI_WHITE	Background color for the button state pressed.
N	SPINBOX_DEFAULT_BUTTON_LCOLOR2	GUI_LIGHTGRAY	Background color for the button state unpressed.
N	SPINBOX_DEFAULT_BUTTON_OCOLOR0	0xAAAAAA	Background color for the button state disabled.
N	SPINBOX_DEFAULT_BUTTON_OCOLOR1	GUI_WHITE	Background color for the button state pressed.
N	SPINBOX_DEFAULT_BUTTON_OCOLOR2	GUI_LIGHTGRAY	Background color for the button state unpressed.
N	SPINBOX_DEFAULT_BKCOLOR0	0xC0C0C0	Background color for the edit state enabled.
N	SPINBOX_DEFAULT_BKCOLOR1	GUI_WHITE	Background color for the edit state disabled.
N	SPINBOX_DEFAULT_TEXTCOLOR0	0xC0C0C0	Background color for the edit state enabled.
N	SPINBOX_DEFAULT_TEXTCOLOR1	GUI_WHITE	Background color for the edit state disabled.
N	SPINBOX_DEFAULT_TRIANGLE_COLOR0	0xAAAAAA	Background color for the button state disabled.
N	SPINBOX_DEFAULT_TRIANGLE_COLOR1	GUI_WHITE	Background color for the button state pressed.
N	SPINBOX_DEFAULT_TRIANGLE_COLOR2	GUI_LIGHTGRAY	Background color for the button state unpressed.

Table 19.833: Configuration options

Type	Macro	Default	Description
N	SPINBOX_DEFAULT_STEP	1	Value will be increased/ decreased by this amount when a button is clicked.
N	SPINBOX_DEFAULT_BUTTON_SIZE	0	X-Size of the buttons.
N	SPINBOX_DEFAULT_EDGE	SPINBOX_EDGE_RIGHT	Determines the position of the buttons. See table below.
N	SPINBOX_TIMER_PERIOD_START	400	Once a button is pressed for this amount of time, a timer is created to increase/ decrease the value continu-ously.
N	SPINBOX_TIMER_PERIOD	50	Once the timer is created values are adjusted at inter-vals of this amount of time.

Table 19.833: Configuration options

Possible values to be defined as SPINBOX_DEFAULT_EDGE	
SPINBOX_EDGE_LEFT	Buttons are displayed on the left edge of the widget.
SPINBOX_EDGE_RIGHT	Buttons are displayed on the right edge of the widget.

19.25.2 Predefined IDs

The following symbols define IDs which may be used to make SPINBOX widgets distinguishable from creation: GUI_ID_SPINBOX0 - GUI_ID_SPINBOX9

19.25.3 Notification codes

The following events are sent from the spinbox widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	Button has been clicked.
WM_NOTIFICATION_RELEASED	Button has been released.
WM_NOTIFICATION_MOVED_OUT	Pointer has been moved out of the widget area.
WM_NOTIFICATION_VALUE_CHANGED	The value of the SPINBOX widget has changed.

Table 19.834: Notification codes

19.25.4 Keyboard reaction

The widget is able to receive the input focus. All key events are forwarded to the embedded edit widget. Detailed information can be taken from the EDIT widget section.

19.25.5 SPINBOX API

The table below lists the available emWin SPINBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
SPINBOX_CreateEx()	Creates a SPINBOX widget.
SPINBOX_CreateIndirect()	Creates a SPINBOX widget. (Obsolete)
SPINBOX_CreateUser()	Creates a SPINBOX widget using extra bytes as user data.
SPINBOX_EnableBlink()	Enables/disables blinking of the cursor.
SPINBOX_GetBkColor()	Returns the background color of the SPINBOX widget.
SPINBOX_GetButtonBkColor()	Returns the background color of the buttons.
SPINBOX_GetDefaultButtonSize()	Returns the default x-size of the buttons.
SPINBOX_GetEditHandle()	Returns the handle to the attached EDIT widget.
SPINBOX_GetUserData()	Retrieves the data which was previously set with SPINBOX_SetUserData().
SPINBOX_GetValue()	Returns the value of the SPINBOX widget.
SPINBOX_SetBkColor()	Sets the background color of the SPINBOX widget.
SPINBOX_SetButtonBkColor()	Sets the background color of the buttons.
SPINBOX_SetButtonSize()	Sets the size of the button.
SPINBOX_SetDefaultButtonSize()	Sets the default x-size of the buttons.
SPINBOX_SetEdge()	Sets the edge to display the buttons on.
SPINBOX_SetEditMode()	Sets 'Step' or 'Edit' mode.
SPINBOX_SetFont()	Sets the font used to display the value.
SPINBOX_SetRange()	Sets the minimum and maximum value.
SPINBOX_SetStep()	Sets the step value to be used for the step mode.
SPINBOX_SetTextColor()	Sets the color of the displayed value.
SPINBOX_SetUserData()	Stores user data using the extra bytes which were reserved by SPINBOX_CreateUser().
SPINBOX_SetValue()	Sets the value of the SPINBOX.

Table 19.835: SPINBOX API list

SPINBOX_CreateEx()

Description

Creates a SPINBOX widget.

Prototype

```
SPINBOX_Handle SPINBOX_CreateEx(int x0,     int      y0,       int xSize,
                                int ySize, WM_HWIN hParent, int WinFlags,
                                int Id,     int      Min,      int Max);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of the parent window. If 0, the widget will be created as a child of the top-level window (desktop).
WinFlags	Window create flags. In order to make the widget visible immediately WM_CF_SHOW should be used. The complete list of available parameters can be found under "WM_CreateWindow()" on page 404.
Id	Window ID to be set for the widget.
Min	Minimum permitted value.
Max	Maximum permitted value.

Table 19.836: SPINBOX_CreateEx() parameter list

Return value

Handle of the created SPINBOX widget. If an error occurred during creation, 0 is returned.

SPINBOX_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function `<WIDGET>_CreateIndirect()`. The element `Flags` of the according `GUI_WIDGET_CREATE_INFO` structure is not used. The upper 16 bit of the element `Para` are used according to the parameter `Max` of the function `SPINBOX_CreateEx()`. The lower 16 bit of the element `Para` are used according to the parameter `Min` of the function `SPINBOX_CreateEx()`.

SPINBOX_CreateUser()

Prototype explained at the beginning of the chapter as `<WIDGET>_CreateUser()`. For a detailed description of the parameters the function `SPINBOX_CreateEx()` can be referred to.

SPINBOX_EnableBlink()

Description

Enables/disables blinking of the cursor.

Prototype

```
void SPINBOX_EnableBlink(SPINBOX_Handle hObj, int Period, int OnOff);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
Period	Period in which the cursor is turned off and on.
OnOff	1 enables blinking, 0 disables blinking.

Table 19.837: SPINBOX_EnableBlink() parameter list

SPINBOX_GetBkColor()

Description

Returns the background color of the SPINBOX widget.

Prototype

```
GUI_COLOR SPINBOX_GetBkColor(SPINBOX_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
Index	Color index. See table below.

Table 19.838: SPINBOX_GetBkColor() parameter list

Permitted values for parameter Index	
SPINBOX_CI_DISABLED	Color for disabled state.
SPINBOX_CI_ENABLED	Color for enabled state.

Return value

Background color of the SPINBOX widget.

SPINBOX_GetButtonBkColor()

Description

Returns the background color of the buttons.

Prototype

```
GUI_COLOR SPINBOX_GetButtonBkColor(SPINBOX_Handle hObj, unsigned int Index);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
Index	Color index. See table below.

Table 19.839: SPINBOX_GetButtonBkColor() parameter list

Permitted values for parameter Index	
SPINBOX_CI_DISABLED	Color for disabled state.
SPINBOX_CI_ENABLED	Color for enabled state.
SPINBOX_CI_PRESSED	Color for pressed state.

Return value

Background color of the buttons.

SPINBOX_GetDefaultButtonSize()

Description

Returns the default x-size of the buttons.

Prototype

```
U16 SPINBOX_GetDefaultButtonSize(void);
```

Return value

Default x-size of the buttons.

SPINBOX_GetEditHandle()

Description

Returns the handle to the attached EDIT widget.

Prototype

```
EDIT_Handle SPINBOX_GetEditHandle(SPINBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.

Table 19.840: SPINBOX_GetEditHandle() parameter list

Return value

Handle of the attached EDIT widget.

SPINBOX_GetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_GetUserData().

SPINBOX_GetValue()

Description

Returns the value of the SPINBOX widget.

Prototype

```
int SPINBOX_GetValue(SPINBOX_Handle hObj);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.

Table 19.841: SPINBOX_GetValue() parameter list

Return value

Value of the SPINBOX widget.

SPINBOX_SetBkColor()

Before	After
	

Table 19.842: SPINBOX_SetBkColor() before after screenshots

Description

Sets the background color of the SPINBOX widget.

Prototype

```
void SPINBOX_SetBkColor(SPINBOX_Handle hObj,   unsigned int Index,
                        GUI_COLOR      Color);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
Index	Color index. See table below.
Color	Color to be used for the background.

Table 19.843: SPINBOX_SetBkColor() parameter list

Permitted values for parameter Index	
SPINBOX_CI_DISABLED	Color for disabled state.
SPINBOX_CI_ENABLED	Color for enabled state.

SPINBOX_SetButtonBkColor()

Before	After
	

Table 19.844: SPINBOX_SetButtonBkColor() before after screenshots

Description

Sets the background color of the buttons.

Prototype

```
void SPINBOX_SetButtonBkColor(SPINBOX_Handle hObj,   unsigned int Index,
                              GUI_COLOR      Color);
```

Parameter	Description
`hObj`	Handle of the SPINBOX widget.
`Index`	Color index. See table below.
`Color`	Color to be used for the background.

Table 19.845: SPINBOX_SetButtonBkColor() parameter list

Permitted values for parameter `Index`	
`SPINBOX_CI_DISABLED`	Color for disabled state.
`SPINBOX_CI_ENABLED`	Color for enabled state.
`SPINBOX_CI_PRESSED`	Color for pressed state.

SPINBOX_SetButtonSize()

Before	After
	

Table 19.846: SPINBOX_SetButtonSize() before after screenshots

Description

Sets the button size of the given SPINBOX widget.

Prototype

```
void SPINBOX_SetButtonSize(SPINBOX_Handle hObj, unsigned ButtonSize);
```

Parameter	Description
`hObj`	Handle of the SPINBOX widget.
`ButtonSize`	Button size in pixels to be used.

Table 19.847: SPINBOX_SetButtonSize() parameter list

SPINBOX_SetDefaultButtonSize()

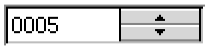
Before	After
	

Table 19.848: SPINBOX_SetDefaultButtonSize() before after screenshots

Description

Sets the default x-size of the buttons.

Prototype

```
void SPINBOX_SetDefaultButtonSize(U16 x);
```

Parameter	Description
x	New default x-size of the buttons.

Table 19.849: SPINBOX_SetDefaultButtonSize() parameter list

Additional information

If the default button size is set to 0, the size of the button is determined automatically on creation.

SPINBOX_SetEdge()

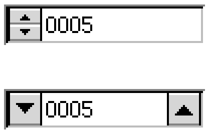
Before	After
	

Table 19.850: SPINBOX_SetEdge() before after screenshots

Description

Sets the edge to display the buttons on.

Prototype

```
void SPINBOX_SetEdge(SPINBOX_Handle hObj, U8 Edge);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
Edge	See table below.

Table 19.851: SPINBOX_SetEdge() parameter list

Permitted values for parameter Edge	
SPINBOX_EDGE_CENTER	Buttons are displayed on the left and the right edge of the widget.
SPINBOX_EDGE_LEFT	Buttons are displayed on the left edge of the widget.
SPINBOX_EDGE_RIGHT	Buttons are displayed on the right edge of the widget.

SPINBOX_SetEditMode()

Description

The widget supports the 'Step' mode (default) and the 'Edit' mode. 'Step' mode means each time one of the buttons is pressed the value of the box is incremented or decremented by the step value. 'Edit' mode means each single digit can be modified separately. If 'Edit' mode is selected the EDIT field becomes editable and a cursor appears. When pressing a button the focussed digit will be incremented or decremented by 1. In 'Edit' mode the EDIT field also accepts digits as keyboard input.

Prototype

```
void SPINBOX_SetEditMode(SPINBOX_Handle hObj, U8 EditMode);
```

Parameter	Description
`hObj`	Handle of the SPINBOX widget.
`EditMode`	See table below.

Table 19.852: SPINBOX_SetEditMode() parameter list

Permitted values for parameter `EditMode`	
`SPINBOX_EM_STEP`	Pressing a button adds or subtracts the step value.
`SPINBOX_EM_EDIT`	Pressing a button increments or decrements a digit.

SPINBOX_SetFont()

Before	After
	

Table 19.853: SPINBOX_SetFont() before after screenshots

Description

Sets the font used to display the value.

Prototype

```
void SPINBOX_SetFont(SPINBOX_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
`hObj`	Handle to the SPINBOX widget.
`pFont`	Pointer to the font to be used.

Table 19.854: SPINBOX_SetFont() parameter list

SPINBOX_SetRange()

Description

Sets the minimum and maximum value.

Prototype

```
void SPINBOX_SetRange(SPINBOX_Handle hObj, I32 Min, I32 Max);
```

Parameter	Description
`hObj`	Handle to the SPINBOX widget.
`Min`	Minimum value.
`Max`	Maximum value.

Table 19.855: SPINBOX_SetRange() parameter list

SPINBOX_SetStep()

Description

Sets the value to be used for incrementing and decrementing in 'Step' mode.

Prototype

```
U16 SPINBOX_SetStep(SPINBOX_Handle hObj, U16 Step);
```

Parameter	Description
hObj	Handle to the SPINBOX widget.
Step	Value to be used.

Table 19.856: SPINBOX_SetStep() parameter list

Return value

Previous used step value.

SPINBOX_SetTextColor()

Before	After
	

Table 19.857: SPINBOX_SetTextColor() before after screenshots

Description

Sets the color of the displayed value.

Prototype

```
void SPINBOX_SetTextColor(SPINBOX_Handle hObj,   unsigned int Index,
                          GUI_COLOR      Color);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
Index	Color index. See table below.
Color	Color to be set for the text.

Table 19.858: SPINBOX_SetTextColor() parameter list

Permitted values for parameter Index	
SPINBOX_CI_DISABLED	Color for disabled state.
SPINBOX_CI_ENABLED	Color for pressed state.

SPINBOX_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

SPINBOX_SetValue()

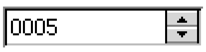
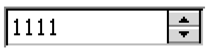
Before	After
	

Table 19.859: SPINBOX_SetValue() before after screenshots

Description

Sets the value of the SPINBOX.

Prototype

```
void SPINBOX_SetValue(SPINBOX_Handle hObj, int v);
```

Parameter	Description
hObj	Handle of the SPINBOX widget.
v	Value to be set.

Table 19.860: SPINBOX_SetValue() parameter list

19.25.6 Example

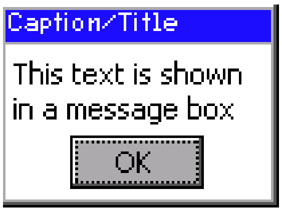
The `Sample` folder contains the following example which shows how the widget can be used:

- `WIDGET_Spinbox.c`

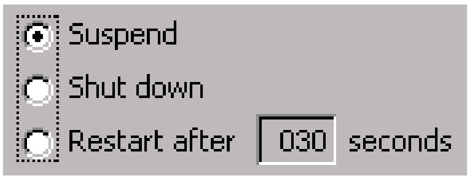
Screenshot of WIDGET_Spinbox.c:

19.26 TEXT: Text widget

Text widgets are typically used in order to display fields of text in dialog boxes, as shown in the message box below:



Of course, text fields may also be used for labeling other widgets, as follows:



All TEXT-related routines are located in the file(s) TEXT*.c, TEXT.h. All identifiers are prefixed TEXT.

19.26.1 Configuration options

Type	Macro	Default	Description
N	TEXT_DEFAULT_BK_COLOR	GUI_INVALID_COLOR	Transparent background per default
N	TEXT_DEFAULT_TEXT_COLOR	GUI_BLACK	Default text color.
N	TEXT_DEFAULT_WRAPMODE	GUI_WRAPMODE_NONE	Default wrapping mode.
S	TEXT_FONT_DEFAULT	&GUI_Font13_1	Font used.

Table 19.861: Configuration options

19.26.2 Predefined IDs

The following symbols define IDs which may be used to make TEXT widgets distinguishable from creation: GUI_ID_TEXT0 - GUI_ID_TEXT9

19.26.3 Notification codes

The following events are sent from an TEXT widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	The widget has been clicked.
WM_NOTIFICATION_RELEASED	The widget has been released.
WM_NOTIFICATION_MOVED_OUT	The pointer was moved out of the widget area while the PID was in pressed state.

Table 19.862: Notification codes

19.26.4 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

19.26.5 TEXT API

The table below lists the available emWin TEXT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
TEXT_Create()	Creates a TEXT widget. (Obsolete)
TEXT_CreateAsChild()	Creates a TEXT widget as a child window. (Obsolete)
TEXT_CreateEx()	Creates a TEXT widget.
TEXT_CreateIndirect()	Creates a TEXT widget from resource table entry.
TEXT_CreateUser()	Creates a TEXT widget using extra bytes as user data.
TEXT_GetBkColor()	Returns the current background color.
TEXT_GetDefaultFont()	Returns the default font used for text.
TEXT_GetDefaultTextColor()	Returns the default text color.
TEXT_GetDefaultWrapMode()	Returns the default mode for wrapping text.
TEXT_GetFont()	Returns the pointer to the font of the TEXT widget.
TEXT_GetNumLines()	Returns the number of lines currently displayed in the widget.
TEXT_GetText()	Copies the text of the given TEXT widget to the given buffer.
TEXT_GetTextAlign()	Returns the text alignment of the TEXT widget.
TEXT_GetTextColor()	Returns the text color of the specified TEXT widget.
TEXT_GetUserData()	Retrieves the data set with TEXT_SetUserData().
TEXT_GetWrapMode()	Returns the currently used mode for wrapping text.
TEXT_SetBkColor()	Sets the background color for the text.
TEXT_SetDefaultFont()	Sets the default font used for TEXT widgets.
TEXT_SetDefaultTextColor()	Sets the default text color used for TEXT widgets.
TEXT_SetDefaultWrapMode()	Sets the default wrap mode for new TEXT widgets.
TEXT_SetFont()	Sets the font used for a specified TEXT widget.
TEXT_SetText()	Sets the text for a specified TEXT widget.
TEXT_SetTextAlign()	Sets the text alignment of a specified TEXT widget.
TEXT_SetTextColor()	Sets the text color of the given TEXT widget.
TEXT_SetUserData()	Sets the extra data of a TEXT widget.
TEXT_SetWrapMode()	Sets the wrap mode of a specified TEXT widget.

Table 19.863: TEXT API list

TEXT_Create()

(Obsolete, `TEXT_CreateEx()` should be used instead)

Description

Creates a TEXT widget of a specified size at a specified location.

Prototype

```
TEXT_Handle TEXT_Create(int          x0,    int y0,
                        int          xSize, int ySize,
                        int          Id,    int Flags,
                        const char * s,     int Align);
```

Parameter	Description
x0	Leftmost pixel of the TEXT widget (in parent coordinates).
y0	Topmost pixel of the TEXT widget (in parent coordinates).
xSize	Horizontal size of the TEXT widget (in pixels).
ySize	Vertical size of the TEXT widget (in pixels).
Id	ID to be returned.

Table 19.864: TEXT_Create() parameter list

Parameter	Description
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
s	Pointer to the text to be displayed.
Align	Alignment attribute for the text (see indirect creation flags under TEXT_CreateIndirect()).

Table 19.864: TEXT_Create() parameter list

Return value

Handle of the created TEXT widget; 0 if the function fails.

TEXT_CreateAsChild()

(Obsolete, TEXT_CreateEx should be used instead)

Description

Creates a TEXT widget as a child window.

Prototype

```
TEXT_Handle TEXT_CreateAsChild(int      x0,      int          y0,
                               int      xSize,   int          ySize,
                               WM_HWIN hParent, int          Id,
                               int      Flags,   const char * s,
                               int      Align);
```

Parameter	Description
x0	X-position of the PROGBAR widget relative to the parent window.
y0	Y-position of the PROGBAR widget relative to the parent window.
xSize	Horizontal size of the TEXT widget (in pixels).
ySize	Vertical size of the TEXT widget (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
Flags	Window create flags (see TEXT_Create()).
s	Pointer to the text to be displayed.
Align	Alignment attribute for the text (see indirect creation flags under TEXT_CreateIndirect()).

Table 19.865: TEXT_CreateAsChild() parameter list

Return value

Handle of the created TEXT widget; 0 if the function fails.

TEXT_CreateEx()

Description

Creates a TEXT widget of a specified size at a specified location.

Prototype

```
TEXT_Handle TEXT_CreateEx(int          x0,      int y0,
                          int          xSize,   int ySize,
                          WM_HWIN      hParent, int WinFlags,
                          int          ExFlags, int Id,
```



```
const char * pText);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new TEXT widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter "The Window Manager (WM)" on page 375 for a list of available parameter values).
ExFlags	Alignment attribute for the text. See permitted values below.
Id	Window ID of the TEXT widget.
pText	Pointer to the text to be displayed.

Table 19.866: TEXT_CreateEx() parameter list

Permitted values for parameter ExFlags	
TEXT_CF_LEFT	Horizontal alignment: left
TEXT_CF_RIGHT	Horizontal alignment: right
TEXT_CF_HCENTER	Horizontal alignment: center
TEXT_CF_TOP	Vertical alignment: top
TEXT_CF_BOTTOM	Vertical alignment: bottom
TEXT_CF_VCENTER	Vertical alignment: center

Return value

Handle of the created TEXT widget; 0 if the function fails.

TEXT_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The element Para of the according GUI_WIDGET_CREATE_INFO structure is not used. The element Flags is used according to the parameter ExFlags of the function TEXT_CreateEx().

TEXT_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function TEXT_CreateEx() can be referred to.

TEXT_GetBkColor()

Description

Returns the current background color of the given TEXT widget.

Prototype

```
GUI_COLOR TEXT_GetBkColor(TEXT_Handle hObj);
```

Parameter	Description
hObj	Handle of TEXT widget.

Table 19.867: TEXT_GetBkColor() parameter list

Return value

The current background color.

TEXT_GetDefaultFont()

Description

Returns the default font used for TEXT widgets.

Prototype

```
const GUI_FONT* TEXT_GetDefaultFont(void);
```

Return value

Pointer to the default font used for TEXT widgets.

TEXT_GetDefaultTextColor()

Description

Returns the default text color used for TEXT widgets.

Prototype

```
GUI_COLOR TEXT_GetDefaultTextColor(void);
```

Return value

Default text color.

TEXT_GetDefaultWrapMode()

Description

Returns the default mode for wrapping text used for TEXT widget.

Prototype

```
GUI_WRAPMODE TEXT_GetDefaultWrapMode(void);
```

Return value

Default wrap mode.

TEXT_GetFont()

Description

Returns a pointer to the font used to display the text of the given TEXT widget.

Prototype

```
const GUI_FONT * TEXT_GetFont(TEXT_Handle hObj);
```

Parameter	Description
hObj	Handle of TEXT widget.

Table 19.868: TEXT_GetFont() parameter list

Return value

Pointer to the font used to display the text of the given TEXT widget.

TEXT_GetNumLines()

Description

Returns the number of lines currently displayed in the widget.

Prototype

```
int TEXT_GetNumLines(TEXT_Handle hObj);
```

Parameter	Description
hObj	Handle of the TEXT widget.

Table 19.869: TEXT_GetNumLines() parameter list

Return value

Number of lines.

TEXT_GetText()

Description

Copies the text of the given TEXT widget to the given buffer. The 0-Byte at the end of the string is written in any case.

Prototype

```
int TEXT_GetText(TEXT_Handle hObj, char * pDest, U32 BufferSize);
```

Parameter	Description
hObj	Handle of the TEXT widget.
pDest	Pointer to a user defined buffer.
BufferSize	Size of the buffer.

Table 19.870: TEXT_GetText() parameter list

Return value

Number of bytes copied.

TEXT_GetTextAlign()

Description

Returns the alignment of the given TEXT widget.

Prototype

```
int TEXT_GetTextAlign(TEXT_Handle hObj);
```

Parameter	Description
hObj	Handle of the TEXT widget.

Table 19.871: TEXT_GetTextAlign() parameter list

Return value

Alignment of the text.

TEXT_GetTextColor()

Description

Returns the text color of the given TEXT widget.

Prototype

```
GUI_COLOR TEXT_GetTextColor(TEXT_Handle hObj);
```

Parameter	Description
hObj	Handle of the TEXT widget.

Table 19.872: TEXT_GetTextColor() parameter list

Return value

Text color of the given TEXT widget.

TEXT_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

TEXT_GetWrapMode()

Description

Returns the currently used mode for wrapping text of the given TEXT widget.

Prototype

```
GUI_WRAPMODE TEXT_GetWrapMode(TEXT_Handle hObj);
```

Parameter	Description
hObj	Handle of the TEXT widget.

Table 19.873: TEXT_GetWrapMode() parameter list

Return value

Currently used wrap mode.

TEXT_SetBkColor()

Description

Sets the background color of the TEXT widget.

Prototype

```
void TEXT_SetBkColor(TEXT_Handle hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of TEXT widget.
Color	Color to be used for the background. (range 0x000000 and 0xFFFFFF or a valid color define) GUI_INVALID_COLOR to make background transparent

Table 19.874: TEXT_SetBkColor() parameter list

Additional information

The background of this widget can either be filled with any available color or transparent. If a valid RGB color is specified, the background is filled with the color, otherwise the background (typically the content of the parent window) is visible. If the background is transparent, the widget is treated as transparent window, otherwise as non-transparent window. Note that using a background color allows more efficient (faster) rendering.

TEXT_SetDefaultFont()

Description

Sets the default font used for TEXT widgets.

Prototype

```
void TEXT_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to the font to be set as default.

Table 19.875: TEXT_SetDefaultFont() parameter list

TEXT_SetDefaultTextColor()

Description

Sets the default text color used for TEXT widgets.

Prototype

```
void TEXT_SetDefaultTextColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color to be used.

Table 19.876: TEXT_SetDefaultTextColor() parameter list

TEXT_SetDefaultWrapMode()

Description

Sets the default text wrapping mode used for new TEXT widgets.

Prototype

```
GUI_WRAPMODE TEXT_SetDefaultWrapMode(GUI_WRAPMODE WrapMode);
```

Parameter	Description
WrapMode	Default text wrapping mode used for new TEXT widgets. See table below.

Table 19.877: TEXT_SetDefaultWrapMode() parameter list

Permitted values for parameter WrapMode	
GUI_WRAPMODE_NONE	No wrapping will be performed.
GUI_WRAPMODE_WORD	Text is wrapped word wise.
GUI_WRAPMODE_CHAR	Text is wrapped char wise.

Return value

Previous default text wrapping mode.

Additional information

The default wrapping mode for TEXT widgets is GUI_WRAPMODE_NONE. For details about text wrapping within the TEXT widget, refer to “TEXT_SetWrapMode()” on page 802.

TEXT_SetFont()

Description

Sets the font to be used for a specified TEXT widget.

Prototype

```
void TEXT_SetFont(TEXT_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of TEXT widget.
pFont	Pointer to the font to be used.

Table 19.878: TEXT_SetFont() parameter list

TEXT_SetText()

Description

Sets the text to be used for a specified TEXT widget.

Prototype

```
int TEXT_SetText(TEXT_Handle hObj, const char * s);
```

Parameter	Description
hObj	Handle of TEXT widget.
s	Text to be displayed.

Table 19.879: TEXT_SetText() parameter list

Return value

0 on success, 1 on error.

TEXT_SetTextAlign()

Description

Sets the text alignment of a specified TEXT widget.

Prototype

```
void TEXT_SetTextAlign(TEXT_Handle hObj, int Align);
```

Parameter	Description
hObj	Handle of TEXT widget.
Align	Text alignment (see TEXT_Create()).

Table 19.880: TEXT_SetTextAlign() parameter list

TEXT_SetTextColor()

Description

Sets the text color of a specified TEXT widget.

Prototype

```
void TEXT_SetTextColor(TEXT_Handle pObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of TEXT widget.
Color	New text color.

Table 19.881: TEXT_SetTextColor() parameter list

TEXT_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

TEXT_SetWrapMode()

Description

Sets the wrapping mode of a specified TEXT widget.

Prototype

```
void TEXT_SetWrapMode(TEXT_Handle hObj, GUI_WRAPMODE WrapMode);
```

Parameter	Description
hObj	Handle of TEXT widget.
WrapMode	See table below.

Table 19.882: TEXT_SetWrapMode() parameter list

Permitted values for parameter WrapMode	
GUI_WRAPMODE_NONE	No wrapping will be performed.
GUI_WRAPMODE_WORD	Text is wrapped word wise.
GUI_WRAPMODE_CHAR	Text is wrapped char wise.

Additional information

The default wrapping mode for TEXT widgets is GUI_WRAPMODE_NONE. For more details about text wrapping, refer to “GUI_DispStringInRectWrap()” on page 101.

19.26.6 Examples

There is no special sample for this widget, since many of the emWin samples use it:

- DIALOG_Count.c
- DIALOG_Radio.c
- WIDGET_GraphXY.c
- ...

19.27 TREEVIEW: Treeview widget

A TREEVIEW widget can be used to show a hierarchical view of information like files in a directory or items of an index, whereas each item can be a node or a leaf. Each node can have a number of sub items and can be closed or opened.
A node consists of a button image, which shows a plus sign in closed state or a minus sign in open state, two item images (one for closed and one for open state) and the item text. Pressing the button image or double clicking the item image toggles the state (open or closed) of the node.
A leaf consists of an item image and the item text.
The current selection can be marked by highlighting the item text or by highlighting the whole row. All items of a tree are joined by lines per default.
All TREEVIEW-related routines are located in the file(s) `TREEVIEW*.c`, `TREEVIEW*.h`. All identifiers are prefixed TREEVIEW. The table below shows the appearances of the TREEVIEW widget:

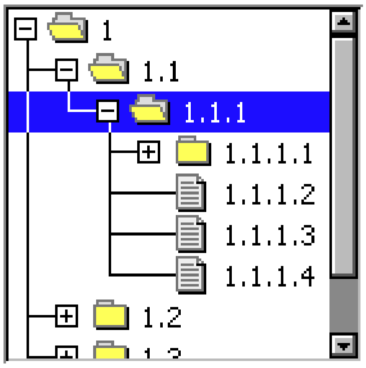
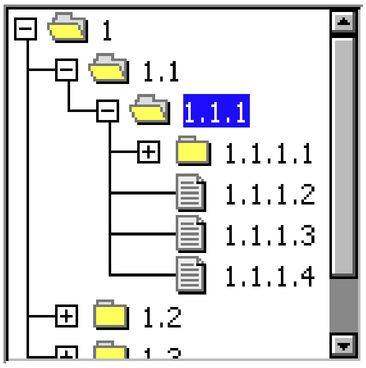
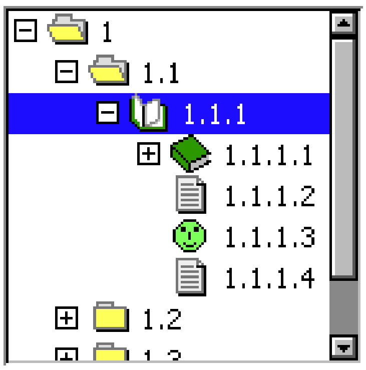
Description	TREEVIEW widget
TREEVIEW widget with row selection enabled.	
TREEVIEW widget with text selection enabled.	
TREEVIEW widget with some application defined bitmaps and lines off.	

Table 19.883: TREEVIEW appearance

19.27.1 Description of terms

Item

This means a TREEVIEW item which can be a leaf or a node.

Leaf

A leaf is a TREEVIEW item which is not able to have any children. It is represented by the leaf bitmap and the item text.

Node

A node is a TREEVIEW item which is able to have children. It is represented by the button bitmap, the node bitmap and the item text. The state of the node can be toggled by pressing the button bitmap or by double clicking the node bitmap or the selected area of the item. In open state the children are visible below the node at the next level of indentation.

Button bitmap

This means the bitmap visible at nodes which can be pressed to toggle the state of the node.

Item bitmap

Left beside the item text the item bitmap is shown. Which bitmap is shown depends in the item (leaf or node) and in case of a node it also depends on the state, collapsed or expanded.

Expanded state

In expanded state the children of a node are visible and the minus sign is shown in the button bitmap.

Collapsed state

In collapsed state the children of a node are hidden and the plus sign is shown in the button bitmap.

Joining lines

Lines which are used to connect the items of a tree. The lines connect the button bitmaps of the nodes and the item bitmaps of the leafs according to the hierarchy of the tree.

19.27.2 Configuration options

Type	Macro	Default	Description
	TREEVIEW_FONT_DEFAULT	&GUI_Font13_1	Default font used to draw the text.
	TREEVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	Background color for unselected state
	TREEVIEW_BKCOLOR1_DEFAULT	GUI_BLUE	Background color for selected state.
	TREEVIEW_BKCOLOR2_DEFAULT	0xC0C0C0	Background color for disabled state.
	TREEVIEW_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color for unselected state.
	TREEVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color for selected state.
	TREEVIEW_TEXTCOLOR2_DEFAULT	GUI_GRAY	Text color for disabled state.
	TREEVIEW_LINECOLOR0_DEFAULT	GUI_BLACK	Line color for unselected state.
	TREEVIEW_LINECOLOR1_DEFAULT	GUI_WHITE	Line color for selected state.
	TREEVIEW_LINECOLOR2_DEFAULT	GUI_GRAY	Line color for disabled state.
	TREEVIEW_IMAGE_CLOSED_DEFAULT		Item image for node in closed state.
	TREEVIEW_IMAGE_OPEN_DEFAULT		Item image for node in open state.
	TREEVIEW_IMAGE_LEAF_DEFAULT		Item image for leaf.
	TREEVIEW_IMAGE_PLUS_DEFAULT		Plus sign.
	TREEVIEW_IMAGE_MINUS_DEFAULT		Minus sign.
	TREEVIEW_INDENT_DEFAULT	16	Number of pixels for indenting.
	TREEVIEW_TEXT_INDENT_DEFAULT	20	Number of pixels for indenting text.

Table 19.884: Configuration options

19.27.3 Predefined IDs

The following symbols define IDs which may be used to make TREEVIEW widgets distinguishable from creation: `GUI_ID_TREEVIEW0` - `GUI_ID_TREEVIEW3`

19.27.4 Notification codes

The following events are sent from a treeview widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_CLICKED	TREEVIEW has been clicked.
WM_NOTIFICATION_RELEASED	TREEVIEW has been released.
WM_NOTIFICATION_MOVED_OUT	TREEVIEW has been clicked and pointer has been moved out of the widget area without releasing.
WM_NOTIFICATION_SEL_CHANGED	Value (selection) of the TREEVIEW widget has changed.

Table 19.885: Notification codes

19.27.5 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	If the cursor is at a closed node, the node is opened. If the cursor is at an open node the cursor moves to the first child of the node.
GUI_KEY_DOWN	The cursor moves to the next visible item below the current position.
GUI_KEY_LEFT	If the cursor is at a leaf the cursor moves to the parent node of the item. If the cursor is at an open node, the node will be closed. If the cursor is at a closed node, the cursor moves to the next parent node.
GUI_KEY_UP	The cursor moves to the previous visible item above the current position.

Table 19.886: Keyboard reaction

19.27.6 TREEVIEW API

The table below lists the available TREEVIEW-related routines of emWin in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
Common routines	
TREEVIEW_AttachItem()	Attaches an already existing item to the given TREEVIEW.
TREEVIEW_CreateEx()	Creates a TREEVIEW widget.
TREEVIEW_CreateIndirect()	Creates a TREEVIEW widget from a resource table.
TREEVIEW_CreateUser()	Creates a TREEVIEW widget using extra bytes as user data.
TREEVIEW_DecSel()	Moves the cursor to the previous visible item.
TREEVIEW_GetDefaultBkColor()	Returns the default background color.
TREEVIEW_GetDefaultFont()	Returns the default font used to draw the item text.
TREEVIEW_GetDefaultLineColor()	Returns the default line color.
TREEVIEW_GetDefaultTextColor()	Returns the default text color.
TREEVIEW_GetItem()	Returns the requested item.
TREEVIEW_GetSel()	Returns the currently selected item.
TREEVIEW_GetUserData()	Retrieves the data set with TREEVIEW_SetUserData().
TREEVIEW_IncSel()	Moves the cursor to the next visible item.
TREEVIEW_InsertItem()	Inserts the given item at the given position.
TREEVIEW_ScrollToSel()	Scrolls the given TREEVIEW widget to show the current selection.
TREEVIEW_SetAutoScrollH()	Manages the automatic use of a horizontal scroll bar.
TREEVIEW_SetAutoScrollV()	Manages the automatic use of a vertical scroll bar.
TREEVIEW_SetBitmapOffset()	Sets the offset of the plus/minus bitmap.
TREEVIEW_SetBkColor()	Sets the background color.
TREEVIEW_SetDefaultBkColor()	Sets the default background color for TREEVIEW widgets.
TREEVIEW_SetDefaultFont()	Sets the default font for TREEVIEW widgets.
TREEVIEW_SetDefaultLineColor()	Sets the default line color for TREEVIEW widgets.
TREEVIEW_SetDefaultTextColor()	Sets the default text color for TREEVIEW widgets.
TREEVIEW_SetFont()	Sets the font used to draw the item text.
TREEVIEW_SetHasLines()	Manages the visibility of the joining lines.
TREEVIEW_SetImage()	Sets the images used to draw the treeview items.
TREEVIEW_SetIndent()	Sets the indentation distance for treeview items.
TREEVIEW_SetLineColor()	Sets the color used to draw the joining lines.
TREEVIEW_SetOwnerDraw()	Enables the TREEVIEW to be owner drawn.
TREEVIEW_SetSel()	Sets the selection of the TREEVIEW widget.
TREEVIEW_SetSelMode()	Manages the highlighting of the current selection.
TREEVIEW_SetTextColor()	Sets the color used to draw the treeview items.

Table 19.887: TREEVIEW API list

Routine	Description
TREEVIEW_SetTextIndent()	Sets the indentation distance for item text.
TREEVIEW_SetUserData()	Sets the extra data of a TREEVIEW widget.
Item related routines	
TREEVIEW_ITEM_Collapse()	Collapses the given node.
TREEVIEW_ITEM_CollapseAll()	Collapses the given node and all subnodes.
TREEVIEW_ITEM_Create()	Creates a new TREEVIEW item.
TREEVIEW_ITEM_Delete()	Deletes the given TREEVIEW item.
TREEVIEW_ITEM_Detach()	Detaches the given item without deleting it.
TREEVIEW_ITEM_Expand()	Expands the given node.
TREEVIEW_ITEM_ExpandAll()	Expands the given node and all subnodes.
TREEVIEW_ITEM_GetInfo()	Returns an information structure of the given item.
TREEVIEW_ITEM_GetText()	Returns the item text.
TREEVIEW_ITEM_GetUserData()	Returns the UserData value of the treeview item.
TREEVIEW_ITEM_SetImage()	Sets the images used to draw the individual given item.
TREEVIEW_ITEM_SetText()	Sets the text of the given item.
TREEVIEW_ITEM_SetUserData()	Sets the UserData value of the TREEVIEW item.

Table 19.887: TREEVIEW API list

19.27.6.1Common routines

TREEVIEW_AttachItem()

Description

Attaches an already existing item to the TREEVIEW widget.

Prototype

```
int TREEVIEW_AttachItem(TREEVIEW_Handle      hObj,
                        TREEVIEW_ITEM_Handle hItem,
                        TREEVIEW_ITEM_Handle hItemAt, int Position);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
hItem	Handle of item to be attached.
hItemAt	Handle of a currently attached item which specifies the position to be used.
Position	See table below.

Table 19.888: TREEVIEW_AttachItem() parameter list

Permitted values for parameter Position	
TREEVIEW_INSERT_ABOVE	Attaches the item above the given position at the same indent level as the given position.
TREEVIEW_INSERT_BELOW	Attaches the item below the given position at the same indent level as the given position.
TREEVIEW_INSERT_FIRST_CHILD	Attaches the item below the given position by indenting it. The given position needs to be a node level.

Return value

0 on success, otherwise 1.

Additional information

The function can be used for attaching a single item as well as for attaching a complete tree. Note that in case of attaching a tree, the root item of the tree needs to be passed as hItem. If attaching the first item to an empty treeview the parameters hItem and Position should be 0.

TREEVIEW_CreateEx()

Description

Creates a TREEVIEW widget of a specified size at a specified location.

Prototype

```
TREEVIEW_Handle TREEVIEW_CreateEx(int     x0,      int y0,
                                  int     xSize,   int ySize,
                                  WM_HWIN hParent, int WinFlags,
                                  int     ExFlags, int Id);
```

Parameter	Description
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Horizontal size of the widget (in pixels).
ySize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new TEXT widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter “The Window Manager (WM)” on page 375 for a list of available parameter values).
ExFlags	See table below.
Id	Window ID of the widget.

Table 19.889: TREEVIEW_CreateEx() parameter list

Permitted values for parameter ExFlags	
TREEVIEW_CF_HIDELINES	Joining lines are not displayed.
TREEVIEW_CF_ROWSELECT	Activates row selection mode.
TREEVIEW_CF_AUTOSCROLLBAR_H	Enables the use of an automatic horizontal scroll bar.
TREEVIEW_CF_AUTOSCROLLBAR_V	Enables the use of an automatic vertical scroll bar.

Return value

Handle of the created widget. 0, if the function fails.

Additional information

The values of parameter ExFlags can be or-combined.

TREEVIEW_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The element Para of the according GUI_WIDGET_CREATE_INFO structure is not used. The element Flags is used according to the parameter ExFlags of the function TREEVIEW_CreateEx().

TREEVIEW_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function TREEVIEW_CreateEx() can be referred to.

TREEVIEW_DecSel()

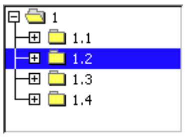
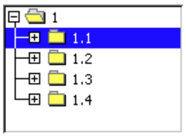
Before	After
	

Table 19.890: TREEVIEW_DecSel() before after screenshots

Description

Moves the cursor to the previous visible item of the given TREEVIEW widget.

Prototype

```
void TREEVIEW_DecSel(TREEVIEW_Handle hObj);
```

Parameter	Description
`hObj`	Handle of TREEVIEW widget.

Table 19.891: TREEVIEW_DecSel() parameter list

Additional information

If there is no previous visible item the cursor remains on the current position.

TREEVIEW_GetDefaultBkColor()

Description

Returns the default background color used for new TREEVIEW widgets.

Prototype

```
GUI_COLOR TREEVIEW_GetDefaultBkColor(int Index);
```

Parameter	Description
`Index`	See table below.

Table 19.892: TREEVIEW_GetDefaultBkColor() parameter list

Permitted values for parameter `Index`	
`TREEVIEW_CI_UNSEL`	Background color of unselected element.
`TREEVIEW_CI_SEL`	Background color of selected element.
`TREEVIEW_CI_DISABLED`	Background color of disabled element.

Return value

Default background color used for new TREEVIEW widgets.

TREEVIEW_GetDefaultFont()

Description

Returns the default font used to draw the item text of new TREEVIEW widgets.

Prototype

```
const GUI_FONT * TREEVIEW_GetDefaultFont(void);
```

Return value

Default font used to draw the item text of new TREEVIEW widgets.

TREEVIEW_GetDefaultLineColor()

Description

Returns the default color used to draw the joining lines of new TREEVIEW widgets.

Prototype

```
GUI_COLOR TREEVIEW_GetDefaultLineColor(int Index);
```

Parameter	Description
Index	See table below.

Table 19.893: TREEVIEW_GetDefaultLineColor() parameter list

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Line color of unselected element.
TREEVIEW_CI_SEL	Line color of selected element.
TREEVIEW_CI_DISABLED	Line color of disabled element.

Return value

Default color used to draw the joining lines of new treeview widgets.

TREEVIEW_GetDefaultTextColor()

Description

Returns the default text color used to draw the item text of new TREEVIEW widgets.

Prototype

```
GUI_COLOR TREEVIEW_GetDefaultTextColor(int Index);
```

Parameter	Description
Index	See table below.

Table 19.894: TREEVIEW_GetDefaultTextColor() parameter list

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Text color of unselected element.
TREEVIEW_CI_SEL	Text color of selected element.
TREEVIEW_CI_DISABLED	Text color of disabled element.

Return value

Default text color used to draw the item text of new TREEVIEW widgets.

TREEVIEW_GetItem()

Description

Returns the handle of the requested treeview item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_GetItem(TREEVIEW_Handle      hObj,
                                      TREEVIEW_ITEM_Handle hItem,
                                      int                  Flags);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
hItem	Handle of TREEVIEW item specifying the position to start search from.
Flags	See table below.

Table 19.895: TREEVIEW_GetItem() parameter list

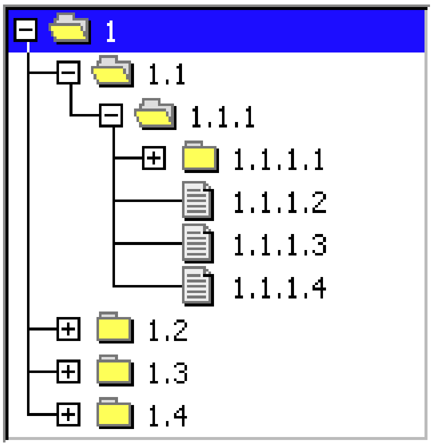
Permitted values for parameter `Flags`	
`TREEVIEW_GET_FIRST`	Returns the first item of the TREEVIEW widget. Parameter `hItem` is not required and can be 0.
`TREEVIEW_GET_LAST`	Returns the last item of the TREEVIEW widget. Parameter `hItem` is not required and can be 0.
`TREEVIEW_GET_NEXT_SIBLING`	Returns the next child item of the parent node of `hItem`.
`TREEVIEW_GET_PREV_SIBLING`	Returns the previous child item of the parent node of `hItem`.
`TREEVIEW_GET_FIRST_CHILD`	Returns the first child of the given node.
`TREEVIEW_GET_PARENT`	Returns the parent node of the given item.

Return value

Handle of the requested TREEVIEW item on success, otherwise 0.

Example

The picture shows a TREEVIEW widget with several items. The following shows how parameter `Flags` can be used for getting TREEVIEW items relative to parameter `hItem`:



- `TREEVIEW_GET_NEXT_SIBLING`
  The next sibling of '1.1' is '1.2'.
- `TREEVIEW_GET_PREV_SIBLING`
  The previous sibling of '1.2' is '1.1'.
- `TREEVIEW_GET_FIRST_CHILD`
  The first child item of '1.1.1' is '1.1.1.1'.
- `TREEVIEW_GET_PARENT`
  The parent item of '1.1' is '1'.

The use of `TREEVIEW_GET_FIRST` and `TREEVIEW_GET_LAST` should be obvious. If the requested item does not exist, the function returns 0.

TREEVIEW_GetSel()

Description

Returns the handle of the currently selected TREEVIEW item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_GetSel(TREEVIEW_Handle hObj);
```

Parameter	Description
`hObj`	Handle of TREEVIEW widget.

Table 19.896: TREEVIEW_GetSel() parameter list

Return value

Handle of the currently selected treeview item. If no item has been selected the return value is 0.

TREEVIEW_GetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_GetUserData()`.

TREEVIEW_IncSel()

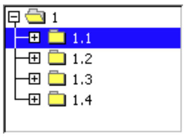
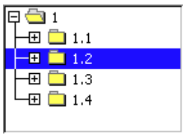
Before	After
	

Table 19.897: TREEVIEW_IncSel() before after screenshots

Description

Moves the cursor to the next visible item of the given TREEVIEW widget.

Prototype

```
void TREEVIEW_IncSel(TREEVIEW_Handle hObj);
```

Parameter	Description
`hObj`	Handle of TREEVIEW widget.

Table 19.898: TREEVIEW_IncSel() parameter list

Additional information

If there is no next visible item the cursor remains on the current position.

TREEVIEW_InsertItem()

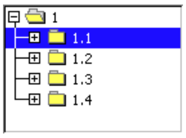
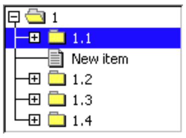
Before	After
	

Table 19.899: TREEVIEW_InsertItem() before after screenshots

Description

The function creates and inserts one new TREEVIEW item relative to the given item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_InsertItem(TREEVIEW_Handle hObj,
                                         int IsNode,
                                         TREEVIEW_ITEM_Handle hItemPrev,
                                         int Position,
                                         const char * s);
```

Parameter	Description
`hObj`	Handle of TREEVIEW widget.
`IsNode`	See table below.
`hItemPrev`	Handle of TREEVIEW item specifying the position of the new item.
`Position`	See table below.
`s`	Text of new TREEVIEW item.

Table 19.900: TREEVIEW_InsertItem() parameter list

Permitted values for parameter `IsNode`	
`TREEVIEW_ITEM_TYPE_LEAF`	New item is a 'leaf'.
`TREEVIEW_ITEM_TYPE_NODE`	New item is a 'node'.

Permitted values for parameter Position	
TREEVIEW_INSERT_FIRST_CHILD	Should be used for the first item of a treeview node.
TREEVIEW_INSERT_ABOVE	Inserts the item above the given item with the same indent level.
TREEVIEW_INSERT_BELOW	Inserts the item below the given item with the same indent level.

Return value

Handle of the new item on success, otherwise 0.

TREEVIEW_ScrollToSel()

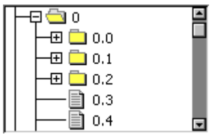
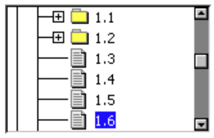
Before	After
	

Table 19.901: TREEVIEW_ScrollToSel() before after screenshots

Description

Scrolls the given TREEVIEW widget to show the current selection.

Prototype

```
void TREEVIEW_ScrollToSel(TREEVIEW_Handle hObj);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.

Table 19.902: TREEVIEW_ScrollToSel() parameter list

TREEVIEW_SetAutoScrollH()

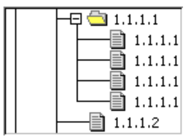
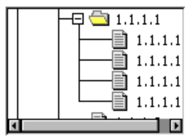
Before	After
	

Table 19.903: TREEVIEW_SetAutoScrollH() before after screenshots

Description

Enables or disables the use of an automatic horizontal scroll bar.

Prototype

```
void TREEVIEW_SetAutoScrollH(TREEVIEW_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
State	1 for enabling an automatic horizontal scroll bar, 0 for disabling.

Table 19.904: TREEVIEW_SetAutoScrollH() parameter list

TREEVIEW_SetAutoScrollV()

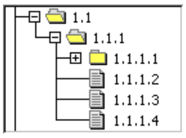
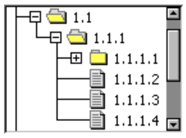
Before	After
	

Table 19.905: TREEVIEW_SetAutoScrollV() before after screenshots

Description

Enables or disables the use of an automatic vertical scroll bar.

Prototype

```
void TREEVIEW_SetAutoScrollV(TREEVIEW_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
State	1 for enabling an automatic vertical scroll bar, 0 for disabling.

Table 19.906: TREEVIEW_SetAutoScrollV() parameter list

TREEVIEW_SetBitmapOffset()

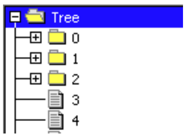
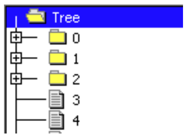
Before	After
	

Table 19.907: TREEVIEW_SetBitmapOffset() before after screenshots

Description

Sets the offset of the plus/minus bitmap.

Prototype

```
void TREEVIEW_SetBitmapOffset(TREEVIEW_Handle hObj, int Index,
                              int             xOff, int yOff);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Index	Currently the only permitted value for this parameter is TREEVIEW_BI_PM.
xOff	Horizontal offset.
yOff	Vertical offset.

Table 19.908: TREEVIEW_SetBitmapOffset() parameter list

Additional information

If `xOff` and `yOff` are set to 0 (default), the plus/minus bitmap is centered horizontally and vertically in the indentation space left of the actual item. The indentation space is related to the parent item (if exists) or to the left border of the widget. See "before / after" screenshots of the function “TREEVIEW_SetIndent()” on page 819.

TREEVIEW_SetBkColor()

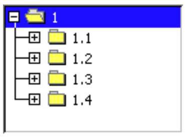
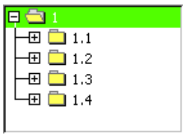
Before	After
	

Table 19.909: TREEVIEW_SetBkColor() before after screenshots

Description

Sets the background color of the given TREEVIEW widget.

Prototype

```
void TREEVIEW_SetBkColor(TREEVIEW_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Index	See table below.
Color	Color to be used.

Table 19.910: TREEVIEW_SetBkColor() parameter list

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Color of unselected item.
TREEVIEW_CI_SEL	Color of selected item.
TREEVIEW_CI_DISABLED	Color of disabled item.

TREEVIEW_SetDefaultBkColor()

Description

Sets the default background color used for new TREEVIEW widgets.

Prototype

```
void TREEVIEW_SetDefaultBkColor(int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Refer to "TREEVIEW_SetBkColor()" on page 816.
Color	Color to be used.

Table 19.911: TREEVIEW_SetDefaultBkColor() parameter list

TREEVIEW_SetDefaultFont()

Description

Sets the default font used for new TREEVIEW widgets.

Prototype

```
void TREEVIEW_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Description
pFont	Pointer to GUI_FONT structure to be used.

Table 19.912: TREEVIEW_SetDefaultFont() parameter list

TREEVIEW_SetDefaultLineColor()

Description

Sets the default line color used for new TREEVIEW widgets.

Prototype

```
void TREEVIEW_SetDefaultLineColor(int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Refer to "TREEVIEW_SetBkColor()" on page 816.
Color	Color to be used.

Table 19.913: TREEVIEW_SetDefaultLineColor() parameter list

TREEVIEW_SetDefaultTextColor()

Description

Sets the default text color used for new TREEVIEW widgets.

Prototype

```
void TREEVIEW_SetDefaultTextColor(int Index, GUI_COLOR Color);
```

Parameter	Description
Index	Refer to "TREEVIEW_SetBkColor()" on page 816.
Color	Color to be used.

Table 19.914: TREEVIEW_SetDefaultTextColor() parameter list

TREEVIEW_SetFont()

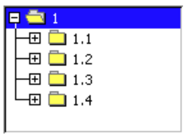
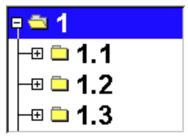
Before	After
	

Table 19.915: TREEVIEW_SetFont() before after screenshots

Description

Sets the font to be used to draw the item text of the given TREEVIEW widget.

Prototype

```
void TREEVIEW_SetFont(TREEVIEW_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
pFont	Pointer to GUI_FONT structure to be used.

Table 19.916: TREEVIEW_SetFont() parameter list

TREEVIEW_SetHasLines()

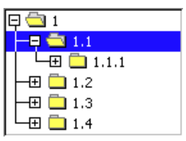
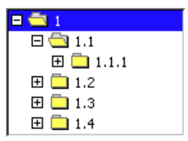
Before	After
	

Table 19.917: TREEVIEW_SetHasLines() before after screenshots

Description

Manages the visibility of the joining lines between the TREEVIEW items.

Prototype

```
void TREEVIEW_SetHasLines(TREEVIEW_Handle hObj, int State);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
State	1 for showing the lines, 0 for not showing the lines.

Table 19.918: TREEVIEW_SetHasLines() parameter list

Additional information

Per default the lines are shown.

TREEVIEW_SetImage()

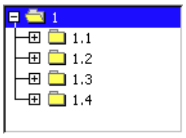
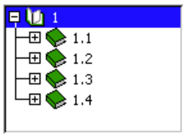
Before	After
	

Table 19.919: TREEVIEW_SetImage() before after screenshots

Description

Sets the images used to draw the TREEVIEW items.

Prototype

```
void TREEVIEW_SetImage(TREEVIEW_Handle    hObj,     int Index,
                       const GUI_BITMAP * pBitmap);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Index	See table below.
pBitmap	Pointer to bitmap structure to be used.

Table 19.920: TREEVIEW_SetImage() parameter list

Permitted values for parameter Index	
TREEVIEW_BI_CLOSED	Image of closed nodes.
TREEVIEW_BI_OPEN	Image of open nodes.
TREEVIEW_BI_LEAF	Image of leaf.
TREEVIEW_BI_PLUS	Plus sign of closed nodes.
TREEVIEW_BI_MINUS	Minus sign of open nodes.

Additional information

The function `TREEVIEW_SetItemImage()` can be used to set individual images for each item.

TREEVIEW_SetIndent()

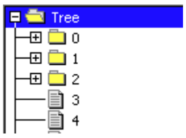
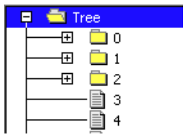
Before	After
	

Table 19.921: TREEVIEW_SetIndent() before after screenshots

Description

Sets the indentation of TREEVIEW items in pixels. Indentation is 16 pixels by default.

Prototype

```
int TREEVIEW_SetIndent(TREEVIEW_Handle hObj, int Indent);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Indent	Distance (in pixels) to indent treeview items.

Table 19.922: TREEVIEW_SetIndent() parameter list

Return value

Previous indentation.

TREEVIEW_SetLineColor()

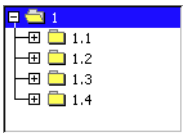
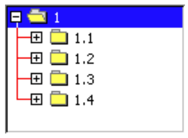
Before	After
	

Table 19.923: TREEVIEW_SetLineColor() before after screenshots

Description

Sets the color used to draw the joining lines between the TREEVIEW items.

Prototype

```
void TREEVIEW_SetLineColor(TREEVIEW_Handle hObj, int Index,
                                                 GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Index	Refer to “TREEVIEW_SetBkColor()” on page 816.
Color	Color to be used.

Table 19.924: TREEVIEW_SetLineColor() parameter list

TREEVIEW_SetOwnerDraw()

Description

Enables the TREEVIEW to be owner drawn.

Prototype

```
void TREEVIEW_SetOwnerDraw(TREEVIEW_Handle         hObj,
```



```
WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
pfDrawItem	Pointer to the owner draw function. See "User drawn widgets" on page 460.

Table 19.925: TREEVIEW_SetOwnerDraw() parameter list

TREEVIEW_SetSel()

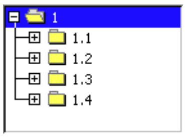
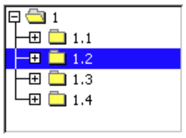
Before	After
	

Table 19.926: TREEVIEW_SetSel() before after screenshots

Description

Sets the currently selected item of the TREEVIEW widget.

Prototype

```
void TREEVIEW_SetSel(TREEVIEW_Handle hObj, TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
hItem	Handle of treeview item to be selected.

Table 19.927: TREEVIEW_SetSel() parameter list

Additional information

If the given TREEVIEW item is a child of a closed node no selection is visible after calling this function.

TREEVIEW_SetSelMode()

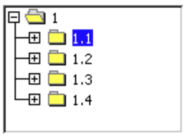
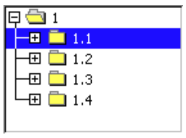
Before	After
	

Table 19.928: TREEVIEW_SetSelMode() before after screenshots

Description

Sets the selection mode of the TREEVIEW widget.

Prototype

```
void TREEVIEW_SetSelMode(TREEVIEW_Handle hObj, int Mode);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Mode	See table below.

Table 19.929: TREEVIEW_SetSelMode() parameter list

Permitted values for parameter Mode	
TREEVIEW_SELMODE_ROW	Activates row selection mode.
TREEVIEW_SELMODE_TEXT	Activates text selection mode.

Additional information

Default selection mode is text selection. If row selection is activated, the complete row can be used to select the item. If text selection is active, only the item text and the item bitmap can be used for selection.

TREEVIEW_SetTextColor()

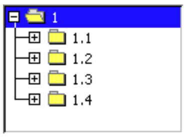
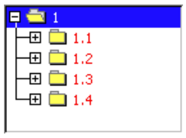
Before	After
	

Table 19.930: TREEVIEW_SetTextColor() before after screenshots

Description

Sets the color used to draw the TREEVIEW items of the given TREEVIEW widget.

Prototype

```
void TREEVIEW_SetTextColor(TREEVIEW_Handle hObj, int Index,
                           GUI_COLOR       Color);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
Index	Refer to "TREEVIEW_SetBkColor()" on page 816.
Color	Color to be used.

Table 19.931: TREEVIEW_SetTextColor() parameter list

TREEVIEW_SetTextIndent()

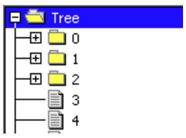
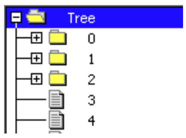
Before	After
	

Table 19.932: TREEVIEW_SetTextIndent() before after screenshots

Description

Sets the indentation of item text in pixels. Text indentation is 20 pixels by default.

Prototype

```
int TREEVIEW_SetTextIndent(TREEVIEW_Handle hObj, int TextIndent);
```

Parameter	Description
hObj	Handle of TREEVIEW widget.
TextIndent	Text indentation to be used.

Table 19.933: TREEVIEW_SetTextIndent() parameter list

Return value

Previous text indentation.

TREEVIEW_SetUserData()

Prototype explained at the beginning of the chapter as `<WIDGET>_SetUserData()`.

19.27.6.2Item related routines

TREEVIEW_ITEM_Collapse()

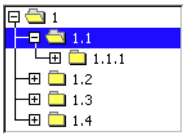
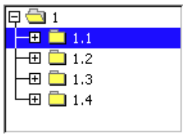
Before	After
	

Table 19.934: TREEVIEW_ITEM_Collapse() before after screenshots

Description

Collapses the given node and shows the plus sign afterwards.

Prototype

```
void TREEVIEW_ITEM_Collapse(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of the item to be collapsed.

Table 19.935: TREEVIEW_ITEM_Collapse() parameter list

Additional information

The given item needs to be a node. Otherwise the function returns immediately.

TREEVIEW_ITEM_CollapseAll()

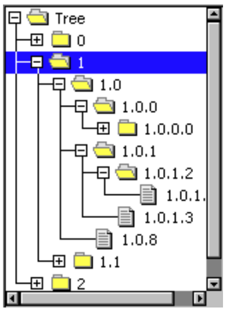
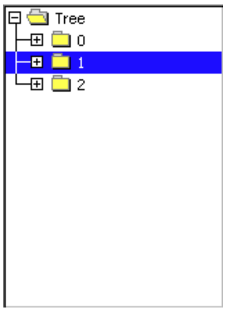
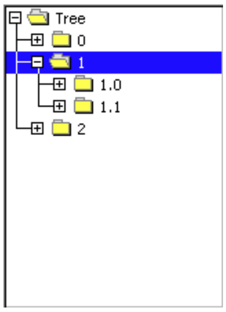
Before	All nodes collapsed	Expanded again
		

Table 19.936: TREEVIEW_ITEM_CollapseAll() screenshots

Description

Collapses the given node and all subnodes and shows the plus sign afterwards.

Prototype

```
void TREEVIEW_ITEM_CollapseAll(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of the item to be collapsed.

Table 19.937: TREEVIEW_ITEM_CollapseAll() parameter list

Additional information

This function collapses all subnodes, so if the given node is expanded again, all subnodes are in collapsed state.

TREEVIEW_ITEM_Create()

Description

Creates a new TREEVIEW item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_CreateItem(int IsNode, const char * s,
                                         U32 UserData);
```

Parameter	Description
IsNode	See table below.
s	Pointer to item text to be shown.
UserData	32 bit value to be used by the application.

Table 19.938: TREEVIEW_ITEM_Create() parameter list

Permitted values for parameter IsNode	
TREEVIEW_ITEM_TYPE_NODE	Used to create a node.
TREEVIEW_ITEM_TYPE_LEAF	Used to create a leaf.

Return value

Handle of new item on success, otherwise 0.

Additional information

After creating a treeview item it contains a copy of the text.

TREEVIEW_ITEM_Delete()

Description

Deletes the given TREEVIEW item.

Prototype

```
void TREEVIEW_ITEM_Delete(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of item to be deleted.

Table 19.939: TREEVIEW_ITEM_Delete() parameter list

Additional information

If the item is currently not attached to any TREEVIEW, the parameter `hObj` should be 0. The function can be used to delete a single item as well as for deleting a complete tree. In case of deleting a tree the root element of the tree should be passed to the function.

TREEVIEW_ITEM_Detach()

Description

Detaches the given TREEVIEW item from the TREEVIEW widget.

Prototype

```
void TREEVIEW_ITEM_Detach(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of item to be detached.

Table 19.940: TREEVIEW_ITEM_Detach() parameter list

Additional information

The function detaches the given item and all of its children from the TREEVIEW widget.

TREEVIEW_ITEM_Expand()

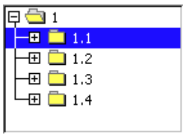
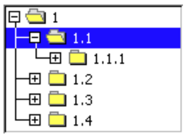
Before	After
	

Table 19.941: TREEVIEW_ITEM_Expand() before after screenshots

Description

Expands the given node and shows the minus sign afterwards.

Prototype

```
void TREEVIEW_ITEM_Expand(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of node to be expanded.

Table 19.942: TREEVIEW_ITEM_Expand() parameter list

Additional information

The given item needs to be a node. Otherwise the function returns immediately.

TREEVIEW_ITEM_ExpandAll()

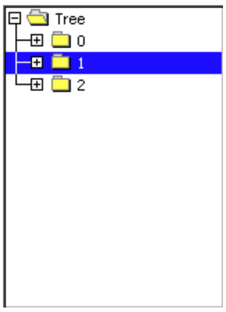
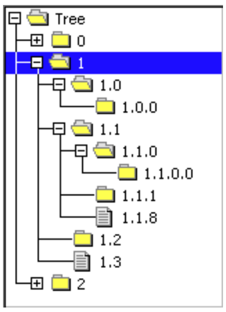
Before	After
	

Table 19.943: TREEVIEW_ITEM_ExpandAll() before after screenshots

Description

Expands the given node and all subnodes and shows the minus sign afterwards.

Prototype

```
void TREEVIEW_ITEM_ExpandAll(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of the item to be expanded.

Table 19.944: TREEVIEW_ITEM_ExpandAll() parameter list

TREEVIEW_ITEM_GetInfo()

Description

Returns a structure with information about the given item.

Prototype

```
void TREEVIEW_ITEM_GetInfo(TREEVIEW_ITEM_Handle   hItem,
                           TREEVIEW_ITEM_INFO   * pInfo);
```

Parameter	Description
hItem	Handle of TREEVIEW item.
pInfo	Pointer to a TREEVIEW_ITEM_INFO structure to be filled by the function.

Table 19.945: TREEVIEW_ITEM_GetInfo() parameter list

Elements of structure TREEVIEW_ITEM_INFO

Data type	Element	Description
int	IsNode	1 if item is a node, 0 if not.
int	IsExpanded	1 if item (node) is open, 0 if closed.
int	HasLines	1 if joining lines are visible, 0 if not.
int	HasRowSelect	1 if row selection is active, 0 if not.
int	Level	Indentation level of item.

Table 19.946: TREEVIEW_ITEM_INFO element list

TREEVIEW_ITEM_GetText()

Description

Returns the item text of the given TREEVIEW item.

Prototype

```
void TREEVIEW_ITEM_GetText(TREEVIEW_ITEM_Handle   hItem,
                           U8                   * pBuffer,
                           int                    MaxNumBytes);
```

Parameter	Description
hItem	Handle of TREEVIEW item.
pBuffer	Pointer to buffer filled by the function.
MaxNumBytes	Size of the buffer in bytes.

Table 19.947: TREEVIEW_ITEM_GetText() parameter list

Additional information

If `MaxNumBytes` is less than the item text length the buffer is filled with the first `MaxNumBytes` of the item text.

TREEVIEW_ITEM_GetUserData()

Description

The function return the 32 bit value associated with the given treeview item which can be used by the application program.

Prototype

```
U32 TREEVIEW_ITEM_GetUserData(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Description
hItem	Handle of TREEVIEW item.

Table 19.948: TREEVIEW_ITEM_GetUserData() parameter list

TREEVIEW_ITEM_SetImage()

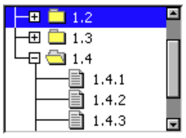
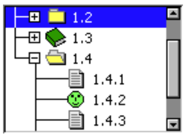
Before	After
	

Table 19.949: TREEVIEW_ITEM_SetImage() before after screenshots

Description

The function sets images to be used only with the given TREEVIEW item.

Prototype

```
void TREEVIEW_ITEM_SetImage(TREEVIEW_ITEM_Handle   hItem,    int Index,
                            const GUI_BITMAP     * pBitmap);
```

Parameter	Description
hItem	Handle of TREEVIEW item.
Index	See table below.
pBitmap	Pointer to bitmap structure to be used.

Table 19.950: TREEVIEW_ITEM_SetImage() parameter list

Permitted values for parameter Index	
TREEVIEW_BI_CLOSED	Image of closed node.
TREEVIEW_BI_OPEN	Image of open node.
TREEVIEW_BI_LEAF	Image of leaf.

Additional information

This function 'overwrites' the default images of the widget. If no individual image is set the default image is used.

TREEVIEW_ITEM_SetText()

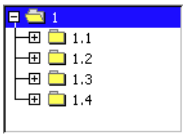
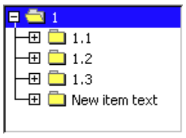
Before	After
	

Table 19.951: TREEVIEW_ITEM_SetText() before after screenshots

Description

The function sets the text of the given item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_ITEM_SetText(TREEVIEW_ITEM_Handle hItem,
```



```
const char * s);
```

Parameter	Description
hItem	Handle of TREEVIEW item.
s	Pointer to text to be used.

Table 19.952: TREEVIEW_ITEM_SetText() parameter list

Return value

Handle of the TREEVIEW item with the new text.

Additional information

The text will be copied into the treeview item. Note that using this function changes the handle of the item. After calling this function, the new handle needs to be used.

TREEVIEW_ITEM_SetUserData()

Description

The function sets a 32 bit value associated with the given treeview item which can be used by the application program.

Prototype

```
void TREEVIEW_ITEM_SetUserData(TREEVIEW_ITEM_Handle hItem, U32 UserData);
```

Parameter	Description
hItem	Handle of TREEVIEW item.
UserData	32 bit value to be used by the application program.

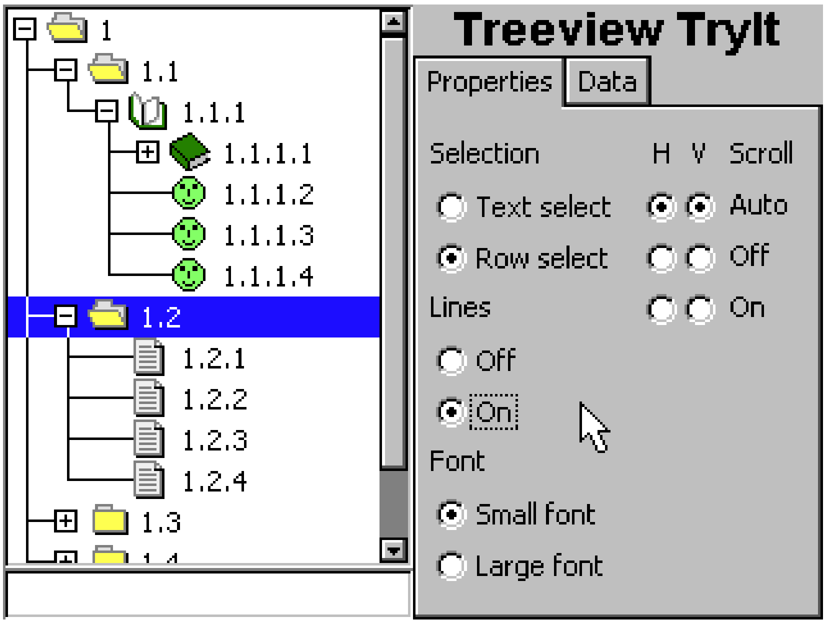
Table 19.953: TREEVIEW_ITEM_SetUserData() parameter list

19.27.7 Example

The `Sample` folder contains the following example which shows how the widget can be used: `WIDGET_TreeviewTryit.c`

Note that several other examples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_TreeviewTryit.c:



19.28 WINDOW: Window widget

The WINDOW widget is used to create a dialog window from a resource table. It should be used if the dialog should not look like a frame window. The window widget acts as background and as a container for child windows: It can contain child windows and fills the background, typically with gray.
It behaves much like a frame-window without frame and title bar and is used for dialogs.
All WINDOW-related routines are located in the file(s) WINDOW.c, DIALOG.h.

19.28.1 Configuration options

Type	Macro	Default	Description
S	WINDOW_BKCOLOR_DEFAULT (with WIDGET_USE_FLEX_SKIN == 0) (with WIDGET_USE_FLEX_SKIN == 1)	0xC0C0C0 GUI_WHITE	Default background color for new WINDOW widgets

Table 19.954: Configuration options

19.28.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

19.28.3 WINDOW API

The table below lists the available emWin WINDOW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
WINDOW_CreateEx()	Creates a WINDOW widget.
WINDOW_CreateIndirect()	Creates a WINDOW widget from a resource table entry.
WINDOW_CreateUser()	Creates a WINDOW widget using extra bytes as user data.
WINDOW_GetUserData()	Retrieves the data set with WINDOW_SetUserData().
WINDOW_SetBkColor()	Sets the background color of the given WINDOW widget.
WINDOW_SetDefaultBkColor()	Sets the default background color for WINDOW widgets.
WINDOW_SetUserData()	Sets the extra data of a WINDOW widget.

Table 19.955: WINDOW API list

WINDOW_CreateEx()

Description

Creates a WINDOW widget of a specified size at a specified location.

Prototype

```
WINDOW_Handle WINDOW_CreateEx(int           x0,      int y0,
                              int           xSize,   int ySize,
                              WM_HWIN       hParent, int WinFlags,
                              int           ExFlags, int Id,
                              WM_CALLBACK * cb);
```

Parameter	Description
x0	Leftmost pixel of the WINDOW widget (in parent coordinates)
y0	Topmost pixel of the WINDOW widget (in parent coordinates)
xSize	Size of the WINDOW widget in X
ySize	Size of the WINDOW widget in Y
hParent	Handle of parent window

Table 19.956: WINDOW_CreateEx() parameter list

Parameter	Description
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (refer to WM_CreateWindow() in the chapter “The Window Manager (WM)” on page 375 for a list of available parameter values)
ExFlags	Not used yet, reserved for future use
Id	Window ID of the WINDOW widget
cb	Pointer to callback routine.

Table 19.956: WINDOW_CreateEx() parameter list

Return value

Handle of the created WINDOW widget; 0 if the function fails.

WINDOW_CreateIndirect()

The prototype of this function is explained at the beginning of this chapter. Details can be found in the description of the function <WIDGET>_CreateIndirect(). The element Para of the according GUI_WIDGET_CREATE_INFO structure is not used. The element Flags is used according to the parameter WinFlags of the function WINDOW_CreateEx(). The Sample folder contains the file WIDGET_Window.c which shows how to use the WINDOW widget in a dialog resource.

WINDOW_CreateUser()

Prototype explained at the beginning of the chapter as <WIDGET>_CreateUser(). For a detailed description of the parameters the function WINDOW_CreateEx() can be referred to.

WINDOW_GetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_GetUserData().

WINDOW_SetBkColor()

Description

Sets the background color for the given WINDOW widget.

Prototype

```
void WINDOW_SetBkColor(WM_HWIN hObj, GUI_COLOR Color);
```

Parameter	Description
hObj	Handle of the WINDOW widget.
Color	Background color to be used.

Table 19.957: WINDOW_SetBkColor() parameter list

WINDOW_SetDefaultBkColor()

Description

Sets the default background color used for WINDOW widgets.

Prototype

```
void WINDOW_SetDefaultBkColor(GUI_COLOR Color);
```

Parameter	Description
Color	Color to be used.

Table 19.958: WINDOW_SetDefaultBkColor() parameter list

WINDOW_SetUserData()

Prototype explained at the beginning of the chapter as <WIDGET>_SetUserData().

Chapter 20

Dialogs

Widgets may be created and used on their own, as they are by nature windows themselves. However, it is often desirable to use dialog boxes, which are windows that contain one or more widgets.
A dialog box (or dialog) is normally a window that appears in order to request input from the user. It may contain multiple widgets, requesting information from the user through various selections, or it may take the form of a message box which simply provides information (such as a note or warning) and an "OK" button.
For common tasks like choosing a file, choosing a color or (as mentioned before) for showing simple text messages emWin offers ’common dialogs’. These dialogs can be configured to achieve the look and feel of the application.

20.1 Dialog basics

Input focus

The Window Manager remembers the window or window object that was last selected by the user with the touch-screen, mouse, keyboard, or other means. This window receives keyboard input messages and is said to have the input focus.
The primary reason for keeping track of input focus is to determine where to send keyboard commands. The window which has input focus will receive events generated by the keyboard.
To move the input focus within a dialog to the next focusable dialog item the key `GUI_KEY_TAB` can be used. To move backwards `GUI_KEY_BACKTAB` can be used.

Blocking vs. non-blocking dialogs

Dialog windows can be blocking or non-blocking.
A blocking dialog blocks the thread of execution. It has input focus by default and must be closed by the user before the thread can continue. A blocking dialog does not disable other dialogs shown at the same time. With other words a blocking dialog is not a modal dialog. Blocking means, the used functions (`GUI_ExecDialogBox()` or `GUI_ExecCreatedDialog()`) does not return until the dialog is closed.
A non-blocking dialog, on the other hand, does not block the calling thread -- it allows the task to continue while it is visible. The function returns immediately after creating the dialog.
Blocking functions should never be called from within a callback function. This may cause malfunction of the application.

Dialog procedure

A dialog box is a window, and it receives messages just as every other window in the system does. Most messages are handled automatically by the client callback routine of the dialog box. The others are passed to the client callback routine which is specified as a parameter upon creation. The client callback function is known as the dialog procedure.
Since a dialog itself usually consists of 2 windows (dialog and client window), messages have to be sent using the correct handle. After a dialog was created there is only one handle to the dialog. In order to access the client window, the function `WM_GetClientWindow()` should be used.

Dialog messages

There are two types of additional messages which are sent to the dialog procedure: `WM_INIT_DIALOG` and `WM_NOTIFY_PARENT`. The `WM_INIT_DIALOG` message is sent to the dialog procedure immediately before a dialog box is displayed. Dialog procedures typically use this message to initialize widgets and carry out any other initialization tasks that affect the appearance of the dialog box. The `WM_NOTIFY_PARENT` message is sent to the dialog box by its child windows in order to notify the parent of any events in order to ensure synchronization. The events sent by a child depend on its type and are documented separately for every type of widget.

20.2 Creating a dialog

Two basic things are required to create a dialog box: a resource table that defines the widgets to be included, and a dialog procedure which defines the initial values for the widgets as well as their behavior. Once both items exist, you need only a single function call (GUI_CreateDialogBox() or GUI_ExecDialogBox()) to actually create the dialog.

20.2.1 Resource table

Dialog boxes may be created in a blocking manner (using GUI_ExecDialogBox()) or as non-blocking (using GUI_CreateDialogBox()). A resource table must first be defined which specifies all widgets to be included in the dialog. The example shown below creates a resource table:

```
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Dialog", 0,                10,  10, 180, 230, 0,               0 },
  { BUTTON_CreateIndirect,   "OK",     GUI_ID_OK,       100,   5,  60,  20, 0,               0 },
  { BUTTON_CreateIndirect,   "Cancel", GUI_ID_CANCEL,   100,  30,  60,  20, 0,               0 },
  { TEXT_CreateIndirect,     "LText",  0,                10,  55,  48,  15, TEXT_CF_LEFT,    0 },
  { TEXT_CreateIndirect,     "RText",  0,                10,  80,  48,  15, TEXT_CF_RIGHT,   0 },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT0,     60,  55, 100,  15, 0,              50 },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT1,     60,  80, 100,  15, 0,              50 },
  { TEXT_CreateIndirect,     "Hex",    0,                10, 100,  48,  15, TEXT_CF_RIGHT,   0 },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT2,     60, 100, 100,  15, 0,               6 },
  { TEXT_CreateIndirect,     "Bin",    0,                10, 120,  48,  15, TEXT_CF_RIGHT,   0 },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT3,     60, 120, 100,  15, 0,               0 },
  { LISTBOX_CreateIndirect,  NULL,     GUI_ID_LISTBOX0,  10,  10,  48,  40, 0,               0 },
  { CHECKBOX_CreateIndirect, NULL,     GUI_ID_CHECK0,    10, 140,   0,   0, 0,               0 },
  { CHECKBOX_CreateIndirect, NULL,     GUI_ID_CHECK1,    30, 140,   0,   0, 0,               0 },
  { SLIDER_CreateIndirect,   NULL,     GUI_ID_SLIDER0,   60, 140, 100,  20, 0,               0 },
  { SLIDER_CreateIndirect,   NULL,     GUI_ID_SLIDER1,   10, 170, 150,  30, 0,               0 }
};
```

Widgets can be included in a dialog by using the <WIDGET>_CreateIndirect() function for indirect creation. Detailed information can be found in the chapter "Widgets (window objects)" on page 449.

20.2.2 Dialog procedure

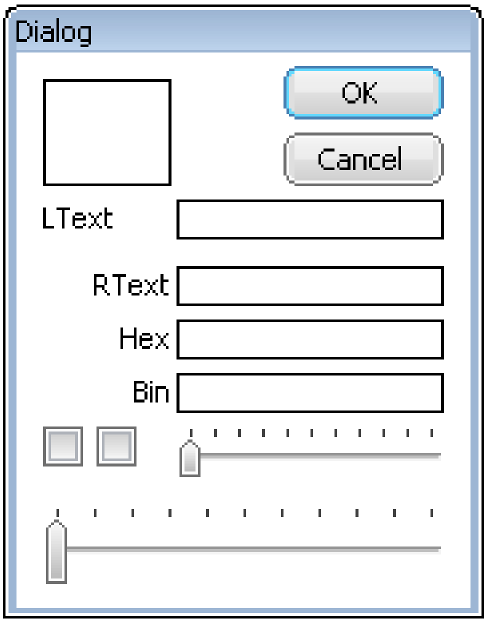
The example above has been created using the blank dialog procedure shown below. This is the basic template which should be used as a starting point when creating any dialog procedure:

```
/*********************************************************************
*
*       Dialog procedure
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  default:
    WM_DefaultProc(pMsg);
  }
}
```

For this example, the dialog box is displayed with the following line of code:

```
GUI_ExecDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate),
                  _cbCallback, 0, 0, 0);
```

The resulting dialog box looks as follows, or similar (the actual appearance will depend on your configuration and default settings):



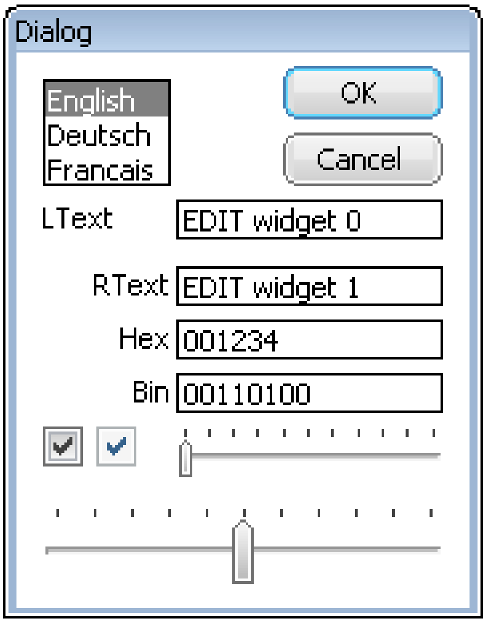
After creation of the dialog box, all widgets included in the resource table will be visible, although as can be seen in the previous screenshot, they will appear "empty". This is because the dialog procedure does not yet contain code that initializes the individual elements. The initial values of the widgets, the actions caused by them, and the interactions between them need to be defined in the dialog procedure.

20.2.2.1 Initializing the dialog

The typical next step is to initialize the widgets with their respective initial values. This is normally done in the dialog procedure as a reaction to the `WM_INIT_DIALOG` message. The program excerpt below illustrates things:

```
/*********************************************************************
*
*       Dialog procedure
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
  WM_HWIN hItem;
  WM_HWIN hWin;

  hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    hItem = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
    EDIT_SetText(hItem, "EDIT widget 0");
    hItem = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
    EDIT_SetText(hItem, "EDIT widget 1");
    EDIT_SetTextAlign(hItem, GUI_TA_LEFT);
    hItem = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
    EDIT_SetHexMode(hItem, 0x1234, 0, 0xffff);
    hItem = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
    EDIT_SetBinMode(hItem, 0x1234, 0, 0xffff);

    hItem = WM_GetDialogItem(hWin, GUI_ID_CHECK0);
    CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK0));
    hItem = WM_GetDialogItem(hWin, GUI_ID_CHECK1);
    WM_DisableWindow(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
    CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK1));

    hItem = WM_GetDialogItem(hWin, GUI_ID_SLIDER0);
    SLIDER_SetWidth(WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
    hItem = WM_GetDialogItem(hWin, GUI_ID_SLIDER1);
    SLIDER_SetValue(WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);

    hItem = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
    LISTBOX_SetText(hItem, _apListBox);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}
```


The initialized dialog box now appears as follows, with all widgets containing their initial values:



20.2.2.2 Defining dialog behavior

Once the dialog has been initialized, all that remains is to add code to the dialog procedure which will define the behavior of the widgets, making them fully operable. Continuing with the same example, the final dialog procedure is shown below:

```
/*********************************************************************
*
*       Dialog procedure
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
  WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3;
  WM_HWIN hListBox;
  WM_HWIN hWin
  int     NCode
  int     Id;

  hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    /* Get window handles for all widgets */
    hEdit0   = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
    hEdit1   = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
    hEdit2   = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
    hEdit3   = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
    hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
    /* Initialize all widgets */
    EDIT_SetText(hEdit0, "EDIT widget 0");
    EDIT_SetText(hEdit1, "EDIT widget 1");
    EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
    EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
    EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
    LISTBOX_SetText(hListBox, _apListBox);
    WM_DisableWindow(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
    CHECKBOX_Check(  WM_GetDialogItem(hWin, GUI_ID_CHECK0));
    CHECKBOX_Check(  WM_GetDialogItem(hWin, GUI_ID_CHECK1));
    SLIDER_SetWidth( WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
    SLIDER_SetValue( WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);
    break;
  case WM_KEY:
    switch (((WM_KEY_INFO*)(pMsg->Data.p))->Key) {
    case GUI_ID_ESCAPE:
      GUI_EndDialog(hWin, 1);
      break;
    case GUI_ID_ENTER:
      GUI_EndDialog(hWin, 0);
      break;
    }
    break;
```

```
    case WM_NOTIFY_PARENT:
      Id    = WM_GetId(pMsg->hWinSrc);     /* Id of widget */
      NCode = pMsg->Data.v;                /* Notification code */
      switch (NCode) {
      case WM_NOTIFICATION_RELEASED:       /* React only if released */
        if (Id == GUI_ID_OK) {             /* OK Button */
          GUI_EndDialog(hWin, 0);
        }
        if (Id == GUI_ID_CANCEL) {         /* Cancel Button */
          GUI_EndDialog(hWin, 1);
        }
        break;
      case WM_NOTIFICATION_SEL_CHANGED:    /* Selection changed */
        FRAMEWIN_SetText(hWin, "Dialog - sel changed");
        break;
      default:
        FRAMEWIN_SetText(hWin, "Dialog - notification received");
      }
      break;
    default:
      WM_DefaultProc(pMsg);
    }
  }
```


20.3 Dialog API

The table below lists the available dialog-related routines in alphabetical. Detailed descriptions follow:

Routine	Description
GUI_CreateDialogBox()	Creates a non-blocking dialog.
GUI_ExecCreatedDialog()	Executes an already created dialog.
GUI_ExecDialogBox()	Creates and execute a dialog.
GUI_EndDialog()	Ends a dialog box.

Table 20.1: Dialog API list

GUI_CreateDialogBox()

Description

Creates a dialog box.

Prototype

```
WM_HWIN GUI_CreateDialogBox(const GUI_WIDGET_CREATE_INFO * paWidget,
                            int      NumWidgets, WM_CALLBACK * cb,
                            WM_HWIN  hParent,    int           x0,
                            int      y0);
```

Parameter	Description
paWidget	Pointer to resource table defining the widgets to be included in the dialog.
NumWidgets	Total number of widgets included in the dialog.
cb	Pointer to an application-specific callback function (dialog procedure).
hParent	Handle of parent window (0 = no parent window).
x0	X-position of the dialog relative to parent window.
y0	Y-position of the dialog relative to parent window.

Table 20.2: GUI_CreateDialogBox() parameter list

Additional information

The parameter cb is used as callback function for the client window. If a callback function should be set for the dialog window, the function `WM_SetCallback()` should be used with the handle returned by `GUI_CreateDialogBox()`.
The handle of the client window can be determined using the function `WM_GetClientWindow()` passing the handle of the dialog which was returned by `GUI_CreateDialogBox()`.

Return value

Handle of the created dialog. This handle can be used to access the first widget from the resource table. This should be a FRAMEWIN or WINDOW widget.

GUI_ExecCreatedDialog()

Description

Executes an already created dialog box.

Prototype

```
int GUI_ExecCreatedDialog(WM_HWIN hDialog);
```

Parameter	Description
hDialog	Handle to dialog box.

Table 20.3: GUI_ExecCreatedDialog() parameter list

Additional information

This is a blocking function. It does not return until the dialog is closed using the function `GUI_EndDialog()`. The `WM_CF_SHOW` flag is set, so the dialog is drawn the next time the Window Manager is executed.

Return value

Return value which was passed to `GUI_EndDialog()`.

GUI_ExecDialogBox()

Description

Creates and executes a dialog box.

Prototype

```
int GUI_ExecDialogBox(const GUI_WIDGET_CREATE_INFO * paWidget,
                      int                            NumWidgets,
                      WM_CALLBACK                  * cb,
                      WM_HWIN                        hParent,
                      int                            x0,
                      int                            y0);
```

Parameter	Description
paWidget	Pointer to a resource table defining the widgets to be included in the dialog.
NumWidgets	Total number of widgets included in the dialog.
cb	Pointer to an application-specific callback function (dialog procedure).
hParent	Handle of parent window (0 = no parent window).
x0	X-position of the dialog relative to parent window.
y0	Y-position of the dialog relative to parent window.

Table 20.4: GUI_ExecDialogBox() parameter list

Return value

Return value which was passed to `GUI_EndDialog()`.

Additional information

This function actually calls GUI_CreateDialogBox() and GUI_ExecCreatedDialog().
It is blocking and therefor does not return until the dialog is closed using the function `GUI_EndDialog()`. The `WM_CF_SHOW` flag is set, so the dialog is drawn the next time the Window Manager is executed.

GUI_EndDialog()

Description

Ends (closes) a dialog box. The dialog and its child windows will be removed from memory.

Prototype

```
void GUI_EndDialog(WM_HWIN hDialog, int r);
```

Parameter	Description
hDialog	Handle to dialog box.
r	Value which is returned by the function `GUI_ExecDialogBox()`. This value is ignored in case a non-blocking dialog is closed.

Table 20.5: GUI_EndDialog() parameter list

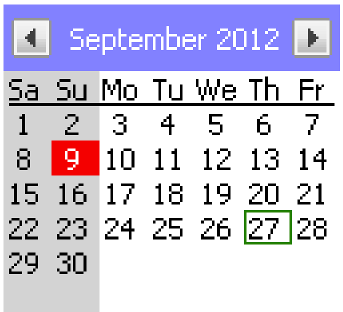
Additional information

The handle `hDialog` is no longer valid after this function was called. In case a non-blocking dialog is closed, this function works the same way the function `WM_DeleteWindow()` works.

20.4 Common dialogs

Common dialogs can be used by an application for several tasks. They can be opened by calling a simple function instead of creating a new and complex dialog by the application. The following shows the available common dialogs.

20.4.1 CALENDAR



The CALENDAR dialog can be used for selecting or setting a date. The dialog consists of 2 buttons for month wise scrolling, a text which shows the current year and month and a pad of days. A small surrounding frame is shown surrounding the current date and the current selection is highlighted. The keyboard and / or the pointer input device (PID) can be used for selecting a date. The dialog supports the Gregorian calendar which is used since 1582.

20.4.1.1 Notification codes

The following events are sent from the dialog to its parent window as part of a `WM_NOTIFY_PARENT` message:

Message	Description
CALENDAR_NOTIFICATION_MONTH_CLICKED	Month/year-text has been clicked.
CALENDAR_NOTIFICATION_MONTH_RELEASED	Month/year-text has been released.
WM_NOTIFICATION_CLICKED	Widget has been clicked.
WM_NOTIFICATION_RELEASED	Widget has been released.
WM_NOTIFICATION_SCROLL_CHANGED	One of the scroll buttons has been pressed.
WM_NOTIFICATION_SEL_CHANGED	The selection has been changed.

Table 20.6: Notification codes

20.4.1.2 Keyboard reaction

The dialog reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_PGUP	Selection moves one month back.
GUI_KEY_PGDOWN	Selection moves one month forward.
GUI_KEY_LEFT	Selection moves to the left.
GUI_KEY_RIGHT	Selection moves to the right.
GUI_KEY_UP	Selection moves one line up.
GUI_KEY_DOWN	Selection moves one line down.

Table 20.7: Keyboard reaction

20.4.1.3 CALENDAR API

The table below lists the available CALENDAR-related routines in alphabetical order. Detailed descriptions follow.

Routine	Description
CALENDAR_Create()	Creates a CALENDAR dialog.
CALENDAR_GetDate()	Returns the current date.
CALENDAR_GetSel()	Returns the selected date.
CALENDAR_SetDate()	Sets the current date.
CALENDAR_SetSel()	Sets the selected date.
CALENDAR_SetDefaultBkColor()	Sets the default background colors to be used.
CALENDAR_SetDefaultColor()	Sets the default foreground colors to be used.
CALENDAR_SetDefaultDays()	Sets the strings to be used to label the days.
CALENDAR_SetDefaultFont()	Sets the default font to be used.
CALENDAR_SetDefaultMonths()	Sets the strings to be used to label the months.
CALENDAR_SetDefaultSize()	Sets the default sizes to be used.

Table 20.8: CALENDAR API list

CALENDAR_Create()

Description

Creates a CALENDAR dialog.

Prototype

```
WM_HWIN CALENDAR_Create(WM_HWIN hParent, int xPos, int yPos,
                        unsigned Year, unsigned Month,
                        unsigned Day,  unsigned FirstDayOfWeek,
                        int      Id,   int      Flags);
```

Parameter	Description
hParent	Handle of the parent window which should receive the notification messages.
xPos	X position in pixels of the dialog in client coordinates.
yPos	Y position in pixels of the dialog in client coordinates.
Year	Current year (1582-9999).
Month	Current month (1-12).
Day	Current day (1-31).
FirstDayOfWeek	First weekday to be used (0=SA, 1=SO, ... , 6=FR).
Id	Id to be used for the CALENDAR dialog.
Flags	Additional flags for the WINDOW widget.

Table 20.9: CALENDAR_Create() parameter list

Return value

Handle of the dialog on success, otherwise 0.

Additional information

`Year`, `month` and `day` specify the current date. Per default this is also the initial selection. `FirstDayOfWeek` determines an offset for the first day to be shown. Default is showing Saturday at first.

CALENDAR_GetDate()

Description

Returns the current date.

Prototype

```
void CALENDAR_GetDate(WM_HWIN hWin, CALENDAR_DATE * pDate);
```

Parameter	Description
hWin	Handle of CALENDAR dialog.
pDate	Pointer to a CALENDAR_DATE structure.

Table 20.10: CALENDAR_GetDate() parameter list

Elements of structure CALENDAR_DATE

Data type	Element	Description
int	Year	Year of requested date.
int	Month	Month of requested date.
int	Day	Day of requested date.

Table 20.11: CALENDAR_DATE element list

Additional information

Current date and selected date are different items. The selection can be moved by the keyboard interface and/or the PID whereas the current date can be specified when creating the dialog or by using the function `CALENDAR_SetDate();`

CALENDAR_GetSel()

Description

Returns the currently selected date.

Prototype

```
void CALENDAR_GetSel(WM_HWIN hWin, CALENDAR_DATE * pDate);
```

Parameter	Description
hWin	Handle of CALENDAR dialog.
pDate	Pointer to a CALENDAR_DATE structure.

Table 20.12: CALENDAR_GetSel() parameter list

Additional information

The elements of `CALENDAR_DATE` are described in the description of "CALENDAR_GetDate()" on page 840.

CALENDAR_SetDate()

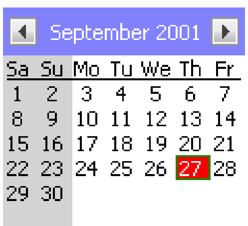
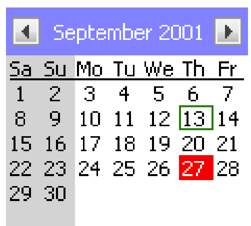
Before	After
	

Table 20.13: CALENDAR_SetDate() before after screenshots

Description

Sets the current date.

Prototype

```
void CALENDAR_SetDate(WM_HWIN hWin, CALENDAR_DATE * pDate);
```

Parameter	Description
hWin	Handle of CALENDAR dialog.
pDate	Pointer to a CALENDAR_DATE structure.

Table 20.14: CALENDAR_SetDate() parameter list

Additional information

The elements of CALENDAR_DATE are described in the description of "CALENDAR_GetDate()" on page 840.

CALENDAR_SetSel()

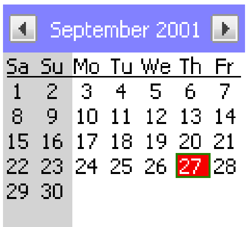
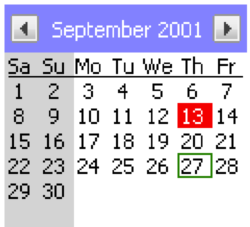
Before	After
	

Table 20.15: CALENDAR_SetSel() before after screenshots

Description

Sets the currently selected date.

Prototype

```
void CALENDAR_SetSel(WM_HWIN hWin, CALENDAR_DATE * pDate);
```

Parameter	Description
hWin	Handle of CALENDAR dialog.
pDate	Pointer to a CALENDAR_DATE structure.

Table 20.16: CALENDAR_SetSel() parameter list

Additional information

The elements of CALENDAR_DATE are described in the description of "CALENDAR_GetDate()" on page 840.

CALENDAR_SetDefaultBkColor()

Description

Sets the default background color to be used for new CALENDAR dialogs.

Prototype

```
void CALENDAR_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Description
Index	(see table below)
Color	Color to be used

Table 20.17: CALENDAR_SetDefaultBkColor() parameter list

Permitted values for parameter Index	
CALENDAR_CI_WEEKEND	Background color to be used for weekend days.
CALENDAR_CI_WEEKDAY	Background color to be used for weekdays.
CALENDAR_CI_SEL	Background color to be used for the selection.
CALENDAR_CI_HEADER	Background color to be used for the header area.

CALENDAR_SetDefaultColor()

Description

Sets the default color to be used for new CALENDAR dialogs.

Prototype

```
void CALENDAR_SetDefaultColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Description
Index	(see table below)
Color	Color to be used

Table 20.18: CALENDAR_SetDefaultColor() parameter list

Permitted values for parameter Index	
CALENDAR_CI_WEEKEND	Color to be used for weekend days.
CALENDAR_CI_WEEKDAY	Color to be used for weekdays.
CALENDAR_CI_SEL	Color to be used for the selection.
CALENDAR_CI_MONTH	Color to be used for the month (and year) text.
CALENDAR_CI_LABEL	Color to be used for labeling the days.
CALENDAR_CI_FRAME	Color to be used for the frame of the current date.

CALENDAR_SetDefaultDays()

Description

Sets the text to be used to label the days.

Prototype

```
void CALENDAR_SetDefaultDays(const char ** apDays);
```

Parameter	Description
apDays	Pointer to an array of 7 string pointers containing the strings to be used.

Table 20.19: CALENDAR_SetDefaultDays() parameter list

Additional information

The first string of the array should point to the abbreviation of Saturday, the second to Sunday and so on. The array needs to have at least 7 strings. If there are too few strings passed to the function the behavior of emWin becomes undefined.

CALENDAR_SetDefaultFont()

Description

Sets the font(s) to be used for drawing the CALENDAR items.

Prototype

```
void CALENDAR_SetDefaultFont(unsigned Index, const GUI_FONT * pFont);
```

Parameter	Description
Index	See table below.
pFont	Font to be used.

Table 20.20: CALENDAR_SetDefaultFont() parameter list

Permitted values for parameter Index	
CALENDAR_FI_CONTENT	Font to be used for labeling and the numbers.
CALENDAR_FI_HEADER	Font to be used for month / year.

CALENDAR_SetDefaultMonths()

Description

Sets the text to be used for the current month / year.

Prototype

```
void CALENDAR_SetDefaultMonths(const char ** apMonths);
```

Parameter	Description
apMonth	Pointer to an array of 12 string pointers containing the strings to be used.

Table 20.21: CALENDAR_SetDefaultMonths() parameter list

Additional information

The first string of the array should point to the text for 'January', the second to the text for 'February' and so on. The array needs to have at least 12 strings. If there are too few strings passed to the function the behavior of emWin becomes undefined.

CALENDAR_SetDefaultSize()

Description

Sets the sizes to be used by the dialog.

Prototype

```
void CALENDAR_SetDefaultSize(unsigned Index, unsigned Size);
```

Parameter	Description
Index	(see table below)
Size	Size to be used.

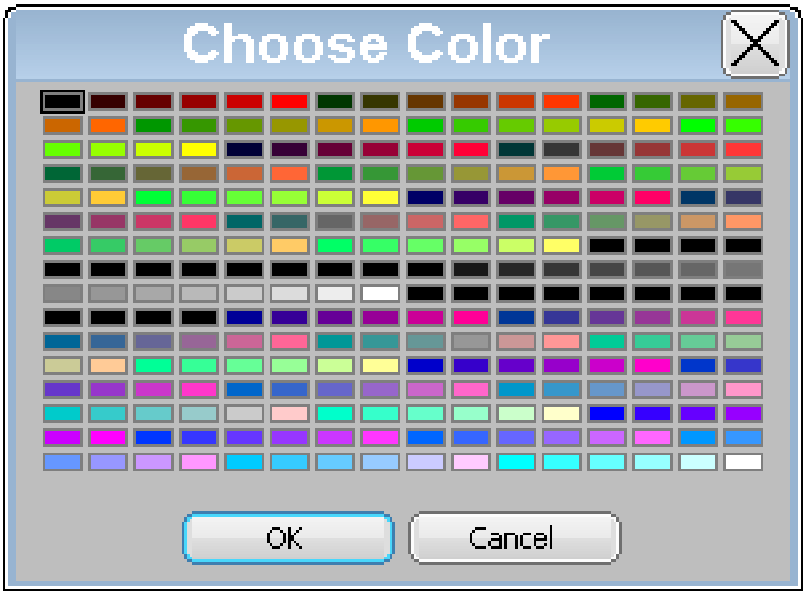
Table 20.22: CALENDAR_SetDefaultSize() parameter list

Permitted values for parameter Index	
CALENDAR_SI_HEADER	Y-size in pixels used for the header area. (default is 25)
CALENDAR_SI_CELL_X	Cell size in X to be used for one item in the day pad. (default is 18)
CALENDAR_SI_CELL_Y	Cell size in Y to be used for one item in the day pad. (default is 13)

Additional information

The size in x of the complete dialog can be calculated as follows:
xSizeDialog = 7 x CellSizeX
The size in y of the complete dialog can be calculated as follows:
ySizeDialog = 7 x CellSizeY + HeaderSizeY

20.4.2 CHOOSECOLOR



The CHOOSECOLOR dialog can be used to select a color from a given color array.

20.4.2.1 Notification codes

The following events are sent from the dialog to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Description
WM_NOTIFICATION_SEL_CHANGED	Sent immediately after a new color has been selected by the PID or the keyboard.
WM_NOTIFICATION_CHILD_DELETED	Sent when the dialog has been closed.
WM_NOTIFICATION_VALUE_CHANGED	Sent if the dialog was closed using the 'Ok' button with a different than the initial selection.

Table 20.23: Notification codes

20.4.2.2 Keyboard reaction

The dialog reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_ESCAPE	Dialog execution will be cancelled.
GUI_KEY_ENTER	Reaction depends on the focussed button.
GUI_KEY_LEFT	Cursor moves to the left.
GUI_KEY_RIGHT	Cursor moves to the right.
GUI_KEY_UP	Cursor moves one line up.
GUI_KEY_DOWN	Cursor moves one line down.

Table 20.24: Keyboard reaction

20.4.2.3 CHOOSECOLOR API

The table below lists the available CHOOSECOLOR-related routines in alphabetical order. Detailed descriptions follow.

Routine	Description
CHOOSECOLOR_Create()	Creates a CHOOSECOLOR dialog.
CHOOSECOLOR_GetSel()	Returns the index of the currently selected color.
CHOOSECOLOR_SetSel()	Selects the given color.
CHOOSECOLOR_SetDefaultColor()	Sets the default colors for color frame and focus.

Table 20.25: CHOOSECOLOR API list

Routine	Description
CHOOSECOLOR_SetDefaultSpace()	Sets the default item space.
CHOOSECOLOR_SetDefaultBorder()	Sets the default border space.
CHOOSECOLOR_SetDefaultButtonSize()	Sets the default button size.

Table 20.25: CHOOSECOLOR API list

CHOOSECOLOR_Create()

Description

Creates a dialog for choosing a color and returns immediately.

Prototype

```
WM_HWIN CHOOSECOLOR_Create(WM_HWIN hParent, int xPos, int yPos,
                           int xSize, int ySize, const GUI_COLOR * pColor,
                           unsigned NumColors, unsigned NumColorsPerLine,
                           int Sel, char * sCaption, int Flags);
```

Parameter	Description
hParent	Handle of the parent window which should receive the notification messages.
xPos	X position in pixels of the dialog in client coordinates.
yPos	Y position in pixels of the dialog in client coordinates.
xSize	X-size of the dialog in pixels.
ySize	Y-size of the dialog in pixels.
pColor	Pointer to an array of 32 bit color values containing the colors to be used.
NumColors	Number of colors to be shown.
NumColorsPerLine	Number of colors to be shown per line.
Sel	Initial index value to be used for the selection / focus.
sCaption	Title to be shown in the title bar.
Flags	Additional flags for the FRAMEWIN widget.

Table 20.26: CHOOSECOLOR_Create() parameter list

Return value

Handle of the dialog on success, otherwise 0.

Additional information

The following default values are used:
- If (xPos < 0) the dialog will be centered horizontally.
- If (yPos < 0) the dialog will be centered vertically.
- If (xSize == 0) the half of the display size in x will be used.
- If (ySize == 0) the half of the display size in y will be used.
- if (sCaption == NULL) 'Choose Color' will be shown in the title bar.

As mentioned above the creation routine returns immediately. It becomes visible with the next call of `WM_Exec()` or it can be executed with `GUI_ExecCreatedDialog()`.

CHOOSECOLOR_GetSel()

Description

Returns the index of the currently selected color.

Prototype

```
int CHOOSECOLOR_GetSel(WM_HWIN hObj);
```

Parameter	Description
hObj	Handle of the CHOOSECOLOR dialog.

Table 20.27: CHOOSECOLOR_GetSel() parameter list

Return value

Index of the currently selected color.

CHOOSECOLOR_SetSel()

Description

Sets the current selection.

Prototype

```
void CHOOSECOLOR_SetSel(WM_HWIN hObj, int Sel);
```

Parameter	Description
hObj	Handle of the CHOOSECOLOR dialog.
Sel	New selection to be used.

Table 20.28: CHOOSECOLOR_SetSel() parameter list

Additional information

The given selection should be smaller than the number of colors. In case of a negative value no initial selection will be shown.

CHOOSECOLOR_SetDefaultColor()

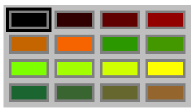
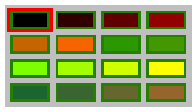
Before	After
	

Table 20.29: CHOOSECOLOR_SetDefaultColor() before after screenshots

Description

Sets the colors to be used to draw the surrounding frame of the colors.

Prototype

```
void CHOOSECOLOR_SetDefaultColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Description
Index	See table below.
Color	Color to be used.

Table 20.30: CHOOSECOLOR_SetDefaultColor() parameter list

Permitted values for parameter Index	
CHOOSECOLOR_CI_FRAME	Color to be used to draw the frame surrounding each color. Default is GUI_GRAY.
CHOOSECOLOR_CI_FOCUS	Color to be used to draw the focus rectangle. Default is GUI_BLACK.

CHOOSECOLOR_SetDefaultSpace()

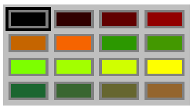
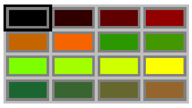
Before	After
	

Table 20.31: CHOOSECOLOR_SetDefaultSpace() before after screenshots

Description

Determines the space between the color rectangles.

Prototype

```
void CHOOSECOLOR_SetDefaultSpace(unsigned Index, unsigned Space);
```

Parameter	Description
Index	See table below.
Space	Space in pixels to be used.

Table 20.32: CHOOSECOLOR_SetDefaultSpace() parameter list

Permitted values for parameter Index	
GUI_COORD_X	Space in X to be used between the colors. Default value is 5.
GUI_COORD_Y	Space in Y to be used between the colors. Default value is 5.

CHOOSECOLOR_SetDefaultBorder()

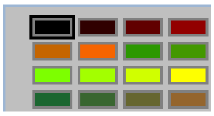
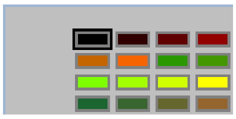
Before	After
	

Table 20.33: CHOOSECOLOR_SetDefaultBorder() before after screenshots

Description

Sets the size of the border between the colors and the dialog frame to be used.

Prototype

```
void CHOOSECOLOR_SetDefaultBorder(unsigned Index, unsigned Border);
```

Parameter	Description
Index	See table below.
Border	Border to be used.

Table 20.34: CHOOSECOLOR_SetDefaultBorder() parameter list

Permitted values for parameter Index	
GUI_COORD_X	Space in X to be used between border and colors. Default value is 4.
GUI_COORD_Y	Space in Y to be used between border and colors. Default value is 4.

Additional information

The horizontal value is also used to determine the space between the buttons.

CHOOSECOLOR_SetDefaultButtonSize()

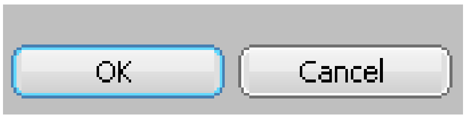
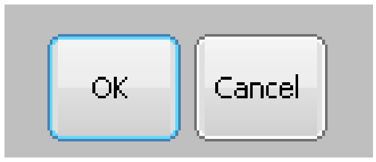
Before	After
	

Table 20.35: CHOOSECOLOR_SetDefaultButtonSize() before after screenshots

Description

Sets the button size to be used.

Prototype

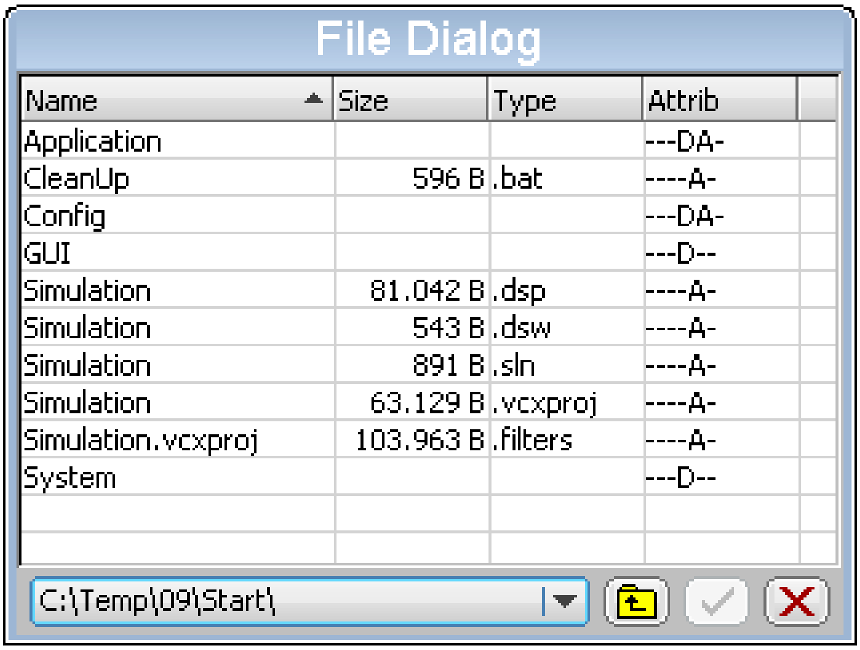
```
void CHOOSECOLOR_SetDefaultButtonSize(unsigned Index, unsigned ButtonSize);
```

Parameter	Description
`Index`	See table below.
`ButtonSize`	Size in pixels to be used.

Table 20.36: CHOOSECOLOR_SetDefaultButtonSize() parameter list

Permitted values for parameter `Index`	
`GUI_COORD_X`	Button size in X.
`GUI_COORD_Y`	Button size in Y.

20.4.3 CHOOSEFILE



The CHOOSEFILE dialog can be used for browsing through a directory and for selecting a file. It uses a user defined callback routine for retrieving data. So it can be used with any file system.

20.4.3.1 Configuration options

Type	Macro	Default	Description
N	CHOOSEFILE_DELIM	\	Default delimiter to be used.

Table 20.37: Configuration options

20.4.3.2 Keyboard reaction

The dialog reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_TAB	The next widget of the dialog gains the input focus.
GUI_KEY_BACKTAB	The previous widget of the dialog gains the input focus.
GUI_KEY_ENTER	The behavior depends on the currently focussed widget.
GUI_KEY_ESCAPE	Dialog will be cancelled.

Table 20.38: Keyboard reaction

20.4.3.3 Path- and file names

The maximum length of path- and file names is limited to 256 bytes.

20.4.3.4 CHOOSEFILE API

The table below lists the available CHOOSEFILE-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
CHOOSEFILE_Create()	Creates a CHOOSEFILE dialog.
CHOOSEFILE_EnableToolTips()	Enables ToolTips for the dialog. Default is disabled.
CHOOSEFILE_SetButtonText()	Sets the text of the given button.
CHOOSEFILE_SetDefaultButtonText()	Sets the default text of the given button.

Table 20.39: CHOOSEFILE API list

Routine	Description
CHOOSEFILE_SetDelim()	Sets the delimiter. Default is a backslash ('\').
CHOOSEFILE_SetToolTips()	Sets the text to be shown by the ToolTips.
CHOOSEFILE_SetTopMode()	Makes the button bar visible at the top of the dialog.

Table 20.39: CHOOSEFILE API list

CHOOSEFILE_Create()

Description

Creates a CHOOSEFILE dialog using the given parameters.

Prototype

```
WM_HWIN CHOOSEFILE_Create(WM_HWIN hParent, int xPos, int yPos,
                          int xSize, int ySize, const char * apRoot[],
                          int NumRoot, int SelRoot, const char * sCaption,
                          int Flags, CHOOSEFILE_INFO * pInfo);
```

Parameter	Description
hParent	Handle of parent window.
xPos	X position in pixels of the dialog in client coordinates.
yPos	Y position in pixels of the dialog in client coordinates.
xSize	X-size of the dialog in pixels.
ySize	Y-size of the dialog in pixels.
apRoot	Pointer to an array of strings containing the root directories to be used.
NumRoot	Number of root directories.
SelRoot	Initial index of the root directory to be used.
sCaption	Title to be shown in the title bar.
Flags	Additional flags for the FRAMEWIN widget.
pInfo	Pointer to a CHOOSEFILE_INFO structure.

Table 20.40: CHOOSEFILE_Create()

Return value

Handle of the dialog on success, otherwise 0.

Elements of structure CHOOSEFILE_INFO

Data type	Element	Description
int	Cmd	See table below.
const char *	pMask	This parameter is passed to the GetData() function and contains a mask which can be used for filtering the search result.
char *	pName	Pointer to the file name of the requested file.
char *	pExt	Pointer to the extension of the requested file.
char *	pAttrib	Pointer to the attribute string of the requested file.
U32	SizeL	Lower 32 bit of the file size.
U32	SizeH	Upper 32 bit of the file size.
U32	Flags	If the requested file is a directory it should be set to CHOOSEFILE_FLAG_DIRECTORY, otherwise it should be set to 0.
char *	pRoot	Pointer to a string containing the complete path of the currently used directory.
int (*)(CHOOSEFILE_INFO *)	pfGetData	Pointer to the GetData() function to be used.

Table 20.41: CHOOSEFILE_INFO element list

Permitted values for element Cmd	
CHOOSEFILE_FINDFIRST	The first entry of the current directory should be returned.
CHOOSEFILE_FINDNEXT	The next entry of the current directory should be returned.

Element CHOOSEFILE_FINDFIRST

This command is sent to the given callback routine to get the first entry of the current directory. The element pRoot of the CHOOSEFILE_INFO structure pointed by the parameter pInfo of the callback function contains the path to be used.
The following elements of the CHOOSEFILE_INFO structure should be used by the application to return information of the requested file: pName, pExt, pAttrib, SizeL, SizeH and Flags.
The parameter pAttrib contains a string to be shown in the 'Attrib' column. This string has to be build by the application. So each attributes independent of the used file system can be shown.
All strings used to return information about the file are copied by the dialog into its own memory locations.
If no file could be found the GetData() function should return 1.

Element CHOOSEFILE_FINDNEXT

This command is sent to the given callback routine to get the next entry of the chosen directory. If no further file could be found the GetData() function should return 1.

Parameter apRoot

This parameter should point to an array of string pointers containing the root directories shown in the DROPDOWN widget of the dialog. The directory names do not need to have a delimiter (slash or backslash) at the end. They are copied by the function to their own locations and do not need to remain valid after creating the dialog. Empty strings are not supported and could lead to an undefined behavior of the dialog.

Prototype of GetData() function

`int (*)(CHOOSEFILE_INFO * pInfo);`

Parameter	Description
`pInfo`	Pointer to a CHOOSEFILE_INFO structure.

Table 20.42: GetData() parameter list

Details about GetData() function

The `GetData()` function pointed by the element `pfGetData` has to be provided by the application. This function is responsible to pass information about the requested file to the dialog. It gets a pointer to a `CHOOSEFILE_INFO` structure which contains all details of the requested file.
The following elements are passed by the dialog to the application:

- **Cmd**
  Determines if information about the first- or the next file should be returned.
- **pRoot**
  Pointer to a string containing the path of the directory to be used.

The `GetData()` function then has to use the following elements for providing information about the requested file to the dialog:

- **pAttrib**
  Should point to a string which is shown in the 'Type' column. Because the CHOOSEFILE dialog can be used with any file system there are no special flags but a string which should be passed by the application to the dialog.
- **pName**
  Should point to a string which contains the file name without path and extension. Shown in the 'Name' column of the dialog.
- **pExt**
  Should point to a string which contains the extension of the file shown in the 'Type' column of the dialog
- **SizeL**
  Should be set to the lower 32 bit of the file length.
- **SizeH**
  Should be set to the upper 32 bit of the file length in case of file larger than 4.294.967.295 bytes.
- **Flags**
  If the requested file is a directory this element has to be set to `CHOOSEFILE_FLAG_DIRECTORY`. Otherwise it has to be 0.

Additional information

The following default values are used:

- If (`xPos` < 0) the dialog will be centered horizontally.
- If (`yPos` < 0) the dialog will be centered vertically.
- If (`xSize` == 0) the half of the display size in x will be used.
- If (`ySize` == 0) the half of the display size in y will be used.
- if (`sCaption` == NULL) 'Choose File' will be shown in the title bar.

Example of GetData() function

The following shows an example of the GetData() function which can be used with WIN32. The sample folder also contains a sample which can be used with emFile. Here the WIN32 example:

```
static const struct {
  U32 Mask;
  char c;
} _aAttrib[] = {
  { FILE_ATTRIBUTE_READONLY , 'R' },
  { FILE_ATTRIBUTE_HIDDEN   , 'H' },
  { FILE_ATTRIBUTE_SYSTEM   , 'S' },
  { FILE_ATTRIBUTE_DIRECTORY, 'D' },
  { FILE_ATTRIBUTE_ARCHIVE  , 'A' },
  { FILE_ATTRIBUTE_NORMAL   , 'N' },
};

static int _GetData(CHOOSEFILE_INFO * pInfo) {
```



```
  static HANDLE hFind;
  static int NewDir;
  static char acDrive [_MAX_DRIVE];
  static char acDir   [_MAX_DIR];
  static char acName  [_MAX_FNAME];
  static char acExt   [_MAX_EXT];
  static char acMask  [_MAX_PATH];
  static char acPath  [_MAX_PATH];
  static char acAttrib[10] = {0};
  WIN32_FIND_DATA Context;
  int i, r;
  char c;

  switch (pInfo->Cmd) {
  case CHOOSEFILE_FINDFIRST:
    if (hFind != 0) {
      FindClose(hFind);
    }
    //
    // Split path into drive and directory
    //
    _splitpath(pInfo->pRoot, acDrive, acDir, NULL, NULL);
    NewDir = 1;
    //
    // Do not 'break' here...
    //
  case CHOOSEFILE_FINDNEXT:
    if (NewDir) {
      _makepath(acMask, acDrive, acDir, NULL, NULL);
      strcat(acMask, pInfo->pMask);
      hFind = FindFirstFile(acMask, &Context);
      if (hFind == INVALID_HANDLE_VALUE) {
        FindClose(hFind);
        hFind = 0;
        return 1;
      }
    } else {
      r = FindNextFile(hFind, &Context);
      if (r == 0) {
        FindClose(hFind);
        hFind = 0;
        return 1;
      }
    }
    NewDir = 0;
    //
    // Generate attribute string (pInfo->pAttrib)
    //
    for (i = 0; i < GUI_COUNTOF(_aAttrib); i++) {
      c = (Context.dwFileAttributes & _aAttrib[i].Mask) ? _aAttrib[i].c : '-';
      acAttrib[i] = c;
    }
    //
    // Make name and extension (pInfo->pName, pInfo->pExt)
    //
    if ((Context.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) == 0) {
      _splitpath(Context.cFileName, NULL, NULL, acName, acExt);
    } else {
      strcpy(acName, Context.cFileName);
      acExt[0] = 0;
    }
    //
    // Pass data to dialog
    //
    pInfo->pAttrib = acAttrib;
    pInfo->pName   = acName;
    pInfo->pExt    = acExt;
    pInfo->SizeL   = Context.nFileSizeLow;
    pInfo->SizeH   = Context.nFileSizeHigh;
    pInfo->Flags   = (Context.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
                     ? CHOOSEFILE_FLAG_DIRECTORY : 0;
  }
  return 0;
}
```


CHOOSEFILE_EnableToolTips()

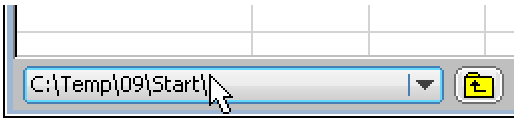
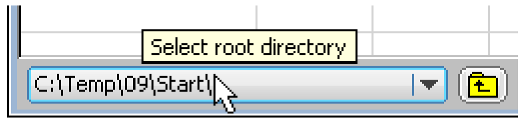
Before	After
	

Table 20.43: CHOOSEFILE_EnableToolTips() before after screenshots

Description

Enables ToolTips for CHOOSEFILE dialogs.

Prototype

```
void CHOOSEFILE_EnableToolTips(void);
```

Additional information

The text of the ToolTips can be configured. Details can be found in the description of “CHOOSEFILE_SetToolTips()” on page 856.

CHOOSEFILE_SetButtonText()

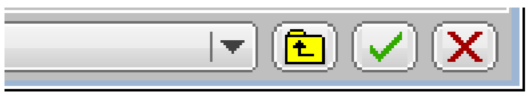
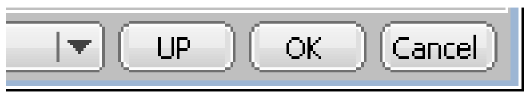
Before	After
	

Table 20.44: CHOOSEFILE_SetButtonText() before after screenshots

Description

Uses text instead of the default image.

Prototype

```
void CHOOSEFILE_SetButtonText(WM_HWIN hWin, unsigned ButtonIndex,
                              const char * pText);
```

Parameter	Description
`hWin`	Handle of the CHOOSEFILE dialog.
`ButtonIndex`	See table below.
`pText`	Pointer to a string to be used.

Table 20.45: CHOOSEFILE_SetButtonText() parameter list

Permitted values for parameter `Index`	
`CHOOSEFILE_BI_CANCEL`	Index of ’cancel’ button.
`CHOOSEFILE_BI_OK`	Index of ’Ok’ button.
`CHOOSEFILE_BI_UP`	Index of ’Up’ button.

Additional information

The function copies the string(s) into its own memory location(s). The size of the buttons depend on the used text. The dialog makes sure, that all buttons which use text instead of an image have the same size.

CHOOSEFILE_SetDefaultButtonText()

Description

Sets the default text to be used for new dialogs.

Prototype

```
void CHOOSEFILE_SetDefaultButtonText(unsigned ButtonIndex,
                                     const char * pText);
```

Parameter	Description
ButtonIndex	See table below.
pText	Text to be used per default.

Table 20.46: CHOOSEFILE_SetDefaultButtonText() parameter list

Permitted values for parameter Index	
CHOOSEFILE_BI_CANCEL	Index of 'cancel' button.
CHOOSEFILE_BI_OK	Index of 'Ok' button.
CHOOSEFILE_BI_UP	Index of 'Up' button.

CHOOSEFILE_SetDelim()

Description

Sets the delimiter used within a path. Default is a backslash.

Prototype

```
void CHOOSEFILE_SetDelim(char Delim);
```

Parameter	Description
Delim	Delimiter to be used.

Table 20.47: CHOOSEFILE_SetDelim() parameter list

CHOOSEFILE_SetToolTips()

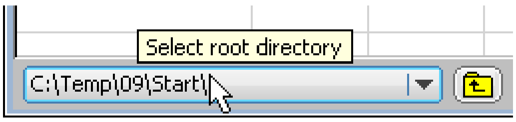
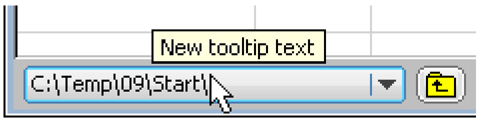
Before	After
	

Table 20.48: CHOOSEFILE_SetToolTips() before after screenshots

Description

Sets the text to be shown by the ToolTips.

Prototype

```
void CHOOSEFILE_SetToolTips(const TOOLTIP_INFO * pInfo, int NumItems);
```

Parameter	Description
pInfo	Pointer to an array of TOOLTIP_INFO structures.
NumItems	Number of items pointed by pInfo.

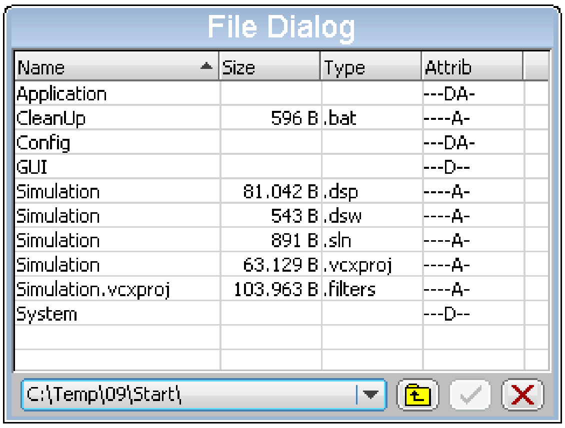
Table 20.49: CHOOSEFILE_SetToolTips() parameter list

Additional information

The elements of the TOOLTIP_INFO structure are described in the section "ToolTips" on page 386.

CHOOSEFILE_SetTopMode()

Before



After

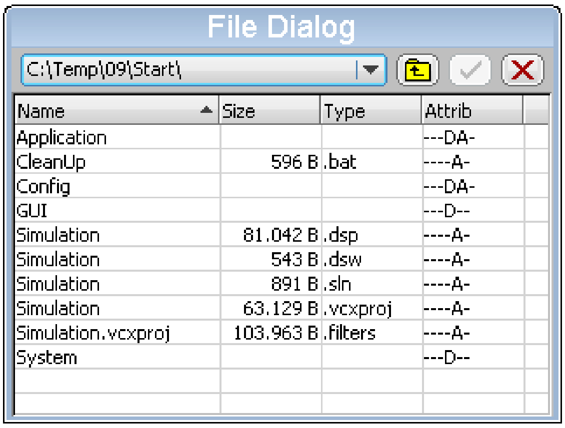


Table 20.50: CHOOSEFILE_SetTopMode() before after screenshots

Description

Makes the button bar visible at the top of the dialog.

Prototype

```
void CHOOSEFILE_SetTopMode(unsigned OnOff);
```

Parameter	Description
OnOff	1 for top mode, 0 (default) for bottom mode.

Table 20.51: CHOOSEFILE_SetTopMode() parameter list

20.4.4 MESSAGEBOX

A MESSAGEBOX is used to show a message in a frame window with a title bar, as well as an "OK" button which must be pressed in order to close the window. It requires only one line of code to create or to create and execute a message box. All MESSAGEBOX-related routines are in the file(s) MESSAGEBOX*.c, MESSAGEBOX.h and GUI.h.

Simple message box

20.4.4.1 Configuration options

Type	Macro	Default	Description
N	MESSAGEBOX_BORDER	4	Distance between the elements of a message box and the elements of the client window frame.
N	MESSAGEBOX_XSIZEOK	50	X-size of the "OK" button.
N	MESSAGEBOX_YSIZEOK	20	Y-size of the "OK" button.
S	MESSAGEBOX_BKCOLOR	GUI_WHITE	Color of the client window background.

Table 20.52: Configuration options

20.4.4.2 Keyboard reaction

The widget consists of a FRAMEWIN, a TEXT and a BUTTON widget. When executing a message box the BUTTON widget gains the input focus. Detailed information on how keyboard events are handled by the BUTTON widget can be found in the section “BUTTON: Button widget” on page 462.

20.4.4.3 MESSAGEBOX API

The table below lists the available emWin MESSAGEBOX-related routines in alphabetical order. Detailed descriptions follow.

Routine	Description
GUI_MessageBox()	Creates and displays a message box.
MESSAGEBOX_Create()	Creates a message box.

Table 20.53: MESSAGEBOX API list

GUI_MessageBox()

Description

Creates and displays a message box.

Prototype

```
int GUI_MessageBox(const char * sMessage, const char * sCaption, int Flags);
```

Parameter	Description
sMessage	Message to display.
sCaption	Caption for the title bar of the frame window.
Flags	See table below.

Table 20.54: GUI_MessageBox() parameter list

Permitted values for parameter Flags	
GUI_MESSAGEBOX_CF_MOVEABLE	The message box can be moved by dragging the title bar or the frame.
0	No function.

Additional information

This function offers the possibility to create and execute a MESSAGEBOX with one line of code. An example implementation can be found in the sample application DIALOG_MessageBox.c which is located in the Sample folder. Details about dragging can be found in the description of “FRAMEWIN_SetMoveable()” on page 552.

MESSAGEBOX_Create()

Description

Creates a message box.

Prototype

```
WM_HWIN GUI_MessageBox(const char * sMessage, const char * sCaption,
                       int Flags);
```

Parameter	Description
sMessage	Message to display.
sCaption	Caption for the title bar of the frame window.
Flags	See table below.

Table 20.55: MESSAGEBOX_Create() parameter list

Permitted values for parameter Flags	
GUI_MESSAGEBOX_CF_MODAL	Creates a modal message box. The default is creating a non modal message box.

Return value

Handle of the message box window.

Additional information

The function creates a message box consisting of a frame window with the caption text in the title bar, a text widget with the message text and a button widget representing the ’OK’ button. After creating the message box the dialog behavior could be changed by using a user defined callback function or the properties of the box items can be modified using the widget API functions. The following IDs can be used for accessing the items:

Id	Description
GUI_ID_TEXT0	Id of the TEXT widget containing the message text.
GUI_ID_OK	Id of the ’OK’ BUTTON widget.

Table 20.56: MESSAGEBOX widget IDs

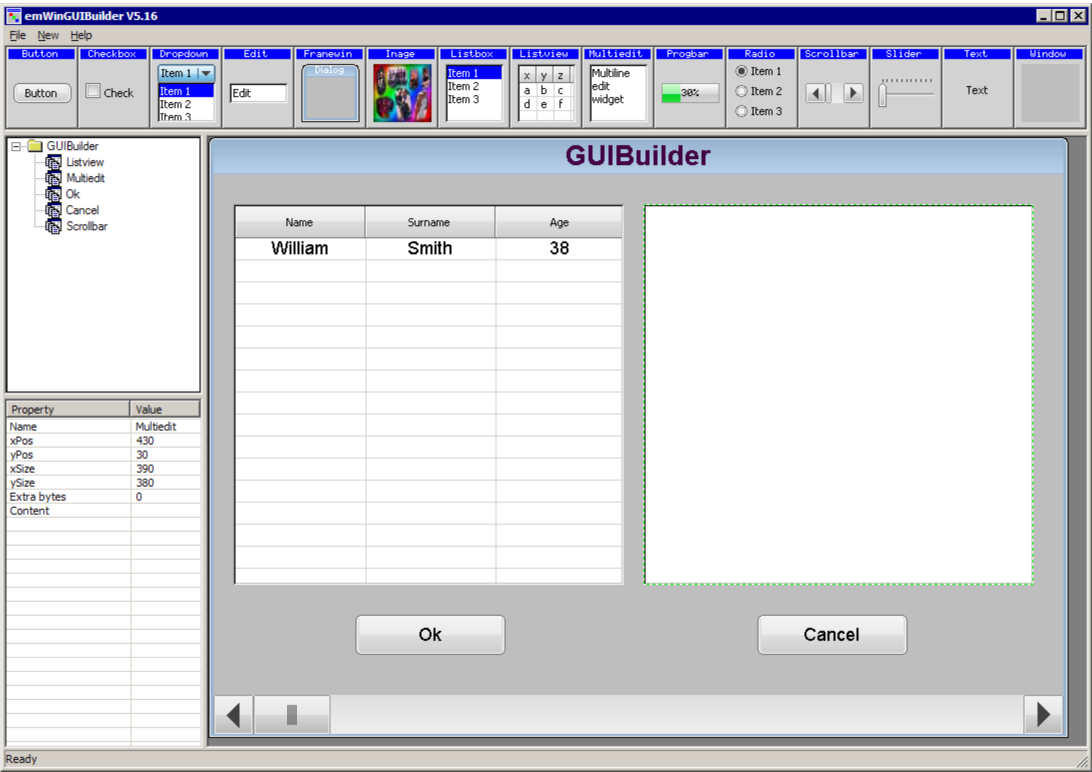
The frame window can be accessed by the handle returned by this function.
The function GUI_ExecCreatedDialog() should be used to execute the message box.

Chapter 21

GUIBuilder

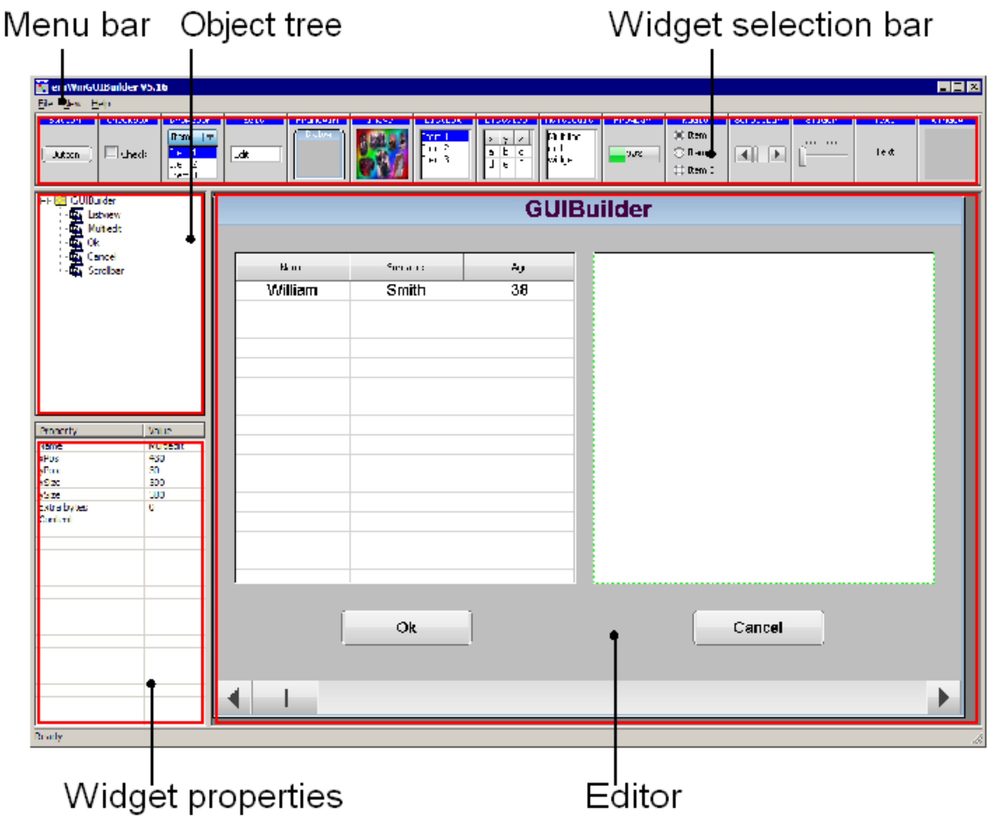
The GUIBuilder application is a tool for creating dialogs without any knowledge of the C programming language. Instead of writing source code the widgets can be placed and sized by drag and drop. Additional properties can be added per context menu. Fine tuning can be done by editing the properties of the widgets. This does not require any knowledge of the C programming language. The dialogs can be saved as C files which can be enhanced by adding user defined code. Of course these C files with the embedded user code can be loaded and modified by the GUIBuilder.

Screenshot



21.1 Introduction

The following diagram shows the elements of the graphical user interface of the GUI-Builder:



Widget selection bar

This bar contains all available widgets of the GUIBuilder. They can be added by a single click into the selection bar on the desired widget or by dragging them into the editor area.

Object tree

This area shows all currently loaded dialogs and their child widgets. It can be used for selecting a widget by clicking on the according entry.

Widget properties

It shows the properties of each widget and can be used for editing them.

Editor

The editor window shows the currently selected dialog. It can be used to place and resize the dialog and its widgets.

21.2 Getting started

Before starting a project, the GUIBuilder needs to know the project path. Per default this is the application path of the GUIBuilder. All files are saved in this folder.

Setting up the project path

After the first execution, the GUIBuilder directory contains the configuration file `GUIBuilder.ini`. Within this file the project path can be changed by editing the value `ProjectPath`:

```
[Settings]
ProjectPath="C:\Work\MyProject\"
```


21.3 Creating a dialog

The following shows how to create a dialog and how to modify the properties of the used widgets.

21.3.1 Selecting a parent widget

Each dialog requires a valid parent widget. So it is required to start with a widget which is able to serve as a parent. Currently there are 2 widgets which can be used at this point:

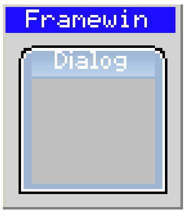
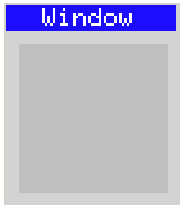
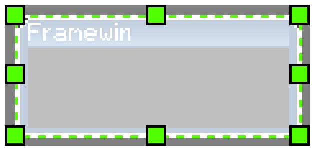
Frame window widget	Window widget
	

Table 21.1: Possible parent widgets

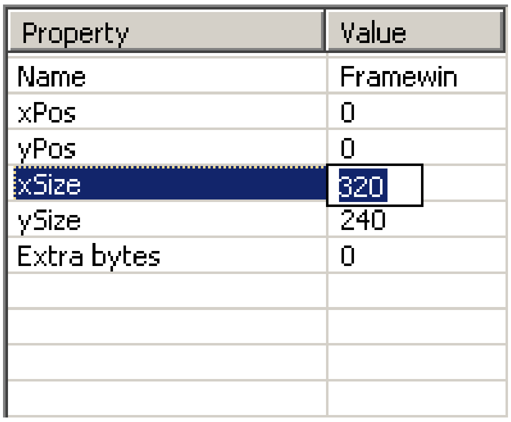
The table above shows the according buttons of the widget selection bar. To get a widget into the editor the buttons can be single clicked, dragged with the mouse into the editor window or created by using the 'New' menu.

21.3.2 Resizing and positioning in the editor



After placing a widget into the editor area it can be moved by using the mouse or the arrow keys of the keyboard. Resizing can be done by dragging the markers.

21.3.3 Modifying the widget properties



Property	Value
Name	Framewin
xPos	0
yPos	0
xSize	320
ySize	240
Extra bytes	0

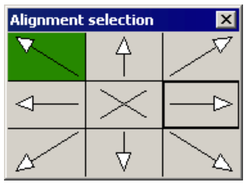
The lower left area of the GUIBuilder contains the property window. After creating a new widget it shows the default properties of the widget: Name, position, size and extra bytes. These properties are available for all kinds of widgets and can not be removed. Contrary to the default properties all additional properties can be removed by the context menu or by pressing <DEL> when the according line is selected. To change a value it can be selected by the keyboard by pressing <ENTER> (if the desired line is selected and the window has the focus) or by single clicking into the value field. Further the 'Edit' entry of the context menu available with a right click can be used to start the edit operation. <ESC> can be used to abort the edit operation.

21.3.4 Adding additional functions to a widget

To get a context menu with the available functions for a widget either a right click in the editor window on the desired widget or a right click in the object tree can be done. Selecting a function adds a new property to the widget and starts the edit operation for the chosen function. In case of numerical or alpha numerical values the edit operation is done within the property window.
In case of choosing fonts, text alignments or colors a separate selection window occurs.

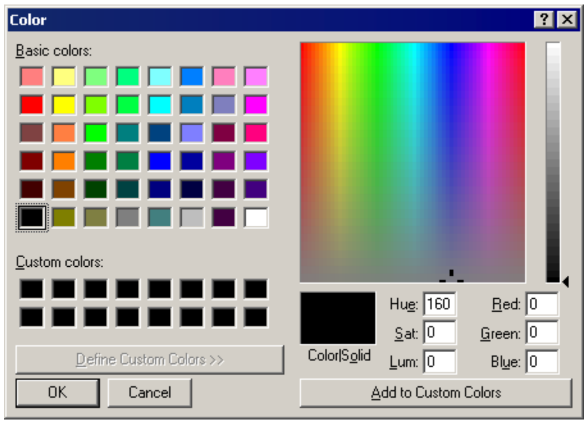
Alignment selection

The alignment selection dialog shows the previous selected alignment in green. A single click within the box selects a new alignment. <ESC> aborts the selection.



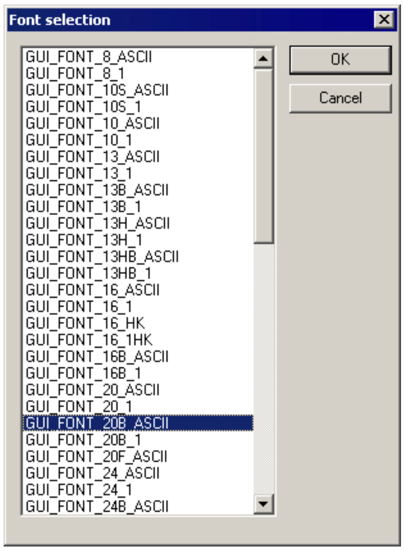
Color selection

For selecting a color the Windows default color selection dialog occurs. <ESC> aborts the selection.



Font selection

The font selection dialog shows all available fonts of the GUIBuilder. The desired font can be selected by a single click on the desired font. <ESC> aborts the selection.



21.3.5 Deleting a widget property

This can be done easily by using the context menu of the property window or by pressing the <DEL> key if the desired property in the widget property window has the focus.

21.3.6 Deleting a widget

A widget can be deleted by pressing the <DEL> key if the widget is activated in the editor window. It can also be removed by selecting it in the object tree window and then pressing the <DEL> key. Deleting a widget automatically causes all of its child windows to be deleted as well.

21.4 Saving the current dialog(s)

With the menu entry 'File/Save...' all currently loaded dialogs will be saved in the project folder. Details on how to set up the project folder can be found in the section "Getting started" on page 863.
Each dialog will be saved as a single C file. The file names are generated automatically by the name of the dialog root widget (FRAMEWIN or WINDOW widget). The file names are build as follows:

<Widget name>DLG.c

If for example the name of the widget is 'Framewin' the file will be named FramewinDLG.c.

21.5 Output of the GUIBuilder

As mentioned above the result of the GUIBuilder are C files only. The following shows a small sample which is generated by the tool:

```
/*********************************************************************
*                                                                    *
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
*                                                                    *
**********************************************************************
*                                                                    *
* C-file generated by:                                               *
*                                                                    *
*        GUI_Builder for emWin version 5.09                          *
*        Compiled Mar 23 2011, 09:52:04                              *
*        (c) 2011 Segger Microcontroller GmbH & Co. KG               *
*                                                                    *
**********************************************************************
*                                                                    *
*        Internet: www.segger.com  Support: support@segger.com       *
*                                                                    *
**********************************************************************
*/

// USER START (Optionally insert additional includes)
// USER END

#include "DIALOG.h"

/*********************************************************************
*
*       Defines
*
**********************************************************************
*/

#define ID_FRAMEWIN_0   (GUI_ID_USER + 0x0A)
#define ID_BUTTON_0     (GUI_ID_USER + 0x0B)

// USER START (Optionally insert additional defines)
// USER END

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

// USER START (Optionally insert additional static data)
// USER END

/*********************************************************************
*
*       _aDialogCreate
*/
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Framewin", ID_FRAMEWIN_0, 0, 0, 320, 240, 0, 0, 0 },
  { BUTTON_CreateIndirect, "Button", ID_BUTTON_0, 5, 5, 80, 20, 0, 0, 0 },
  // USER START (Optionally insert additional widgets)
  // USER END
};

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

// USER START (Optionally insert additional static code)
// USER END

/*********************************************************************
*
*       _cbDialog
*/
static void _cbDialog(WM_MESSAGE * pMsg) {
  WM_HWIN hItem;
  int Id, NCode;
```



```
  // USER START (Optionally insert additional variables)
  // USER END

  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    //
    // Initialization of 'Framewin'
    //
    hItem = pMsg->hWin;
    FRAMEWIN_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
    FRAMEWIN_SetFont(hItem, GUI_FONT_24_ASCII);
    //
    // Initialization of 'Button'
    //
    hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_0);
    BUTTON_SetText(hItem, "Press me...");
    // USER START (Opt. insert additional code for further widget initialization)
    // USER END
    break;
  case WM_NOTIFY_PARENT:
    Id    = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;
    switch(Id) {
    case ID_BUTTON_0: // Notifications sent by 'Button'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      case WM_NOTIFICATION_RELEASED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      // USER START (Opt. insert additional code for further notification handling)
      // USER END
      }
      break;
    // USER START (Optionally insert additional code for further IDs)
    // USER END
    }
    break;
  // USER START (Optionally insert additional message handling)
  // USER END
  default:
    WM_DefaultProc(pMsg);
    break;
  }
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/
/*********************************************************************
*
*       CreateFramewin
*/
WM_HWIN CreateFramewin(void) {
  WM_HWIN hWin;

  hWin = GUI_CreateDialogBox(_aDialogCreate,
                             GUI_COUNTOF(_aDialogCreate), _cbDialog, WM_HBKWIN, 0, 0);
  return hWin;
}

// USER START (Optionally insert additional public code)
// USER END

/*************************** End of file ****************************/
```


21.6 Modifying the C files

As the sample code shows, it contains many sections for custom code. These are the following sections:

```
// USER START (Optionally insert ...)
// USER END
```

The start and end lines may not be modified. They are required for the GUIBuilder to be able to distinguish between user code and generated code. The following shows how it should work:

```
// USER START (Optionally insert additional includes)
#ifndef WIN32
  #include <ioat91sam9261.h>
#endif
// USER END
```

21.7 How to use the C files

As the sample output shows, the code does not contain any code which uses the dialogs or with other words makes them visible on the display. Each file contains a creation routine at the end named `Create<Widget name>()`. These routines create the according dialog. Simply call these routines to make them occur on the display.

Example

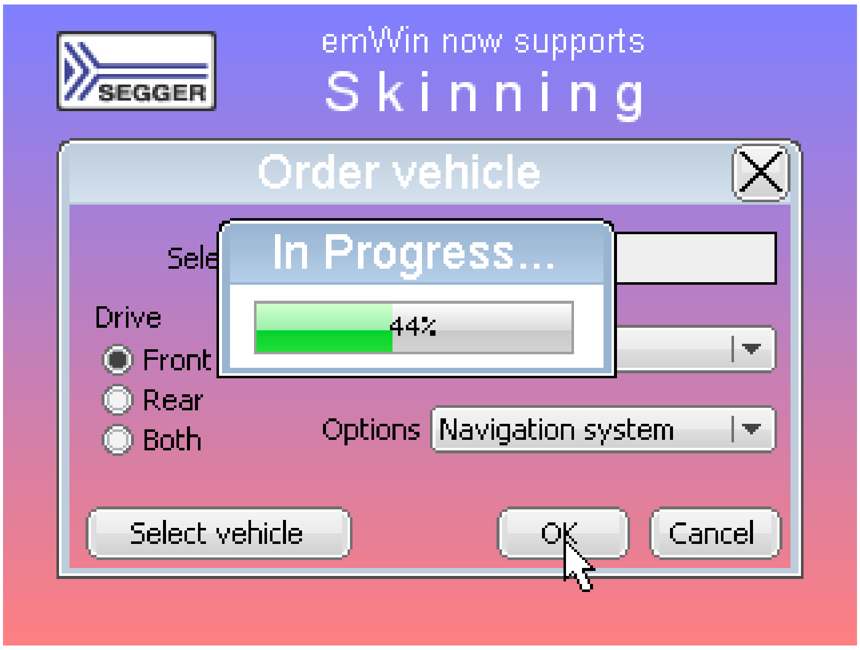
The following code shows how to draw the dialog of the previous output sample on a display:

```
#include "DIALOG.h"

/*********************************************************************
*
*       Externals
*
**********************************************************************
*/
WM_HWIN CreateFramewin(void);

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/
/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  WM_HWIN hDlg;
  GUI_Init();
  //
  // Call creation function for the dialog
  //
  hDlg = CreateFramewin();
  //
  // May do anything with hDlg
  //
  ...
  //
  // Keep program allive...
  //
  while (1) {
    GUI_Delay(10);
  }
}
```


Chapter 22

Skinning

Skinning is a method of changing the appearance of one or multiple widgets. It allows changing the look by using a dedicated skin which defines how the widgets are rendered. This makes it easy to change the appearance of a complete group of widgets in a similar way by changing only the skin.
Without skinning, widget member functions have to be used to change the look for each single widget or the callback function has to be overwritten.



22.1 What is a ’skin’?

A skin is just a simple callback function which is available for drawing all details of a widget. It works by exactly same way as a ’user draw function’ of a widget, an older method of widget customization which was available before ’skinning’ was implemented.

22.2 From using API functions to skinning

There are different methods to change the appearance of a widget. Widget API functions, user draw functions, skinning and overwriting the callback function can be used to modify the appearance of a widget. The decision of the method to be used depends on what should be changed. The following explains what can be achieved with each method.

Using widget API functions

The default API functions can be used to change attributes like size, color, font or bitmaps used to draw a widget using the classical design. The following screenshot shows a typical sample of what can be done:

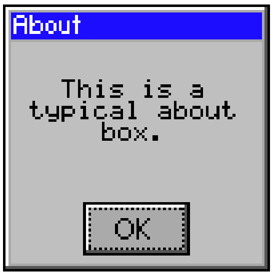
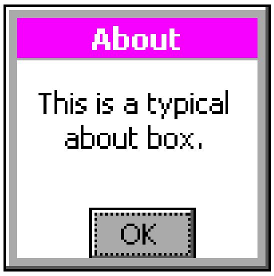
Before	After
	

Table 22.1: Widget API functions before after screenshots

Some attributes can be changed but the basic appearance stays the same.

User draw functions

Some widgets like LISTBOX, FRAMEWIN, GRAPH or BUTTON widgets offer user draw functions. These functions can be used to draw some additional details or to replace the default drawing method for some items. The following screenshot shows a user drawn title area of a frame window. The user draw function renders the gradient in the title area, which can’t be achieved with the widget API functions:

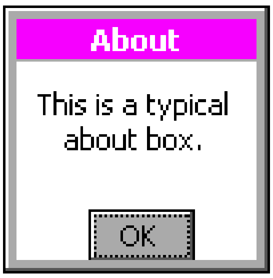
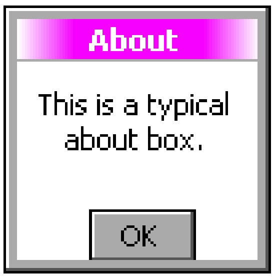
Before	After
	

Table 22.2: User draw functions before after screenshots

Skinning

Contrary to the methods mentioned above skinning covers the drawing of the whole widget and not only some details. We also used this opportunity to lift the appearance of the skinnable widgets which look much more up-to-date as the classical widget design. The following table shows the look of the about box from above in comparison with the new default skin:

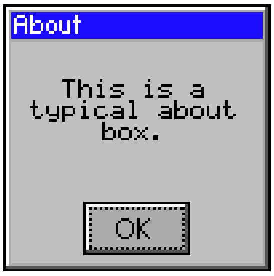
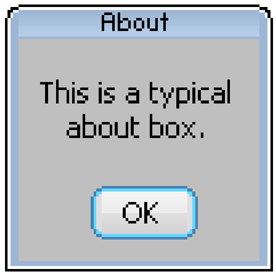
<table>
<tr><th>Classical design</th><th>Default skin</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 22.3: Comparison classical and skinned design

Overwriting the callback function of a widget

Before skinning was implemented, the only method of changing the complete appearance of a widget was overwriting the callback function of a widget. This gives full control over the complete message processing of the widget. It can be used in combination with the other methods. The main disadvantages of overwriting the callback function is that lots of code needs to be written by the user. This process is error-prone.

22.3 Skinnable widgets

Skinning only makes sense if a widget consists of several widget specific details. It does not make sense for each kind of widget. A TEXT widget for example does not require a separate skin, because it consists only of the text itself.
Currently the following widgets support skinning:

- BUTTON
- CHECKBOX
- DROPDOWN
- FRAMEWIN
- HEADER
- MENU
- MULTIPAGE
- PROGBAR
- RADIO
- SCROLLBAR
- SLIDER
- SPINBOX

22.4 Using a skin

The shipment of emWin contains a ready-to-use default skin for all above listed skinnable widgets. They have been named `<WIDGET>_SKIN_FLEX`.
The following table shows the available default skins for all skinnable widgets:

Widget	Default skin
BUTTON	BUTTON_SKIN_FLEX
CHECKBOX	CHECKBOX_SKIN_FLEX
DROPDOWN	DROPDOWN_SKIN_FLEX
FRAMEWIN	FRAMEWIN_SKIN_FLEX
HEADER	HEADER_SKIN_FLEX
MENU	MENU_SKIN_FLEX
MULTIPAGE	MULTIPAGE_SKIN_FLEX
PROGBAR	PROGBAR_SKIN_FLEX
RADIO	RADIO_SKIN_FLEX
SCROLLBAR	SCROLLBAR_SKIN_FLEX
SLIDER	SLIDER_SKIN_FLEX
SPINBOX	SPINBOX_SKIN_FLEX

Table 22.4: Available default skins

22.4.1 Runtime configuration

To use these skins the function `<WIDGET>_SetSkin(<WIDGET>_SKIN_FLEX)` can be used. Further it is possible to set a default skin by `<WIDGET>_SetDefaultSkin()` which is used automatically for each new widget.

Switching from classic design to a skin

The most recommended way of using a skin is first setup the widget behavior and then creating the widget.

Example

The following example shows how a skin can be used:

```
BUTTON_SetSkin(hButton, BUTTON_SKIN_FLEX); // Sets the skin for the given widget
BUTTON_SetDefaultSkin(BUTTON_SKIN_FLEX);   // Sets the default skin for new widgets
```

22.4.2 Compile time configuration

If Skinning should be used by default the compile time configuration macro `WIDGET_USE_FLEX_SKIN` needs to be defined as 1 in GUIConf.h.

Example

To use skinning per default the macro should be added to the file `GUIConf.h`:

```
#define WIDGET_USE_FLEX_SKIN 1
```


22.5 Simple changes to the look of the 'Flex' skin

Similar to the API functions available for changing the attributes of the classical look the attributes of the 'Flex' skin can also be changed. This can be done without knowing all details of the skinning mechanism.
The function(s) <WIDGET>_SetSkinFlexProps() explained in detail later in this chapter can be used to change the attributes. For each skin exist functions for getting and setting the attributes.

Example

The following code shows how to change the attributes of the button skin:

```
BUTTON_GetSkinFlexProps(&Props, BUTTON_SKINFLEX_FOCUSSED);
Props.aColorFrame[0] = 0x007FB13C;
Props.aColorFrame[1] = 0x008FfF8F;
Props.Radius         = 6;
BUTTON_SetSkinFlexProps(&Props, BUTTON_SKINFLEX_FOCUSSED);
WM_InvalidateWindow(hWin);
```

Since skin properties are general to a certain type of widget, setting skin properties does not invalidate any window. Widgets which are affected by any changes have to be invalidated as shown above.

Screenshot

Before	After
	

Table 22.5: Flex skin changes before after screenshots

22.6 Major changes to the look of the 'Flex' skin

The drawing mechanism of the default design without skinning is a 'black box' for the application designer. The same is true for skinning if no major changes of the default look are required. If changing the attributes of the default skin is not sufficient to realize the required look, it is required to understand the details of the drawing mechanism of skinning.

22.6.1 The skinning callback mechanism

The drawing mechanism for all skinnable widgets is very similar and looks as follows:

```
int <WIDGET>_DrawSkin(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW_BACKGROUND:
    /* Draw the background */
    break;
  case WIDGET_ITEM_DRAW_TEXT:
    /* Draw the text */
    break;
  case WIDGET_ITEM_CREATE:
    /* Additional function calls required to create the widget */
    break;
  ...
}
```

Elements of structure WIDGET_ITEM_DRAW_INFO

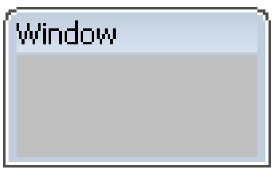
Data type	Element	Description
WM_HWIN	hWin	Handle to the widget.
int	Cmd	Command to be processed.
int	ItemIndex	Index of item to be drawn.
int	x0	Leftmost coordinate in window coordinates.
int	y0	Topmost coordinate in window coordinates.
int	x1	Rightmost coordinate in window coordinates.
int	y1	Bottommost coordinate in window coordinates.
void *	p	Data pointer to widget specific information.

Table 22.6: WIDGET_ITEM_DRAW_INFO element list

This scheme is identical to all skinnable widgets. The callback function receives a pointer to a `WIDGET_ITEM_DRAW_INFO` structure. The structure pointed by `pDrawItemInfo` contains a command which has to be processed, a handle to the widget and further information whose meaning may vary by widget. The skinning callback function has to react with drawing a dedicated detail or with returning a dedicated value.
How to use the drawing information in detail is explained later in this chapter.

22.6.2 Changing the look of the default skin

Understanding the above callback mechanism is important because changing a skin can easily be done by deriving a new skin from an existing one. A small example should show how the look of the default skin of a widget can be changed.
Assuming the default look of the frame window skin should be changed because an icon should be shown on the left side of the title bar. The default appearance of the FRAMEWIN skin is as follows:



This should be changed to the following:

This can be done easily by using a customized skin derived from the default skin. The following code shows how this can be achieved. It shows a custom skinning callback function which is used as skin by the function `FRAMEWIN_SetSkin()`. Because the icon should be drawn in the text area of the frame window the function overwrites the default behaviour of the text drawing:

```
  case WIDGET_ITEM_DRAW_TEXT:
    ...
```

All other tasks should be performed by the default skin:

```
  default:
    return FRAMEWIN_DrawSkinFlex(pDrawItemInfo);
```

Example

```
static int _DrawSkinFlex_FRAME(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  char acBuffer[20];
  GUI_RECT Rect;

  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW_TEXT:
    //
    // Draw icon at the left side
    //
    GUI_DrawBitmap(&_bmLogo_30x15, pDrawItemInfo->x0, pDrawItemInfo->y0);
    //
    // Draw text beneath
    //
    FRAMEWIN_GetText(pDrawItemInfo->hWin, acBuffer, sizeof(acBuffer));
    GUI_SetColor(GUI_BLACK);
    Rect.x0 = pDrawItemInfo->x0   // Default position of text
            + _bmLogo_30x15.XSize // + X-size of icon
            + 4;                  // + small gap between icon and text
    Rect.y0 = pDrawItemInfo->y0;
    Rect.x1 = pDrawItemInfo->x1;
    Rect.y1 = pDrawItemInfo->y1;
    GUI_DispStringInRect(acBuffer, &Rect, GUI_TA_VCENTER);
    break;
  default:
    //
    // Use the default skinning routine for processing all other commands
    //
    return FRAMEWIN_DrawSkinFlex(pDrawItemInfo);
  }
  return 0;
}

void _SetSkin(WM_HWIN) {
  //
  // Set the derived
  //
  FRAMEWIN_SetSkin(hFrame, _DrawSkinFlex_FRAME);
}
```


22.6.3 List of commands

The skinning routine receives a pointer to a `WIDGET_ITEM_DRAW_INFO` structure. The `Cmd` member of this structure contains the command which needs to be processed. There are several commands which are sent to the skinning routine of a widget, but only a set of commands is sent to a certain type of widget. Further the exact meaning may vary according to the widget. How to react to commands is explained in the widget specific sections of this chapter. The following table gives an overview of the commands which are sent to the skinning routines:

Command Id (`Cmd`)	**Description**
Creation messages	
WIDGET_ITEM_CREATE	Sent to each skinnable widget after it has been created but before it is drawn.
Information messages	
WIDGET_ITEM_GET_BORDERSIZE_B	Used to get the size of the bottom border.
WIDGET_ITEM_GET_BORDERSIZE_L	Used to get the size of the left border.
WIDGET_ITEM_GET_BORDERSIZE_R	Used to get the size of the right border.
WIDGET_ITEM_GET_BORDERSIZE_T	Used to get the size of the top border.
WIDGET_ITEM_GET_BUTTONSIZE	Used to get the button size.
WIDGET_ITEM_GET_XSIZE	Used to get the X-size.
WIDGET_ITEM_GET_YSIZE	Used to get the Y-size.
Drawing messages	
WIDGET_ITEM_DRAW_ARROW	Used to draw an arrow.
WIDGET_ITEM_DRAW_BACKGROUND	Used to draw the background.
WIDGET_ITEM_DRAW_BITMAP	Used to draw a bitmap.
WIDGET_ITEM_DRAW_BUTTON	Used to draw the button area.
WIDGET_ITEM_DRAW_BUTTON_L	Used to draw the left button area.
WIDGET_ITEM_DRAW_BUTTON_R	Used to draw the right button area.
WIDGET_ITEM_DRAW_FOCUS	Used to draw the focus rectangle.
WIDGET_ITEM_DRAW_FRAME	Used to draw the frame of a widget.
WIDGET_ITEM_DRAW_OVERLAP	Used to draw the overlapping region.
WIDGET_ITEM_DRAW_SEP	Used to draw a separator.
WIDGET_ITEM_DRAW_SHAFT	Used to draw the shaft area.
WIDGET_ITEM_DRAW_SHAFT_L	Used to draw the left shaft area.
WIDGET_ITEM_DRAW_SHAFT_R	Used to draw the right shaft area.
WIDGET_ITEM_DRAW_TEXT	Used to draw the text.
WIDGET_ITEM_DRAW_THUMB	Used to draw the thumb area.
WIDGET_ITEM_DRAW_TICKS	Used to draw tick marks.

Table 22.7: List of commands

22.7 General Skinning API

The table below lists available skinning-related routines in alphabetical order. These functions are common to all skinnable widgets, and are listed here in order to avoid repetition. Detailed descriptions of the routines follow. The additional skinning member functions available for each widget may be found in later sections.

Routine	Description
`<WIDGET>_DrawSkinFlex()`	Skinning callback function of the default skin.
`<WIDGET>_GetSkinFlexProps()`	Returns the current properties of the skin.
`<WIDGET>_SetDefaultSkin()`	Sets the default skin used for new widgets.
`<WIDGET>_SetDefaultSkinClassic()`	Sets the classical design as default for new widgets.
`<WIDGET>_SetSkin()`	Sets a skin for the given widget.
`<WIDGET>_SetSkinClassic()`	Sets the classical design for the given widget.
`<WIDGET>_SetSkinFlexProps()`	Sets the properties of the skin.

Table 22.8: Skinning API list

<WIDGET>_DrawSkinFlex()

Description

These functions are the skinning callback functions of the default skin and are responsible to draw the complete widget.

Prototype

```
int <WIDGET>_DrawSkinFlex(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Description
`pDrawItemInfo`	Pointer to a data structure of type `WIDGET_ITEM_DRAW_INFO`.

Table 22.9: <WIDGET>_DrawSkinFlex() parameter list

Additional information

A derived skin can use this function for drawing details of the default skin.

<WIDGET>_GetSkinFlexProps()

Description

These functions return the attributes of the default skin. The widget specific descriptions later in this chapter explain the skin attributes in detail.

Prototype

```
void <WIDGET>_GetSkinFlexProps(<WIDGET>_SKINFLEX_PROPS * pProps, int Index);
```

Parameter	Description
`pProps`	Pointer to a skin specific configuration structure of type `<WIDGET>_SKINFLEX_PROPS` to be filled by the function.
`Index`	Widget state (pressed, active, selected, ...) for which the details should be retrieved.

Table 22.10: <WIDGET>_GetSkinFlexProps() parameter list

<WIDGET>_SetDefaultSkin()

Description

These functions set the default skin which is used for new widgets of the dedicated type.

Prototype

```
void <WIDGET>_SetDefaultSkin(WIDGET_DRAW_ITEM_FUNC * pfDrawSkin);
```

Parameter	Description
pfDrawSkin	Pointer to a skinning callback function of type WIDGET_DRAW_ITEM_FUNC.

Table 22.11: <WIDGET>_SetDefaultSkin() parameter list

Additional information

The given pointer should point to the skinning callback routine to be used for all new widgets. Details can be found in the description of "<WIDGET>_SetSkin()" on page 880.

<WIDGET>_SetDefaultSkinClassic()

Description

These functions set the classical design for all new widgets of the dedicated type.

Prototype

```
void <WIDGET>_SetDefaultSkinClassic(void);
```

Additional information

The behaviour of widgets which use the classical design is completely identical to the behaviour before implementing the skinning feature.

<WIDGET>_SetSkin()

Description

These functions can be used for setting a skin for the given widget.

Prototype

```
void <WIDGET>_SetSkin(<WIDGET>_Handle          hObj,
                      WIDGET_DRAW_ITEM_FUNC * pfDrawSkin);
```

Parameter	Description
hObj	Handle to the dedicated widget.
pfDrawSkin	Pointer to a skinning callback function of type WIDGET_DRAW_ITEM_FUNC.

Table 22.12: <WIDGET>_SetSkin() parameter list

WIDGET_DRAW_ITEM_FUNC

```
typedef int WIDGET_DRAW_ITEM_FUNC(const WIDGET_ITEM_DRAW_INFO *
                                                     pDrawItemInfo);
```

Additional information

Default widget API functions may have no effect in case a skin is used.

<WIDGET>_SetSkinClassic()

Description

These functions switch to the classical design without skinning for the given widget.

Prototype

```
void <WIDGET>_SetSkinClassic(<WIDGET>_Handle hObj);
```

Parameter	Description
hObj	Handle to the dedicated widget.

Table 22.13: <WIDGET>_SetSkinClassic() parameter list

Additional information

Additional information can be found in the description of “<WIDGET>_SetDefaultSkinClassic()” on page 880.

<WIDGET>_SetSkinFlexProps()

Description

With these functions some attributes of the default skin can be changed without deriving an own skin. The widget specific descriptions later in this chapter will explain in detail what can be changed.

Prototype

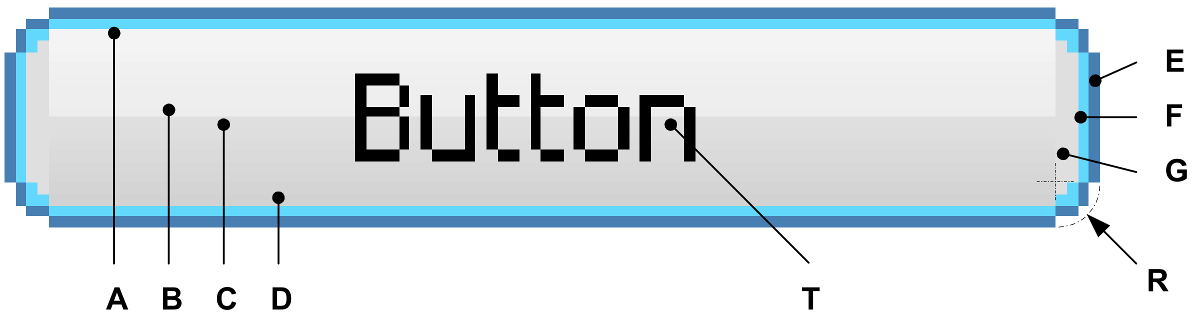
```
void <WIDGET>_SetSkinFlexProps(const <WIDGET>_SKINFLEX_PROPS * pProps,
                               int Index);
```

Parameter	Description
pProps	Pointer to a skin specific configuration structure of type <WIDGET>_SKINFLEX_PROPS.
Index	Details of the state (pressed, active, selected, ...) for which the details should be valid.

Table 22.14: <WIDGET>_SetSkinFlexProps() parameter list

22.8 BUTTON_SKIN_FLEX

The following picture shows the details of the skin:



The BUTTON skin consists of a rounded border and a rectangular inner area which is filled by 2 gradients. The surrounding border is drawn by 2 colors.

Detail	Description
A	Top color of top gradient.
B	Bottom color of top gradient.
C	Top color of bottom gradient.
D	Bottom color of bottom gradient.
E	Outer color of surrounding frame.
F	Inner color of surrounding frame.
G	Color of area between surrounding frame and inner rectangular area.
R	Radius of rounded corner.
T	Optional text.

Table 22.15: Skin details

22.8.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `BUTTON_SKINFLEX_PROPS` are used:

Elements of structure BUTTON_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorFrame[3]	[0] - Outer color of surrounding frame. [1] - Inner color of surrounding frame. [2] - Color of area between frame and inner area.
U32	aColorUpper[2]	[0] - First (upper) color of upper gradient. [1] - Second (lower) color of upper gradient.
U32	aColorLower[2]	[0] - First (upper) color of lower gradient. [1] - Second (lower) color of lower gradient.
int	Radius	Radius of rounded corner.

Table 22.16: BUTTON_SKINFLEX_PROPS element list

22.8.2 Configuration options

The default appearance of the skin can be defined using custom configuration structures of the type `BUTTON_SKINFLEX_PROPS` in `GUIConf.h`. The following table shows the identifiers which are used for the different states of the skinned BUTTON widget:

Macro	Description
BUTTON_SKINPROPS_PRESSED	Defines the default attributes used for pressed state.

Table 22.17: Configuration options

Macro	Description
BUTTON_SKINPROPS_FOCUSSED	Defines the default attributes used for focussed state.
BUTTON_SKINPROPS_ENABLED	Defines the default attributes used for enabled state.
BUTTON_SKINPROPS_DISABLED	Defines the default attributes used for disabled state.

Table 22.17: Configuration options

22.8.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
`BUTTON_DrawSkinFlex()`	Skinning callback function of BUTTON_SKIN_FLEX. (Explained at the beginning of the chapter)
`BUTTON_GetSkinFlexProps()`	Returns the properties of the given button skin. (Explained at the beginning of the chapter)
`BUTTON_SetDefaultSkin()`	Sets the default skin used for new button widgets. (Explained at the beginning of the chapter)
`BUTTON_SetDefaultSkinClassic()`	Sets the classical design as default for new button widgets. (Explained at the beginning of the chapter)
`BUTTON_SetSkin()`	Sets a skin for the given button widget. (Explained at the beginning of the chapter)
`BUTTON_SetSkinClassic()`	Sets the classical design for the given button widget. (Explained at the beginning of the chapter)
`BUTTON_SetSkinFlexProps()`	Sets the properties of the given button skin.

Table 22.18: BUTTON Skinning API list

BUTTON_SetSkinFlexProps()

Before	After
	

Table 22.19: BUTTON_SetSkinFlexProps()

Description

The function can be used to change the properties of the skin.

Prototype

```
void BUTTON_SetSkinFlexProps(const BUTTON_SKINFLEX_PROPS * pProps,
                             int                           Index);
```

Parameter	Description
`pProps`	Pointer to a structure of type BUTTON_SKINFLEX_PROPS.
`Index`	See table below.

Table 22.20: BUTTON_SetSkinFlexProps() parameter list

Permitted values for parameter `Index`	
BUTTON_SKINFLEX_PI_PRESSED	Properties for pressed state.
BUTTON_SKINFLEX_PI_FOCUSSED	Properties for focussed state.
BUTTON_SKINFLEX_PI_ENABLED	Properties for enabled state.
BUTTON_SKINFLEX_PI_DISABLED	Properties for disabled state.

Additional information

The function passes a pointer to a `BUTTON_SKINFLEX_PROPS` structure. It can be used to set up the colors and the radius of the skin.

The function BUTTON_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.8.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the BUTTON_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the background.
WIDGET_ITEM_DRAW_BITMAP	The skinning function should draw the optional button bitmap.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the optional button text.

Table 22.21: List of commands

The WIDGET_ITEM_DRAW_INFO structure is explained at the beginning of the chapter.

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BACKGROUND

The background of the widget should be drawn.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	See table below.
x0	Leftmost coordinate in window coordinates, normally 0.
y0	Topmost coordinate in window coordinates, normally 0.
x1	Rightmost coordinate in window coordinates.
y1	Bottommost coordinate in window coordinates.

Table 22.22: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element ItemIndex	
BUTTON_SKINFLEX_PI_PRESSED	The widget is pressed.
BUTTON_SKINFLEX_PI_FOCUSSED	The widget is not pressed but focussed.
BUTTON_SKINFLEX_PI_ENABLED	The widget is not focussed but enabled.
BUTTON_SKINFLEX_PI_DISABLED	The widget is disabled.

WIDGET_ITEM_DRAW_BITMAP

The optional button bitmap should be drawn.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 884.

Additional information

The function BUTTON_GetBitmap() can be used to get the optional button bitmap.

WIDGET_ITEM_DRAW_TEXT

The optional button text should be drawn.

WIDGET_ITEM_DRAW_INFO

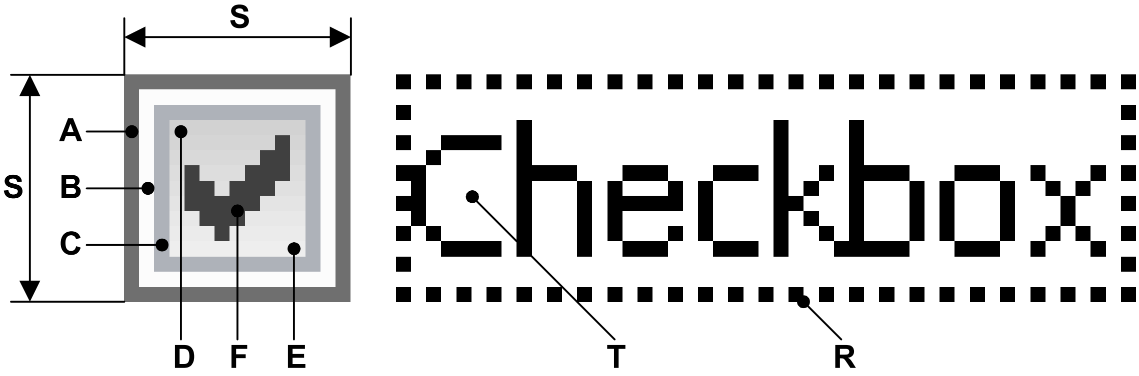
A detailed description of the elements can be found under "WIDGET_ITEM_DRAW_BACKGROUND" on page 884.

Additional information

The function `BUTTON_GetText()` can be used to get the optional text.

22.9 CHECKBOX_SKIN_FLEX

The following picture shows the details of the skin:



The button area of the CHECKBOX skin consists of a frame and a rectangular inner area which is filled by a gradient. The frame is drawn by 3 colors. If it is checked, a checkmark is shown in the center of the box:

Detail	Description
A	First color of frame.
B	Second color of frame.
C	Third color of frame.
D	Upper color of gradient.
E	Lower color of gradient.
F	Color of checkmark.
R	Focus rectangle.
S	Size in pixels of button area.
T	Optional text.

Table 22.23: Skin details

22.9.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `CHECKBOX_SKINFLEX_PROPS` are used:

Elements of structure CHECKBOX_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorFrame[3]	[0] - Outer color of frame. [1] - Middle color of frame. [2] - Inner color of frame.
U32	aColorInner[2]	[0] - First (upper) color of gradient. [1] - Second (lower) color of gradient.
U32	ColorCheck	Color of checkmark.
int	ButtonSize	Size in pixels of the button area. (Obsolete. Use the functions CHECKBOX_GetSkinFlexButtonSize() and CHECKBOX_SetSkinFlexButtonSize().)

Table 22.24: CHECKBOX_SKINFLEX_PROPS element list

22.9.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
CHECKBOX_SKINPROPS_ENABLED	Defines the default attributes used for enabled state.
CHECKBOX_SKINPROPS_DISABLED	Defines the default attributes used for disabled state.

Table 22.25: Configuration options

22.9.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
CHECKBOX_DrawSkinFlex()	Skinning callback function of CHECKBOX_SKIN_FLEX. (Explained at the beginning of the chapter)
CHECKBOX_GetSkinFlexButtonSize()	Returns the button size of the given CHECKBOX widget.
CHECKBOX_GetSkinFlexProps()	Returns the properties of the given checkbox skin. (Explained at the beginning of the chapter)
CHECKBOX_SetDefaultSkin()	Sets the default skin used for new checkbox widgets. (Explained at the beginning of the chapter)
CHECKBOX_SetDefaultSkinClassic()	Sets the classical design as default for new checkbox widgets. (Explained at the beginning of the chapter)
CHECKBOX_SetSkin()	Sets a skin for the given checkbox widget. (Explained at the beginning of the chapter)
CHECKBOX_SetSkinClassic()	Sets the classical design for the given checkbox widget. (Explained at the beginning of the chapter)
CHECKBOX_SetSkinFlexButtonSize()	Sets button size of the given CHECKBOX widget.
CHECKBOX_SetSkinFlexProps()	Sets the properties of the given checkbox skin.

Table 22.26: CHECKBOX Skinning API list

CHECKBOX_GetSkinFlexButtonSize()

Description

Returns the button size of the specified CHECKBOX widget.

Prototype

```
int CHECKBOX_GetSkinFlexButtonSize(CHECKBOX_Handle hObj);
```

Parameter	Description
hObj	Handle to a CHECKBOX widget.

Table 22.27: CHECKBOX_GetSkinFlexButtonSize() parameter list

Return value

Button size.

CHECKBOX_SetSkinFlexButtonSize()

Before	After
	

Table 22.28: CHECKBOX_SetSkinFlexButtonSize() before after screenshots

Description

Sets the button size of the specified CHECKBOX widget.

Prototype

```
void CHECKBOX_SetSkinFlexButtonSize(CHECKBOX_Handle hObj, int ButtonSize);
```

Parameter	Description
hObj	Handle to a CHECKBOX widget.
ButtonSize	Size to be set.

Table 22.29: CHECKBOX_SetSkinFlexButtonSize() parameter list

CHECKBOX_SetSkinFlexProps()

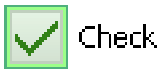
Before	After
	

Table 22.30: CHECKBOX_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the properties of the skin.

Prototype

```
void CHECKBOX_SetSkinFlexProps(const CHECKBOX_SKINFLEX_PROPS * pProps,
                               int                             Index);
```

Parameter	Description
pProps	Pointer to a structure of type CHECKBOX_SKINFLEX_PROPS.
Index	See table below.

Table 22.31: CHECKBOX_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
CHECKBOX_SKINFLEX_PI_ENABLED	Properties for enabled state.
CHECKBOX_SKINFLEX_PI_DISABLED	Properties for disabled state.

Additional information

The function passes a pointer to a `CHECKBOX_SKINFLEX_PROPS` structure. It can be used to set up the colors of the skin.
Please note that the size of the widgets using the skin won't be changed if for example the new button size is different to the old button size. This can not be done by the skin, because it does not 'know' which widget is using it. If required resizing should be done by the application, for example with `WM_ResizeWindow()`.
The function `CHECKBOX_GetSkinFlexProps()` can be used to get the current attributes of the skin.

22.9.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the CHECKBOX_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BUTTON	The background of the button area should be drawn.
WIDGET_ITEM_DRAW_BITMAP	The checkmark of the button area should be drawn.
WIDGET_ITEM_DRAW_FOCUS	The focus rectangle should be drawn.
WIDGET_ITEM_DRAW_TEXT	The optional text should be drawn.

Table 22.32: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BUTTON

The button area of the widget without checkmark should be drawn. It is typically drawn at the left side of the widget area.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
x0	Leftmost coordinate of widget area in window coordinates, normally 0.
y0	Topmost coordinate of widget area in window coordinates, normally 0.
x1	Rightmost coordinate of widget area in window coordinates.
y1	Bottommost coordinate of widget area in window coordinates.

Table 22.33: WIDGET_ITEM_DRAW_INFO element list

The content of hWin, x0, y0, x1 and y1 is the same for all commands of this skin.

WIDGET_ITEM_DRAW_BITMAP

The checkmark should be drawn in the center of the button area.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
ItemIndex	1 - The widget is checked. 2 - Second checked state when using a 3 state button.
hWin,x0,y0,x1,y1	These elements are described under “WIDGET_ITEM_DRAW_BUTTON” on page 889.

Table 22.34: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_FOCUS

The focus rectangle should be drawn around the text.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
p	The void pointer points to the zero terminated optional text of the widget.
hWin,x0,y0,x1,y1	These elements are described under “WIDGET_ITEM_DRAW_BUTTON” on page 889.

Table 22.35: WIDGET_ITEM_DRAW_INFO element list

Additional information

The element p can be casted to a text pointer. Details can be found in the description of “WIDGET_ITEM_DRAW_TEXT” on page 890.

WIDGET_ITEM_DRAW_TEXT

The optional text should be drawn. The text is typically drawn at the right side of the button area.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
p	The void pointer points to the zero terminated optional text of the widget.
hWin,x0,y0,x1,y1	These elements are described under “WIDGET_ITEM_DRAW_BUTTON” on page 889.

Table 22.36: WIDGET_ITEM_DRAW_INFO element list

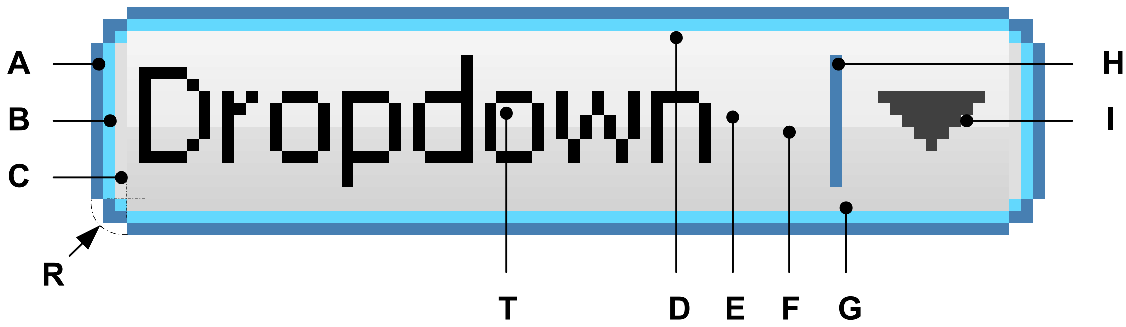
Additional information

To get a text pointer the element p can be casted to a text pointer:

```
char * s;
s = (char *)pDrawItemInfo->p;
GUI_DispString(s);
```


22.10 DROPDOWN_SKIN_FLEX

The following picture shows the details of the skin:



The DROPDOWN skin consists of a rounded frame and a rectangular inner area which is filled by two gradients. The rounded frame is drawn by 3 colors. At the right side a small triangle is drawn. Between text and triangle a small separator is drawn:

Detail	Description
A	First color of frame.
B	Second color of frame.
C	Third color of frame.
D	Top color of top gradient.
E	Bottom color of top gradient.
F	Top color of bottom gradient.
G	Bottom color of bottom gradient.
H	Separator between text and triangle.
I	Triangle.
R	Radius of rounded corner.
T	Optional text.

Table 22.37: Skin details

The dropdown widget in open state consists of an additional LISTBOX widget. The skin does not affect the LISTBOX.

22.10.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `DROPDOWN_SKINFLEX_PROPS` are used:

Elements of structure DROPDOWN_SKINFLEX_PROPS

Data type	Element	Description
`U32`	`aColorFrame[3]`	[0] - Outer color of surrounding frame. [1] - Inner color of surrounding frame. [2] - Color of area between frame and inner area.
`U32`	`aColorUpper[2]`	[0] - Top color of top gradient. [1] - Bottom color of top gradient.
`U32`	`aColorLower[2]`	[0] - Top color of bottom gradient. [1] - Bottom color of bottom gradient.
`U32`	`ColorArrow`	Color used to draw the arrow.
`U32`	`ColorText`	Color used to draw the text.
`U32`	`ColorSep`	Color used to draw the separator.
`int`	`Radius`	Radius of rounded corner.

Table 22.38: DROPDOWN_SKINFLEX_PROPS element list

22.10.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
DROPDOWN_SKINPROPS_OPEN	Defines the default attributes used for open state.
DROPDOWN_SKINPROPS_FOCUSSED	Defines the default attributes used for focussed state.
DROPDOWN_SKINPROPS_ENABLED	Defines the default attributes used for enabled state.
DROPDOWN_SKINPROPS_DISABLED	Defines the default attributes used for disabled state.

Table 22.39: Configuration options

22.10.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
DROPDOWN_DrawSkinFlex()	Skinning callback function of DROPDOWN_SKIN_FLEX. (Explained at the beginning of the chapter)
DROPDOWN_GetSkinFlexProps()	Returns the properties of the given dropdown skin. (Explained at the beginning of the chapter)
DROPDOWN_SetDefaultSkin()	Sets the default skin used for new dropdown widgets. (Explained at the beginning of the chapter)
DROPDOWN_SetDefaultSkinClassic()	Sets the classical design as default for new dropdown widgets. (Explained at the beginning of the chapter)
DROPDOWN_SetSkin()	Sets a skin for the given dropdown widget. (Explained at the beginning of the chapter)
DROPDOWN_SetSkinClassic()	Sets the classical design for the given dropdown widget. (Explained at the beginning of the chapter)
DROPDOWN_SetSkinFlexProps()	Sets the properties of the given dropdown skin.

Table 22.40: DROPDOWN Skinning API list

DROPDOWN_SetSkinFlexProps()

Before	After
	

Table 22.41: DROPDOWN_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the properties of the skin.

Prototype

```
void DROPDOWN_SetSkinFlexProps(const DROPDOWN_SKINFLEX_PROPS * pProps,
                               int                              Index);
```

Parameter	Description
pProps	Pointer to a structure of type DROPDOWN_SKINFLEX_PROPS.
Index	See table below.

Table 22.42: DROPDOWN_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
DROPDOWN_SKINFLEX_PI_OPEN	Properties for open state.
DROPDOWN_SKINFLEX_PI_FOCUSSED	Properties for focussed state.
DROPDOWN_SKINFLEX_PI_ENABLED	Properties for enabled state.
DROPDOWN_SKINFLEX_PI_DISABLED	Properties for disabled state.

Additional information

The function passes a pointer to a DROPDOWN_SKINFLEX_PROPS structure. It can be used to set up the colors and the radius of the skin.
The function DROPDOWN_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.10.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the DROPDOWN_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_ARROW	The skinning function should draw the arrow.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the background.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the optional button text.

Table 22.43: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_ARROW

The triangle (arrow) at the right side should be drawn.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 893.

WIDGET_ITEM_DRAW_BACKGROUND

The background of the widget should be drawn.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	See table below.
x0	Leftmost coordinate in window coordinates, normally 0.
y0	Topmost coordinate in window coordinates, normally 0.
x1	Rightmost coordinate in window coordinates.
y1	Bottommost coordinate in window coordinates.

Table 22.44: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element ItemIndex	
DROPDOWN_SKINFLEX_PI_EXPANDED	The widget is expanded.
DROPDOWN_SKINFLEX_PI_FOCUSSED	The widget is in not pressed but focussed.
DROPDOWN_SKINFLEX_PI_ENABLED	The widget is in not focussed but enabled.
DROPDOWN_SKINFLEX_PI_DISABLED	The widget is disabled.

WIDGET_ITEM_DRAW_TEXT

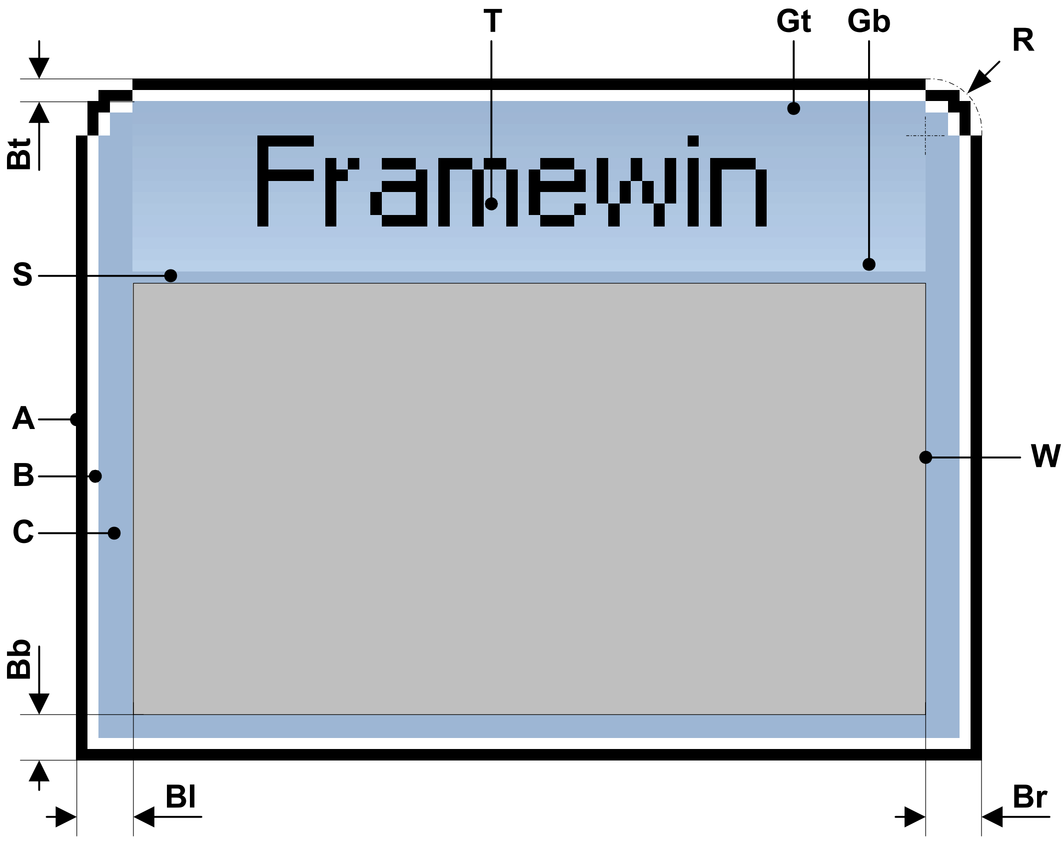
The text of the currently selected string should be drawn within the button area of the dropdown widget. The text is typically drawn at the left side of the button area.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 893.

22.11 FRAMEWIN_SKIN_FLEX

The following picture shows the details of the skin:



The FRAMEWIN skin consists of a title bar, rounded corners at the top, a gradient used to draw the background of the title bar, a border whose size is configurable and a separator between title bar and client area:

Detail	Description
A	Outer color of surrounding frame.
B	Inner color of surrounding frame.
C	Color of area between frame and inner area.
Gt	Top color of top title bar gradient.
Gb	Bottom color of title bar gradient.
Bt	Top size of border.
Bb	Bottom size of border.
Bl	Left size of border.
Br	Right size of border.
W	Area of client window.
R	Radius of rounded corner.
T	Optional text.

Table 22.45: Skin details

22.11.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `FRAMEWIN_SKINFLEX_PROPS` are used:

Elements of structure FRAMEWIN_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorFrame[3]	[0] - Outer color of surrounding frame. [1] - Inner color of surrounding frame. [2] - Color of area between frame and inner area.
U32	aColorTitle[2]	[0] - Top color of top title bar gradient. [1] - Bottom color of title bar gradient.
int	Radius	Radius of rounded corners.
int	SpaceX	Optional space in X between title text and border of title gradient.
int	BorderSizeL	Left size of border.
int	BorderSizeR	Right size of border.
int	BorderSizeT	Top size of border.
int	BorderSizeB	Bottom size of border.

Table 22.46: FRAMEWIN_SKINFLEX_PROPS element list

22.11.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in GUIConf.h. The following table shows the available configuration options:

Macro	Description
FRAMEWIN_SKINPROPS_ACTIVE	Defines the default attributes used for active state.
FRAMEWIN_SKINPROPS_INACTIVE	Defines the default attributes used for inactive state.

Table 22.47: Configuration options

22.11.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
FRAMEWIN_DrawSkinFlex()	Skinning callback function of FRAMEWIN_SKIN_FLEX. (Explained at the beginning of the chapter)
FRAMEWIN_GetSkinFlexProps()	Returns the properties of the given FRAMEWIN skin. (Explained at the beginning of the chapter)
FRAMEWIN_SetDefaultSkin()	Sets the default skin used for new framewin widgets. (Explained at the beginning of the chapter)
FRAMEWIN_SetDefaultSkinClassic()	Sets the classical design as default for new framewin widgets. (Explained at the beginning of the chapter)
FRAMEWIN_SetSkin()	Sets a skin for the given framewin widget. (Explained at the beginning of the chapter)
FRAMEWIN_SetSkinClassic()	Sets the classical design for the given framewin widget. (Explained at the beginning of the chapter)
FRAMEWIN_SetSkinFlexProps()	Sets the properties of the given framewin skin.

Table 22.48: FRAMEWIN Skinning API list

FRAMEWIN_SetSkinFlexProps()

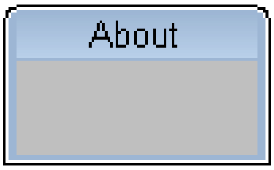
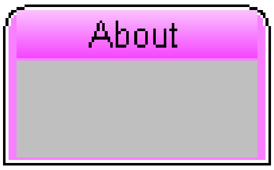
Before	After
	

Table 22.49: FRAMEWIN_SetSkinFlexProps()

Description

The function can be used to change the properties of the skin.

Prototype

```
void FRAMEWIN_SetSkinFlexProps(const FRAMEWIN_SKINFLEX_PROPS * pProps,
                               int                             Index);
```

Parameter	Description
pProps	Pointer to a structure of type FRAMEWIN_SKINFLEX_PROPS.
Index	See table below.

Table 22.50: FRAMEWIN_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
FRAMEWIN_SKINFLEX_PI_ACTIVE	Properties for active state.
FRAMEWIN_SKINFLEX_PI_INACTIVE	Properties for inactive state.

Additional information

The function passes a pointer to a FRAMEWIN_SKINFLEX_PROPS structure. It can be used to set up the colors, radius and border size of the skin.
The function FRAMEWIN_GetSkinFlexProps() can be used to get the current attributes of the skin.
When creating a frame window using this skin the values for inactive state are used for calculating size and position of the client window.

22.11.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the FRAMEWIN_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the title background.
WIDGET_ITEM_DRAW_FRAME	The skinning function should draw the frame.
WIDGET_ITEM_DRAW_SEP	The skinning function should draw the separator.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the title text.
WIDGET_ITEM_GET_BORDERSIZE_L	The skinning function should return the left border size.
WIDGET_ITEM_GET_BORDERSIZE_R	The skinning function should return the right border size.
WIDGET_ITEM_GET_BORDERSIZE_T	The skinning function should return the top border size.
WIDGET_ITEM_GET_BORDERSIZE_B	The skinning function should return the bottom border size.
WIDGET_ITEM_GET_RADIUS	The skinning function should return the radius of the corners.

Table 22.51: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BACKGROUND

The skinning routine should draw the background of the title area.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	See table below.
x0	Leftmost coordinate of title area in window coordinates.
y0	Topmost coordinate of title area in window coordinates.
x1	Rightmost coordinate of title area in window coordinates.
y1	Bottommost coordinate of title area in window coordinates.

Table 22.52: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element ItemIndex	
FRAMEWIN_SKINFLEX_PI_ACTIVE	The widget is in active state.
FRAMEWIN_SKINFLEX_PI_INACTIVE	The widget is in inactive state.

WIDGET_ITEM_DRAW_FRAME

The skinning routine should draw the complete border without the title area and the separator.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	See table below.
x0	Leftmost coordinate in window coordinates, normally 0.
y0	Topmost coordinate in window coordinates, normally 0.
x1	Rightmost coordinate in window coordinates (xSize of window - 1).
y1	Bottommost coordinate in window coordinates (ySize of window - 1).

Table 22.53: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element ItemIndex	
FRAMEWIN_SKINFLEX_PI_ACTIVE	The widget is in active state.
FRAMEWIN_SKINFLEX_PI_INACTIVE	The widget is in inactive state.

WIDGET_ITEM_DRAW_SEP

The skinning routine should draw the separator between title area and client window.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	See table below.
x0	Leftmost coordinate of separator in window coordinates.
y0	Topmost coordinate of separator in window coordinates.
x1	Rightmost coordinate of separator in window coordinates.
y1	Bottommost coordinate of separator in window coordinates.

Table 22.54: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element ItemIndex	
FRAMEWIN_SKINFLEX_PI_ACTIVE	The widget is in active state.
FRAMEWIN_SKINFLEX_PI_INACTIVE	The widget is in inactive state.

WIDGET_ITEM_DRAW_TEXT

The skinning routine should draw title text.

WIDGET_ITEM_DRAW_INFO

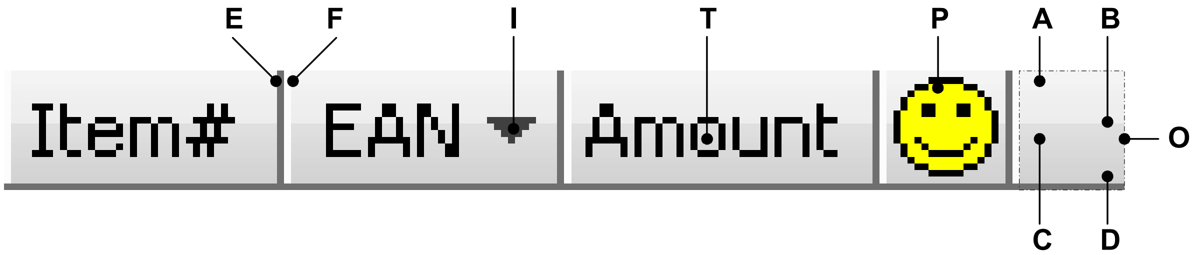
A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 898.

WIDGET_ITEM_GET_BORDERSIZE_L, WIDGET_ITEM_GET_BORDERSIZE_R, WIDGET_ITEM_GET_BORDERSIZE_T, WIDGET_ITEM_GET_BORDERSIZE_B

The skinning routine should return the size of the according border.

22.12 HEADER_SKIN_FLEX

The following picture shows the details of the skin:



The HEADER skin consists of a bar with a thin border which is divided into separate items. The background of the bar consists of a top and a bottom gradient. Each item can have a text, a bitmap and an indicator which can be used for example to show the sorting order:

Detail	Description
A	Top color of top gradient.
B	Bottom color of top gradient.
C	Top color of bottom gradient.
D	Bottom color of bottom gradient.
E	First color of frame.
F	Second color of frame.
I	Indicator.
T	Text (optional).
P	Bitmap (optional).

Table 22.55: Skin details

22.12.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `HEADER_SKINFLEX_PROPS` are used:

Elements of structure HEADER_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorFrame[2]	[0] - First color of frame and separators. [1] - Second color of frame and separators.
U32	aColorUpper[2]	[0] - Top color of top gradient. [1] - Bottom color of top gradient.
U32	aColorLower[2]	[0] - Top color of bottom gradient. [1] - Bottom color of bottom gradient.
U32	ColorArrow	Color of indicator.

Table 22.56: HEADER_SKINFLEX_PROPS element list

22.12.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
HEADER_SKINPROPS	Defines the default attributes used for drawing the skin.

Table 22.57: Configuration options

22.12.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
HEADER_DrawSkinFlex()	Skinning callback function of HEADER_SKIN_FLEX. (Explained at the beginning of the chapter)
HEADER_GetSkinFlexProps()	Returns the properties of the given HEADER skin. (Explained at the beginning of the chapter)
HEADER_SetDefaultSkin()	Sets the default skin used for new HEADER widgets. (Explained at the beginning of the chapter)
HEADER_SetDefaultSkinClassic()	Sets the classical design as default for new HEADER widgets. (Explained at the beginning of the chapter)
HEADER_SetSkin()	Sets a skin for the given HEADER widget. (Explained at the beginning of the chapter)
HEADER_SetSkinClassic()	Sets the classical design for the given HEADER widget. (Explained at the beginning of the chapter)
HEADER_SetSkinFlexProps()	Sets the properties of the given HEADER skin.

Table 22.58: HEADER Skinning API list

HEADER_SetSkinFlexProps()

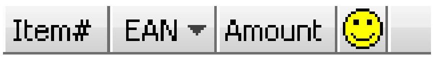
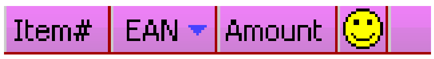
Before	After
	

Table 22.59: HEADER_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the properties of the skin.

Prototype

```
void HEADER_SetSkinFlexProps(const HEADER_SKINFLEX_PROPS * pProps,
                             int                            Index);
```

Parameter	Description
pProps	Pointer to a structure of type HEADER_SKINFLEX_PROPS.
Index	Should be 0.

Table 22.60: HEADER_SetSkinFlexProps() parameter list

Additional information

The function passes a pointer to a `HEADER_SKINFLEX_PROPS` structure. It can be used to set up the colors of the skin.
The function `HEADER_GetSkinFlexProps()` can be used to get the current attributes of the skin.

22.12.4 List of commands

The skinning routine receives a pointer to a `WIDGET_ITEM_DRAW_INFO` structure. The `Cmd` member of this structure contains the command which needs to be processed. The following table shows all commands passed to the `HEADER_SKIN_FLEX` callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_ARROW	The indicator arrow of the header item should be drawn.
WIDGET_ITEM_DRAW_BACKGROUND	The background of the header item should be drawn.
WIDGET_ITEM_DRAW_BITMAP	The bitmap of the header item should be drawn.
WIDGET_ITEM_DRAW_OVERLAP	The overlapping region of the widget should be drawn.
WIDGET_ITEM_DRAW_TEXT	The test of the header item should be drawn.

Table 22.61: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_ARROW

The skinning routine should draw the optional direction indicator. This message is sent only if the indicator of the header item is enabled.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 902.

WIDGET_ITEM_DRAW_BACKGROUND

The skinning routine should draw the background of an item area.

Elements of structure WIDGET_ITEM_DRAW_INFO structure:

Element	Description
hWin	Handle to the widget.
ItemIndex	Is always 0.
x0	Leftmost coordinate of item area in window coordinates.
y0	Topmost coordinate of item area in window coordinates.
x1	Rightmost coordinate of item area in window coordinates.
y1	Bottommost coordinate of item area in window coordinates.

Table 22.62: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_BITMAP

The skinning routine should draw the optional item bitmap. The message is only sent in case of an existing bitmap.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 902.

WIDGET_ITEM_DRAW_OVERLAP

The skinning routine should draw the overlapping region.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 902.

WIDGET_ITEM_DRAW_TEXT

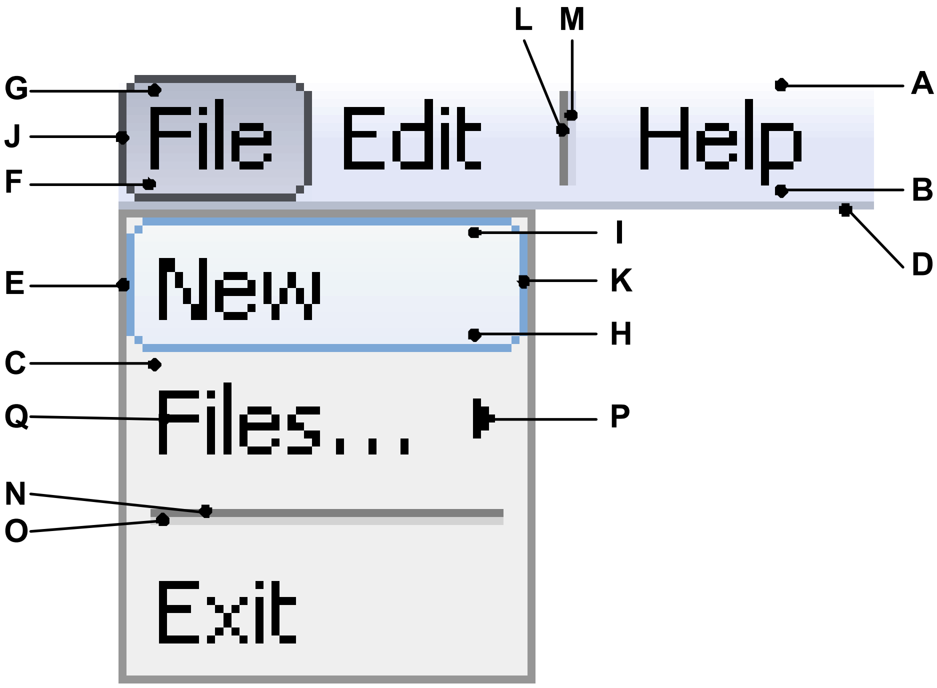
The skinning routine should draw the optional item text. The message is only sent in case of an existing text.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under "WIDGET_ITEM_DRAW_BACKGROUND" on page 902.

22.13 MENU_SKIN_FLEX

The following picture shows the details of the skin:



The MENU skin covers horizontal as well as vertical MENU widgets. Since both variations require the ability to be handled differently, most items can be colored separately. The background is drawn by a gradient for horizontal MENU widgets. The vertical version only consists of a single background color. Selected and selected items having a submenu are drawn using a gradient an a frame. All vertical items are surrounded by a frame. Horizontal items are just underlined. Vertical items with a submenu consist of an arrow indicating the submenu. Separators are drawn using 2 lines with 2 different colors. The item text is displayed using the same color for all items.

Detail	Description
A	Top color of the background gradient. (horizontal)
B	Bottom color of the background gradient. (horizontal)
C	Background color. (vertical)
D	Frame color. (horizontal)
E	Frame color. (vertical)
F	Bottom color of the background gradient for selected items. (horizontal)
G	Top color of the background gradient for selected items. (horizontal)
H	Bottom color of the background gradient for selected items. (vertical)
I	Top color of the background gradient for selected items. (vertical)
J	Frame color for selected items. (horizontal)
K	Frame color for selected items. (vertical)
L	Left separator color. (horizontal)
M	Right separator color. (horizontal)
N	Top separator color. (vertical)
O	Bottom separator color. (vertical)
P	Color of the arrow indicating submenus.
Q	Color of the text.

Table 22.63: Skin details

22.13.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration a structure of the type MENU_SKINFLEX_PROPS must be used:

Elements of structure MENU_SKINFLEX_PROPS

Data type	Element	Description
Background		
U32	aBkColorH[2]	Horizontal: [0] - Top color of background gradient. [1] - Bottom color of horizontal background gradient.
U32	BkColorV	Background color. (vertical)
U32	FrameColorH	Frame color. (horizontal)
U32	FrameColorV	Frame color. (vertical)
Selection		
U32	aSelColorH[2]	Horizontal: [0] - Top color of the background gradient for selected items. [1] - Bottom color of the background gradient for selected items.
U32	aSelColorV[2]	Vertical: [0] - Top color of the background gradient for selected items. [1] - Bottom color of the background gradient for selected items.
U32	FrameColorSelH	Frame color for selected items. (horizontal)
U32	FrameColorSelV	Frame color for selected items. (vertical)
Separator		
U32	aSepColorH[2]	Horizontal: [0] - Left separator color. [1] - Right separator color.
U32	aSepColorV[2]	Vertical: [0] - Top separator color. [1] - Bottom separator color.
General		
U32	ArrowColor	Color of the arrow indicating submenus.
U32	TextColor	Color of the text.

Table 22.64: MENU_SKINFLEX_PROPS element list

22.13.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in GUIConf.h. The following table shows the available configuration options:

Macro	Description
MENU_SKINPROPS_ACTIVE_SUBMENU	Defines the default attributes which are used to draw the item in case its submenu is active.
MENU_SKINPROPS_DISABLED	Defines the default attributes which are used to draw the item in disabled state.
MENU_SKINPROPS_DISABLED_SEL	Defines the default attributes which are used to draw the item in case it is selected in disabled state.
MENU_SKINPROPS_ENABLED	Defines the default attributes which are used to draw the item in enabled state.
MENU_SKINPROPS_SELECTED	Defines the default attributes which are used to draw the item in selected state.

Table 22.65: Configuration options

22.13.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
MENU_DrawSkinFlex()	Skinning callback function of MENU_SKIN_FLEX. (Explained at the beginning of the chapter)
MENU_GetSkinFlexProps()	Returns the properties of the given MENU skin. (Explained at the beginning of the chapter)
MENU_SetDefaultSkin()	Sets the default skin used for new MENU widgets. (Explained at the beginning of the chapter)
MENU_SetDefaultSkinClassic()	Sets the classical design as default for new MENU widgets. (Explained at the beginning of the chapter)
MENU_SetSkin()	Sets a skin for the given MENU widget. (Explained at the beginning of the chapter)
MENU_SetSkinClassic()	Sets the classical design for the given MENU widget. (Explained at the beginning of the chapter)
MENU_SetSkinFlexProps()	Sets the properties of the given MENU skin.
MENU_SkinEnableArrow()	Enable or disable drawing of an arrow.

Table 22.66: MENU Skinning API list

MENU_SetSkinFlexProps()

Before	After
	

Table 22.67: MENU_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the colors of the skin.

Prototype

```
void MENU_SetSkinFlexProps(const MENU_SKINFLEX_PROPS * pProps,
                                 int                   Index);
```

Parameter	Description
pProps	Pointer to a structure of type PROGBAR_SKINFLEX_PROPS.
Index	See table below.

Table 22.68: MENU_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
MENU_SKINFLEX_PI_ENABLED	Properties for enabled state.
MENU_SKINFLEX_PI_SELECTED	Properties for selected state.
MENU_SKINFLEX_PI_DISABLED	Properties for disabled state.
MENU_SKINFLEX_PI_DISABLED_SEL	Properties for disabled selected state.
MENU_SKINFLEX_PI_ACTIVE_SUBMENU	Properties for active submenu state.

Additional information

The function passes a pointer to a MENU_SKINFLEX_PROPS structure. It can be used to set up the colors of the skin. The function MENU_GetSkinFlexProps() can be used to get the current attributes of the skin.

MENU_SkinEnableArrow()

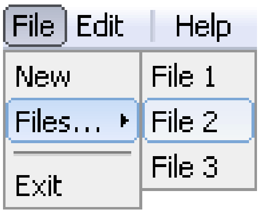
Before	After
	

Table 22.69: MENU_SkinEnableArrow() before after screenshots

Description

Toggles the drawing of an arrow. Arrows are drawn only...

- ...if the item is part of a vertical MENU widget.
- ...if the item consists of a submenu.
- ...if drawing of arrows is enabled using this function.

Prototype

```
void MENU_SkinEnableArrow(MENU_Handle hObj, int OnOff);
```

Parameter	Description
hObj	Handle the MENU widget.
OnOff	1 to enable drawing of arrows. 0 to disable drawing of arrows.

Table 22.70: MENU_SkinEnableArrow() parameter list

22.13.4 List of commands

The skinning routine receives a pointer to a `WIDGET_ITEM_DRAW_INFO` structure. The `Cmd` member of this structure contains the command which needs to be processed. The following table shows all commands passed to the `MENU_SKIN_FLEX` callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_ARROW	The skinning function should draw the arrow.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the background.
WIDGET_ITEM_DRAW_FRAME	The skinning function should draw the frame.
WIDGET_ITEM_DRAW_SEP	The skinning function should draw the separator.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the text.

Table 22.71: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_ARROW

The skinning routine should draw the arrow.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of the according item (>= 0).
x0	Leftmost coordinate of the item area in window coordinates.
y0	Topmost coordinate of the item area in window coordinates.
x1	Rightmost coordinate of the item area in window coordinates.
y1	Bottommost coordinate of the item area in window coordinates.

Table 22.72: WIDGET_ITEM_DRAW_INFO element list

Additional information

This message is sent only in case drawing arrows is enabled. Drawing is enabled by default when using MENU_SKIN_FLEX. Detailed information on how to enable / disable drawing of arrows can be found in the description of the function “MENU_SkinEnableArrow()” on page 907.

WIDGET_ITEM_DRAW_BACKGROUND

The skinning routine should draw the background.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of the according item (>= 0) or -1.
x0	Leftmost coordinate of the item area in window coordinates.
y0	Topmost coordinate of the item area in window coordinates.
x1	Rightmost coordinate of the item area in window coordinates.
y1	Bottommost coordinate of the item area in window coordinates.

Table 22.73: WIDGET_ITEM_DRAW_INFO element list

Additional Information

This message is sent once per each item of the MENU widget. In case a horizontal MENU widget is not completely covered by items, WIDGET_ITEM_DRAW_BACKGROUND is sent one more time with ItemIndex == -1 and the coordinates of the unused area right of the last item.

WIDGET_ITEM_DRAW_FRAME

The skinning routine should draw the surrounding frame.

WIDGET_ITEM_DRAW_INFO

See WIDGET_ITEM_DRAW_ARROW.

WIDGET_ITEM_DRAW_TEXT

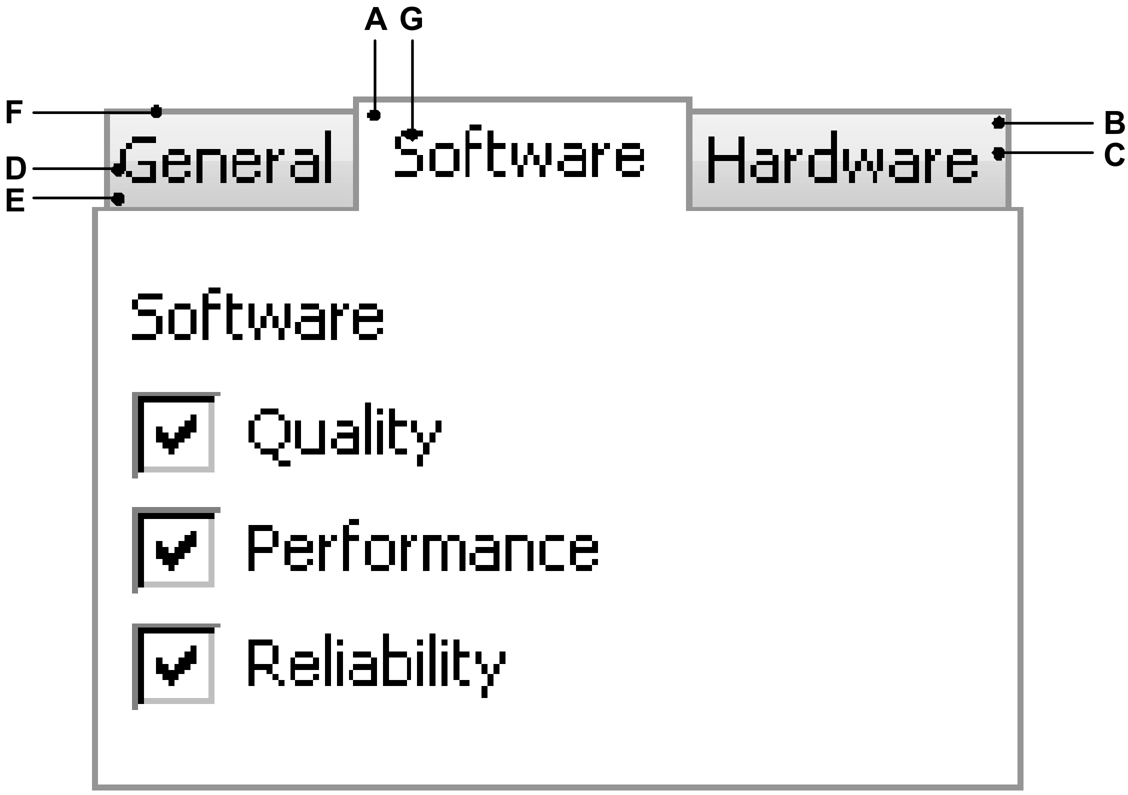
The skinning routine should draw the text.

WIDGET_ITEM_DRAW_INFO

See WIDGET_ITEM_DRAW_ARROW.

22.14 MULTIPAGE_SKIN_FLEX

The following picture shows the details of the skin:



The MULTIPAGE skin consists of the tabs which are drawn using a frame and 2 horizontal gradients as background. The according text is displayed on top of the background:

Detail	Description
A	Background color for selected items.
B	Top color of top gradient.
C	Bottom color of top gradient.
D	Top color of bottom gradient.
E	Bottom color of bottom gradient.
F	Frame color.
G	Text color.

Table 22.74: Skin details

22.14.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `MULTIPAGE_SKINFLEX_PROPS` are used:

Elements of structure MULTIPAGE_SKINFLEX_PROPS

Data type	Element	Description
U32	BkColor	Background color for selected items.
U32	aBkUpper[2]	[0] - Top color of top gradient. [1] - Bottom color of top gradient.
U32	aBkLower[2]	[0] - Top color of bottom gradient. [1] - Bottom color of bottom gradient.
U32	FrameColor	Frame color.
U32	TextColor	Text color.

Table 22.75: MULTIPAGE_SKINFLEX_PROPS element list

22.14.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
MULTIPAGE_SKINPROPS_ENABLED	Defines the default attributes for drawing enabled tabs.
MULTIPAGE_SKINPROPS_SELECTED	Defines the default attributes for drawing selected tabs.
MULTIPAGE_SKINPROPS_DISABLED	Defines the default attributes for drawing disabled tabs.

Table 22.76: Configuration options

22.14.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
MULTIPAGE_DrawSkinFlex()	Skinning callback function of MULTIPAGE_SKIN_FLEX. (Explained at the beginning of the chapter)
MULTIPAGE_GetSkinFlexProps()	Returns the properties of the given MULTIPAGE skin. (Explained at the beginning of the chapter)
MULTIPAGE_SetDefaultSkin()	Sets the default skin used for new MULTIPAGE widgets. (Explained at the beginning of the chapter)
MULTIPAGE_SetDefaultSkinClassic()	Sets the classical design as default for new MULTIPAGE widgets. (Explained at the beginning of the chapter)
MULTIPAGE_SetSkin()	Sets a skin for the given MULTIPAGE widget. (Explained at the beginning of the chapter)
MULTIPAGE_SetSkinClassic()	Sets the classical design for the given MULTIPAGE widget. (Explained at the beginning of the chapter)
MULTIPAGE_SetSkinFlexProps()	Sets the properties of the given MULTIPAGE skin.

Table 22.77: MULTIPAGE Skinning API list

MULTIPAGE_SetSkinFlexProps()

Before	After
	

Table 22.78: MULTIPAGE_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the colors of the skin.

Prototype

```
void MULTIPAGE_SetSkinFlexProps(const MULTIPAGE_SKINFLEX_PROPS * pProps,
                                int                               Index);
```

Parameter	Description
pProps	Pointer to a structure of type MULTIPAGE_SKINFLEX_PROPS.
Index	See table below.

Table 22.79: MULTIPAGE_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
MULTIPAGE_PI_ENABLED	Properties for enabled state.
MULTIPAGE_PI_SELECTED	Properties for selected state.
MULTIPAGE_PI_DISABLED	Properties for disabled state.

Additional information

The function passes a pointer to a MULTIPAGE_SKINFLEX_PROPS structure. It can be used to set up the colors of the skin.
The function MULTIPAGE_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.14.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the MULTIPAGE_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the background.
WIDGET_ITEM_DRAW_FRAME	The skinning function should draw the frame.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the text.

Table 22.80: List of commands

All commands make use of the WIDGET_ITEM_DRAW_INFO structure, which contains the required information to draw the widget using a skin.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of the item to display.
x0	Leftmost coordinate of the client area / current tab in window coordinates.
y0	Topmost coordinate of the client area / current tab in window coordinates.
x1	Rightmost coordinate of the client area / current tab in window coordinates.
y1	Bottommost coordinate of the client area / current tab in window coordinates.
p	Pointer to a MULTIPAGE_SKIN_INFO structure.

Table 22.81: WIDGET_ITEM_DRAW_INFO element list

Elements of structure MULTIPAGE_SKIN_INFO

Data type	Element	Description
GUI_ROTATION *	pRotation	GUI_ROTATE_0, if the MULTIPAGE widget is horizontal. GUI_ROTATE_CW, if the MULTIPAGE widget is vertical.
unsigned	Align	Current alignment of the MULTIPAGE widget. Contains an or-combination of the bit definitions listed below.
int	Sel	Index of the currently selected item. This helps to determine if the currently processed item is selected and therefor might require to be drawn in a different way.
U16	State	Current state of the MULTIPAGE widget. See table below.
U8	FrameFlags	Determines which lines of the frame need to be drawn using an or-combination of the bit definitions listed below.
U8	PageStatus	State of the current page.

Table 22.82: MULTIPAGE_SKIN_INFO element list

Possible bits set in the element Align	
MULTIPAGE_ALIGN_RIGHT	If set, items must be aligned to the right. Otherwise items must be aligned to the left.
MULTIPAGE_ALIGN_BOTTOM	If set, items must be aligned to the bottom. Otherwise items must be aligned to the left.

Possible bits set in the element State	
WIDGET_STATE_VERTICAL	If set, items must be drawn in vertical order. Otherwise items must be drawn in horizontal order.

Possible bits set in the element FrameFlags	
MULTIPAGE_SKIN_FRAME_TOP	If set, the top line of the frame needs to be drawn.
MULTIPAGE_SKIN_FRAME_BOTTOM	If set, the bottom line of the frame needs to be drawn.
MULTIPAGE_SKIN_FRAME_LEFT	If set, the left line of the frame needs to be drawn.
MULTIPAGE_SKIN_FRAME_RIGHT	If set, the right line of the frame needs to be drawn.

Possible bits set in the element PageStatus	
MULTIPAGE_STATE_ENABLED	If set, items must be drawn in vertical order. Otherwise items must be drawn in horizontal order.

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BACKGROUND

The skinning routine should draw the background of the given tab.

WIDGET_ITEM_DRAW_FRAME

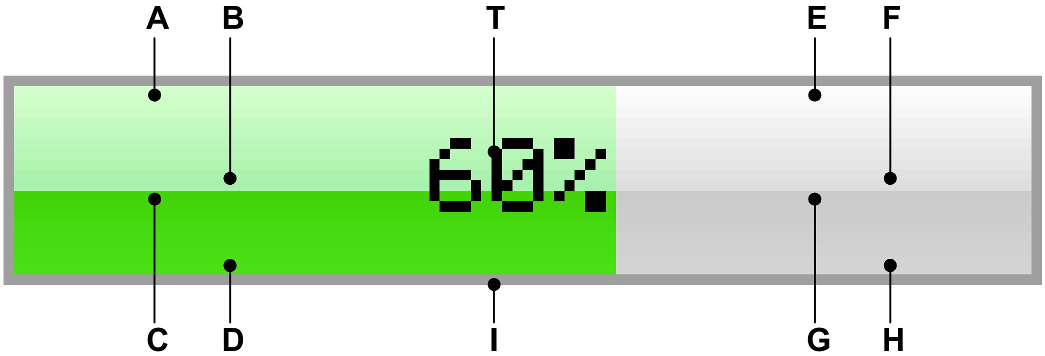
The skinning routine should draw the surrounding frame. In case ItemIndex is given with -1 the frame of the client window has to be drawn. In case of ItemIndex >= 0, a single tab should be drawn. The coordinates in the WIDGET_ITEM_DRAW_INFO structure are set according to the ItemIndex.

WIDGET_ITEM_DRAW_TEXT

The skinning routine should draw the text.

22.15 PROGBAR_SKIN_FLEX

The following picture shows the details of the skin:



The PROGBAR skin consists of a bar with a thin border. The background is drawn by 4 gradients, a top and a bottom gradient at the left and at the right side and a text which shows the current state per default:

Detail	Description
A	Top color of top left gradient.
B	Bottom color of top left gradient.
C	Top color of bottom left gradient.
D	Bottom color of bottom left gradient.
E	Top color of top right gradient.
F	Bottom color of top right gradient.
G	Top color of bottom right gradient.
H	Bottom color of bottom right gradient.
I	Color of frame.
T	Text (optional).

Table 22.83: Skin details

22.15.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `PROGBAR_SKINFLEX_PROPS` are used:

Elements of structure PROGBAR_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorUpperL[2]	[0] - Top color of top gradient. [1] - Bottom color of top gradient.
U32	aColorLowerL[2]	[0] - Top color of bottom gradient. [1] - Bottom color of bottom gradient.
U32	aColorUpperR[2]	[0] - Top color of top gradient. [1] - Bottom color of top gradient.
U32	aColorLowerR[2]	[0] - Top color of bottom gradient. [1] - Bottom color of bottom gradient.
U32	ColorFrame	Color of frame.
U32	ColorText	Color of text.

Table 22.84: PROGBAR_SKINFLEX_PROPS element list

22.15.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in GUIConf.h. The following table shows the available configuration options:

Macro	Description
PROGBAR_SKINPROPS	Defines the default attributes used for drawing the skin.

Table 22.85: Configuration options

22.15.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
PROGBAR_DrawSkinFlex()	Skinning callback function of PROGBAR_SKIN_FLEX. (Explained at the beginning of the chapter)
PROGBAR_GetSkinFlexProps()	Returns the properties of the given PROGBAR skin. (Explained at the beginning of the chapter)
PROGBAR_SetDefaultSkin()	Sets the default skin used for new PROGBAR widgets. (Explained at the beginning of the chapter)
PROGBAR_SetDefaultSkinClassic()	Sets the classical design as default for new PROGBAR widgets. (Explained at the beginning of the chapter)
PROGBAR_SetSkin()	Sets a skin for the given PROGBAR widget. (Explained at the beginning of the chapter)
PROGBAR_SetSkinClassic()	Sets the classical design for the given PROGBAR widget. (Explained at the beginning of the chapter)
PROGBAR_SetSkinFlexProps()	Sets the properties of the given PROGBAR skin.

Table 22.86: PROGBAR Skinning API list

PROGBAR_SetSkinFlexProps()

Before	After
	

Table 22.87: PROGBAR_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the colors of the skin.

Prototype

```
void PROGBAR_SetSkinFlexProps(const PROGBAR_SKINFLEX_PROPS * pProps,
                              int                            Index);
```

Parameter	Description
pProps	Pointer to a structure of type PROGBAR_SKINFLEX_PROPS.
Index	Should be 0.

Table 22.88: PROGBAR_SetSkinFlexProps() parameter list

Additional information

The function passes a pointer to a PROGBAR_SKINFLEX_PROPS structure. It can be used to set up the colors of the skin.
The function PROGBAR_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.15.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the PROGBAR_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the background.
WIDGET_ITEM_DRAW_FRAME	The skinning function should draw the frame.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the text.

Table 22.89: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BACKGROUND

The skinning routine should draw the background.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Is always 0.
x0	Leftmost coordinate of widget area in window coordinates.
y0	Topmost coordinate of widget area in window coordinates.
x1	Rightmost coordinate of widget area in window coordinates.
y1	Bottommost coordinate of widget area in window coordinates.
p	Pointer to a PROGBAR_SKINFLEX_INFO structure.

Table 22.90: WIDGET_ITEM_DRAW_INFO element list

Elements of structure PROGBAR_SKINFLEX_INFO

Data type	Element	Description
int	IsVertical	0 if the progress bar is horizontal, 1 if it is vertical.
int	Index	See table below.
const char *	pText	Pointer to the text to be drawn.

Table 22.91: PROGBAR_SKINFLEX_INFO element list

Permitted values for element Index	
PROGBAR_SKINFLEX_L	Horizontal progress bar: The left part should be drawn. Vertical progress bar: The top part should be drawn.
PROGBAR_SKINFLEX_R	Horizontal progress bar: The right part should be drawn. Vertical progress bar: The bottom part should be drawn.

Additional Information

The message is sent twice, once for the left/top part and once for the right/bottom part of the progress bar. The information in the PROGBAR_SKINFLEX_INFO structure pointed by element p of the WIDGET_ITEM_DRAW_INFO structure can be used to get the information what exactly should be drawn. The parameters x0, y0, x1 and y1 of the WIDGET_ITEM_DRAW_INFO structure mark only the area which should be drawn, left/right or top/bottom.

WIDGET_ITEM_DRAW_FRAME

The skinning routine should draw the surrounding frame.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Is always 0.
x0	Leftmost coordinate of widget area in window coordinates.
y0	Topmost coordinate of widget area in window coordinates.
x1	Rightmost coordinate of widget area in window coordinates.
y1	Bottommost coordinate of widget area in window coordinates.

Table 22.92: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_TEXT

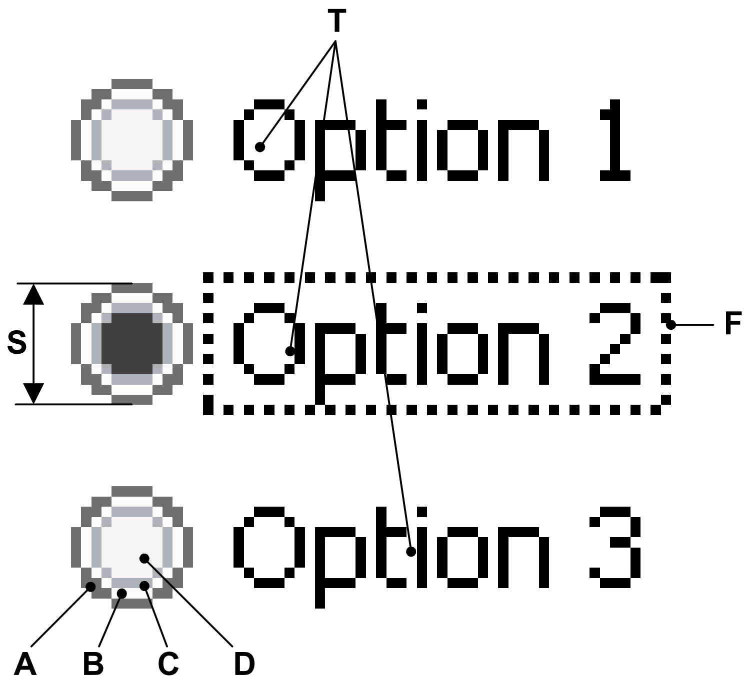
The skinning routine should draw the text.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_FRAME” on page 917.

22.16 RADIO_SKIN_FLEX

The following picture shows the details of the skin:



The RADIO skin consists of a configurable button and a text for each item. If the widget has the input focus the currently selected item text is surrounded by a focus rectangle:

Detail	Description
A	Outer color of button frame.
B	Middle color of button frame.
C	Inner color of button frame.
D	Inner color of button.
F	Focus rectangle.
S	Size of button.
T	Item text.

Table 22.93: Skin details

22.16.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `RADIO_SKINFLEX_PROPS` are used:

Elements of structure RADIO_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorButton[4]	[0] - Outer color of button frame. [1] - Middle color of button frame. [2] - Inner color of button frame. [3] - Inner color of button.
int	ButtonSize	Size of the button in pixels.

Table 22.94: RADIO_SKINFLEX_PROPS element list

22.16.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
RADIO_SKINPROPS_CHECKED	Defines the default attributes used for checked state.
RADIO_SKINPROPS_UNCHECKED	Defines the default attributes used for unchecked state.

Table 22.95: Configuration options

22.16.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
RADIO_DrawSkinFlex()	Skinning callback function of RADIO_SKIN_FLEX. (Explained at the beginning of the chapter)
RADIO_GetSkinFlexProps()	Returns the properties of the given RADIO skin. (Explained at the beginning of the chapter)
RADIO_SetDefaultSkin()	Sets the default skin used for new RADIO widgets. (Explained at the beginning of the chapter)
RADIO_SetDefaultSkinClassic()	Sets the classical design as default for new RADIO widgets. (Explained at the beginning of the chapter)
RADIO_SetSkin()	Sets a skin for the given RADIO widget. (Explained at the beginning of the chapter)
RADIO_SetSkinClassic()	Sets the classical design for the given RADIO widget. (Explained at the beginning of the chapter)
RADIO_SetSkinFlexProps()	Sets the properties of the given RADIO skin.

Table 22.96: RADIO Skinning API list

RADIO_SetSkinFlexProps()

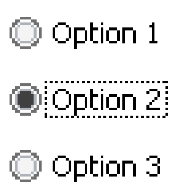
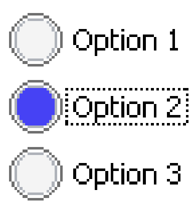
Before	After
	

Table 22.97: RADIO_SetSkinFlexProps() before after screenshots

Description

The function can be used to change the colors of the skin and the size of the button.

Prototype

```
void RADIO_SetSkinFlexProps(const RADIO_SKINFLEX_PROPS * pProps,
                            int                          Index);
```

Parameter	Description
pProps	Pointer to a structure of type RADIO_SKINFLEX_PROPS.
Index	Should be 0.

Table 22.98: RADIO_SetSkinFlexProps() parameter list

Additional information

The function passes a pointer to a RADIO_SKINFLEX_PROPS structure. It can be used to set up the colors and the button size of the skin.
The function RADIO_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.16.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the RADIO_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BUTTON	The skinning function should draw the button of one item.
WIDGET_ITEM_DRAW_FOCUS	The skinning function should draw the focus rectangle.
WIDGET_ITEM_DRAW_TEXT	The skinning function should draw the text of one item.
WIDGET_ITEM_GET_BUTTONSIZE	The skinning function should return the button size.

Table 22.99: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BUTTON

The skinning routine should draw the button of one item.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the button area in window coordinates.
y0	Topmost coordinate of the button area in window coordinates.
x1	Rightmost coordinate of the button area in window coordinates.
y1	Bottommost coordinate of the button area in window coordinates.

Table 22.100: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_FOCUS

The skinning routine should draw the focus rectangle around the text of the currently selected item.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the focus rectangle in window coordinates.
y0	Topmost coordinate of the focus rectangle in window coordinates.
x1	Rightmost coordinate of the focus rectangle in window coordinates.
y1	Bottommost coordinate of the focus rectangle in window coordinates.

Table 22.101: WIDGET_ITEM_DRAW_INFO element list

Additional Information

The given rectangular area in x0, y0, x1 and y1 considers the font settings and the item text.

WIDGET_ITEM_DRAW_TEXT

The skinning routine should draw the text of one item.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the text area in window coordinates.
y0	Topmost coordinate of the text area in window coordinates.
x1	Rightmost coordinate of the text area in window coordinates.
y1	Bottommost coordinate of the text area in window coordinates.

Table 22.102: WIDGET_ITEM_DRAW_INFO element list

Additional Information

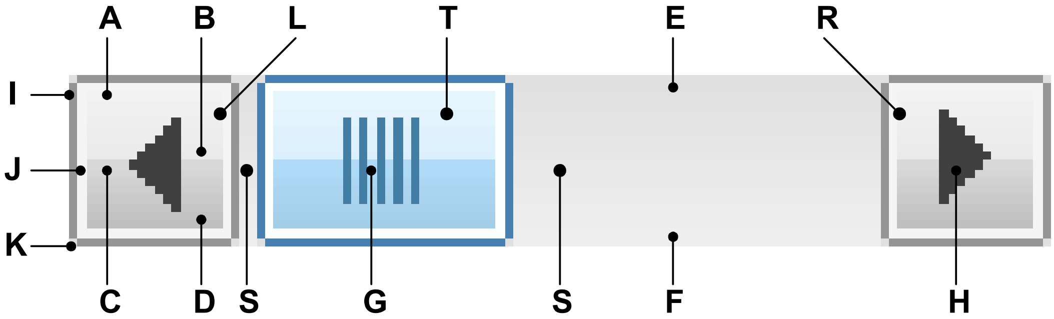
The given rectangular area in x0, y0, x1 and y1 considers the font settings and the item text.

WIDGET_ITEM_GET_BUTTONSIZE

The skinning routine should return the button size.

22.17 SCROLLBAR_SKIN_FLEX

The following picture shows the details of the skin:



The SCROLLBAR skin consists of a left and a right button with an arrow, a shaft area and a thumb with a grasp:

Detail	Description
A	Top color of top gradient.
B	Bottom color of top gradient.
C	Top color of bottom gradient.
D	Bottom color of bottom gradient.
E	Top color of shaft gradient.
F	Bottom color of shaft gradient.
G	Grasp of thumb area.
H	Button arrow.
I	Outer frame color.
J	Inner frame color.
K	Color of frame edges.
L	Left button.
T	Thumb area.
R	Right button.
S	Shaft area.

Table 22.103: SCROLLBAR Skinning API list

22.17.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `SCROLLBAR_SKINFLEX_PROPS` are used:

Elements of structure SCROLL_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorFrame[3]	[0] - Outer frame color. [1] - Inner frame color. [2] - Color of frame edges
U32	aColorUpper[2]	[0] - Top color of top gradient. [1] - Bottom color of top gradient.
U32	aColorLower[2]	[0] - Top color of bottom gradient. [1] - Bottom color of bottom gradient.
U32	aColorShaft[2]	[0] - Top color of shaft gradient. [1] - Bottom color of shaft gradient.
U32	ColorArrow	Color of button arrow.
U32	ColorGrasp	Color of grasp.

Table 22.104: SCROLL_SKINFLEX_PROPS element list

22.17.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
SCROLLBAR_SKINPROPS_PRESSED	Defines the default attributes used for pressed state.
SCROLLBAR_SKINPROPS_UNPRESSED	Defines the default attributes used for unpressed state.

Table 22.105: Configuration options

22.17.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
SCROLLBAR_DrawSkinFlex()	Skinning callback function of SCROLLBAR_SKIN_FLEX. (Explained at the beginning of the chapter)
SCROLLBAR_GetSkinFlexProps()	Returns the properties of the given SCROLLBAR skin. (Explained at the beginning of the chapter)
SCROLLBAR_SetDefaultSkin()	Sets the default skin used for new SCROLLBAR widgets. (Explained at the beginning of the chapter)
SCROLLBAR_SetDefaultSkinClassic()	Sets the classical design as default for new SCROLLBAR widgets. (Explained at the beginning of the chapter)
SCROLLBAR_SetSkin()	Sets a skin for the given SCROLLBAR widget. (Explained at the beginning of the chapter)
SCROLLBAR_SetSkinClassic()	Sets the classical design for the given SCROLLBAR widget. (Explained at the beginning of the chapter)
SCROLLBAR_SetSkinFlexProps()	Sets the properties of the given SCROLLBAR skin.

Table 22.106: SCROLLBAR Skinning API list

SCROLLBAR_SetSkinFlexProps()

Before	After
	

Table 22.107: SCROLLBAR_SetSkinFlexProps() before after screenshot

Description

The function can be used to change the colors of the skin.

Prototype

```
void SCROLLBAR_SetSkinFlexProps(const SCROLLBAR_SKINFLEX_PROPS * pProps,
                                int                              Index);
```

Parameter	Description
pProps	Pointer to a structure of type SCROLLBAR_SKINFLEX_PROPS.
Index	See table below.

Table 22.108: SCROLLBAR_SetSkinFlexProps() parameter list

Permitted values for parameter `Index`	
SCROLLBAR_SKINFLEX_PI_PRESSED	Properties for pressed state.
SCROLLBAR_SKINFLEX_PI_UNPRESSED	Properties for unpressed state.

Additional information

The function passes a pointer to a SCROLLBAR_SKINFLEX_PROPS structure. It can be used to set up the colors of the skin.
The function SCROLLBAR_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.17.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the SCROLLBAR_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BUTTON_L	The skinning function should draw the left button.
WIDGET_ITEM_DRAW_BUTTON_R	The skinning function should draw the right button.
WIDGET_ITEM_DRAW_OVERLAP	The skinning function should draw the overlapping area.
WIDGET_ITEM_DRAW_SHAFT_L	The skinning function should draw the left part of the shaft.
WIDGET_ITEM_DRAW_SHAFT_R	The skinning function should draw the right part of the shaft.
WIDGET_ITEM_DRAW_THUMB	The skinning function should draw the thumb.
WIDGET_ITEM_GET_BUTTONSIZE	The skinning function should return the button size.

Table 22.109: List of commands

WIDGET_ITEM_DRAW_BUTTON_L, WIDGET_ITEM_DRAW_BUTTON_R

The skinning routine should draw a button.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the button in window coordinates.
y0	Topmost coordinate of the button in window coordinates.
x1	Rightmost coordinate of the button in window coordinates.
y1	Bottommost coordinate of the button in window coordinates.
p	Pointer to a SCROLLBAR_SKINFLEX_INFO structure.

Table 22.110: WIDGET_ITEM_DRAW_INFO element list

Elements of structure SCROLLBAR_SKINFLEX_INFO

Data type	Element	Description
int	IsVertical	0 if the progress bar is horizontal, 1 if it is vertical.
int	State	See table below.

Table 22.111: SCROLLBAR_SKINFLEX_INFO element list

Permitted values for element State	
PRESSED_STATE_NONE	Nothing is pressed.
PRESSED_STATE_RIGHT	The right button is pressed.
PRESSED_STATE_LEFT	The left button is pressed.
PRESSED_STATE_THUMB	The thumb is pressed.

WIDGET_ITEM_DRAW_OVERLAP

The skinning routine should draw the thumb.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
`hWin`	Handle to the widget.
`ItemIndex`	Index of item to be drawn.
`x0`	Leftmost coordinate of the overlapping area in window coordinates.
`y0`	Topmost coordinate of the overlapping area in window coordinates.
`x1`	Rightmost coordinate of the overlapping area in window coordinates.
`y1`	Bottommost coordinate of the overlapping area in window coordinates.

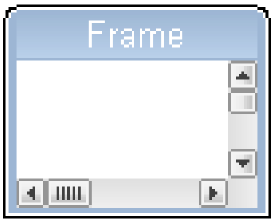
Table 22.112: WIDGET_ITEM_DRAW_INFO element list

Additional information

An overlapping area can exist if a dialog has a vertical and a horizontal scroll bar at the borders. Normally the overlapping region looks identical to the shaft area.

Example

The following screenshot shows a window with 2 scroll bars which have an overlapping region at the lower right corner of the client window:



WIDGET_ITEM_DRAW_SHAFT_L, WIDGET_ITEM_DRAW_SHAFT_R

The skinning routine should draw a shaft area.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
`hWin`	Handle to the widget.
`ItemIndex`	Index of item to be drawn.
`x0`	Leftmost coordinate of the shaft area in window coordinates.
`y0`	Topmost coordinate of the shaft area in window coordinates.
`x1`	Rightmost coordinate of the shaft area in window coordinates.
`y1`	Bottommost coordinate of the shaft area in window coordinates.

Table 22.113: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_THUMB

The skinning routine should draw the thumb.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the thumb area in window coordinates.
y0	Topmost coordinate of the thumb area in window coordinates.
x1	Rightmost coordinate of the thumb area in window coordinates.
y1	Bottommost coordinate of the thumb area in window coordinates.
p	Pointer to a SCROLLBAR_SKINFLEX_INFO structure.

Table 22.114: WIDGET_ITEM_DRAW_INFO element list

SCROLLBAR_SKINFLEX_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BUTTON_L, WIDGET_ITEM_DRAW_BUTTON_R” on page 924.

WIDGET_ITEM_GET_BUTTONSIZE

The skinning routine should return the button size. The button size means the following:

- A horizontal scroll bar should return the height of the scroll bar.
- A vertical scroll bar should return the width of the scroll bar.

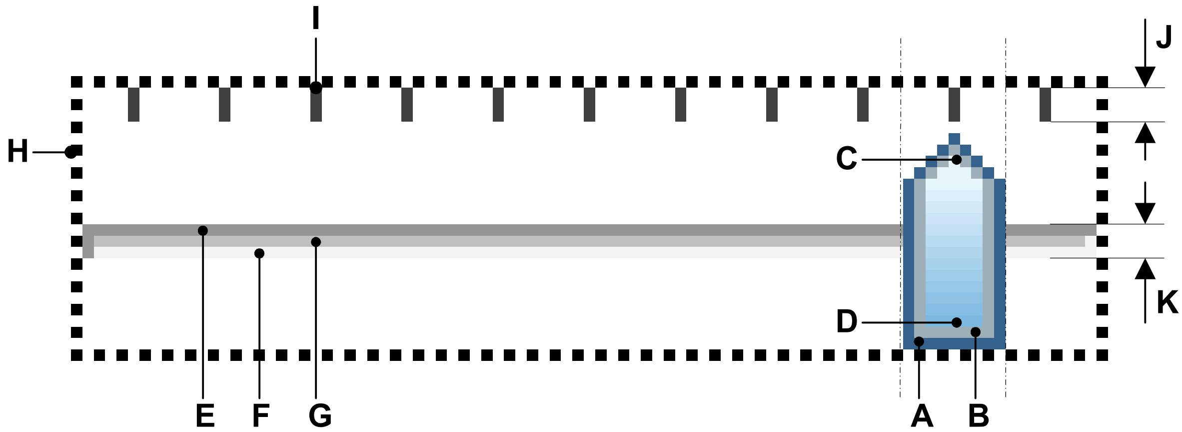
Example

The following code can be used to return the right values in most cases:

```
int _SkinningCallback(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  SCROLLBAR_SKINFLEX_INFO * pSkinInfo;

  pSkinInfo = (SCROLLBAR_SKINFLEX_INFO *)pDrawItemInfo->p;
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_GET_BUTTONSIZE:
    return (pSkinInfo->IsVertical) ?
           pDrawItemInfo->x1 - pDrawItemInfo->x0 + 1 :
           pDrawItemInfo->y1 - pDrawItemInfo->y0 + 1;
    ...
  }
}
```


22.18 SLIDER_SKIN_FLEX

The following picture shows the details of the skin:



The SLIDER skin consists of a shaft with slider and tick marks above. Further a focus rectangle is shown if the widget has the input focus. The slider is drawn by a frame and a gradient:

Detail	Description
A	Outer color of slider frame.
B	Inner color of slider frame
C	Top color of gradient.
D	Bottom color of gradient.
E	First color of shaft.
F	Second color of shaft.
G	Third color of shaft.
H	Focus rectangle.
I	Tick marks.
J	Size of a tick mark.
K	Size of the shaft.

Table 22.115: Skin details

22.18.1 Configuration structure

To set up the default appearance of the skin or to change it at run time configuration structures of type `SLIDER_SKINFLEX_PROPS` are used:

Elements of structure SLIDER_SKINFLEX_PROPS

Data type	Element	Description
U32	aColorFrame[2]	[0] - Outer frame color. [1] - Inner frame color.
U32	aColorInner[2]	[0] - Top color of gradient. [1] - Bottom color of gradient.
U32	aColorShaft[3]	[0] - First frame color of shaft. [1] - Second frame color of shaft. [2] - Inner color of shaft.
U32	ColorTick	Color of tick marks.
U32	ColorFocus	Color of focus rectangle.
int	TickSize	Size of tick marks.
int	ShaftSize	Size of shaft.

Table 22.116: SLIDER_SKINFLEX_PROPS element list

22.18.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
SLIDER_SKINPROPS_PRESSED	Defines the default attributes used for pressed state.
SLIDER_SKINPROPS_UNPRESSED	Defines the default attributes used for unpressed state.

Table 22.117: Configuration options

22.18.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
SLIDER_DrawSkinFlex()	Skinning callback function of SLIDER_SKIN_FLEX. (Explained at the beginning of the chapter)
SLIDER_GetSkinFlexProps()	Returns the properties of the given SLIDER skin. (Explained at the beginning of the chapter)
SLIDER_SetDefaultSkin()	Sets the default skin used for new SLIDER widgets. (Explained at the beginning of the chapter)
SLIDER_SetDefaultSkinClassic()	Sets the classical design as default for new SLIDER widgets. (Explained at the beginning of the chapter)
SLIDER_SetSkin()	Sets a skin for the given SLIDER widget. (Explained at the beginning of the chapter)
SLIDER_SetSkinClassic()	Sets the classical design for the given SLIDER widget. (Explained at the beginning of the chapter)
SLIDER_SetSkinFlexProps()	Sets the properties of the given SLIDER skin.

Table 22.118: SLIDER Skinning API list

SLIDER_SetSkinFlexProps()

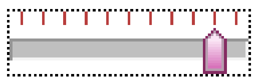
Before	After
	

Table 22.119: SLIDER_SetSkinFlexProps() before after screenshots

Description

The function can be used to change colors, tick mark and shaft size of the skin.

Prototype

```
void SLIDER_SetSkinFlexProps(const SLIDER_SKINFLEX_PROPS * pProps,
                             int                           Index);
```

Parameter	Description
pProps	Pointer to a structure of type SLIDER_SKINFLEX_PROPS.
Index	See table below.

Table 22.120: SLIDER_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
SLIDER_SKINFLEX_PI_PRESSED	Properties for pressed state.
SLIDER_SKINFLEX_PI_UNPRESSED	Properties for unpressed state.

Additional information

The function passes a pointer to a SLIDER_SKINFLEX_PROPS structure. It can be used to set up the colors of the skin.
The function SLIDER_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.18.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the SLIDER_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_FOCUS	The skinning function should draw the focus rectangle.
WIDGET_ITEM_DRAW_SHAFT	The skinning function should draw the shaft.
WIDGET_ITEM_DRAW_THUMB	The skinning function should draw the slider.
WIDGET_ITEM_DRAW_TICKS	The skinning function should draw the tick marks.

Table 22.121: List of commands

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_FOCUS

The skinning routine should draw the focus rectangle.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the widget in window coordinates.
y0	Topmost coordinate of the widget in window coordinates.
x1	Rightmost coordinate of the widget in window coordinates.
y1	Bottommost coordinate of the widget in window coordinates.

Table 22.122: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_SHAFT

The skinning routine should draw the shaft.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the widget + 1 in window coordinates.
y0	Topmost coordinate of the widget + 1 in window coordinates.
x1	Rightmost coordinate of the widget - 1 in window coordinates.
y1	Bottommost coordinate of the widget - 1 in window coordinates.

Table 22.123: WIDGET_ITEM_DRAW_INFO element list

WIDGET_ITEM_DRAW_THUMB

The skinning routine should draw the slider itself.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the slider in window coordinates.
y0	Topmost coordinate of the slider in window coordinates.
x1	Rightmost coordinate of the slider in window coordinates.
y1	Bottommost coordinate of the slider in window coordinates.
p	Pointer to a SLIDER_SKINFLEX_INFO structure.

Table 22.124: WIDGET_ITEM_DRAW_INFO element list

SLIDER_SKINFLEX_INFO

Data type	Element	Description
int	Width	With of the slider.
int	IsPressed	1 if the slider is pressed, 0 if not.
int	IsVertical	0 if the slider is horizontal, 1 if it is vertical.

Table 22.125: SLIDER_SKINFLEX_INFO

WIDGET_ITEM_DRAW_TICKS

The skinning routine should draw the tick marks.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	Index of item to be drawn.
x0	Leftmost coordinate of the widget + 1 in window coordinates.
y0	Topmost coordinate of the widget + 1 in window coordinates.
x1	Rightmost coordinate of the widget - 1 in window coordinates.
y1	Bottommost coordinate of the widget - 1 in window coordinates.
p	Pointer to a SLIDER_SKINFLEX_INFO structure.

Table 22.126: WIDGET_ITEM_DRAW_INFO element list

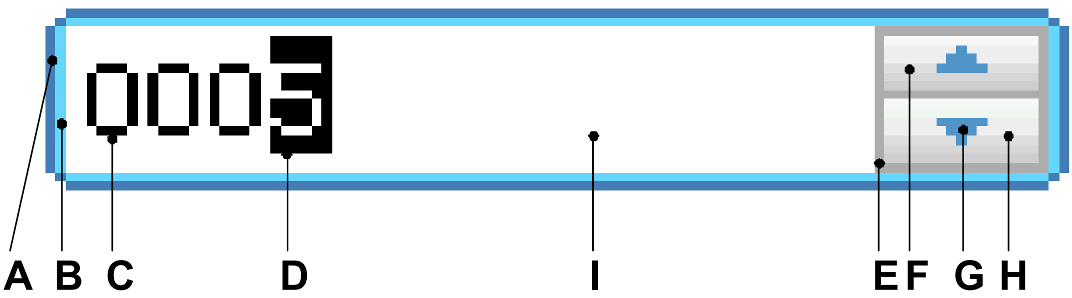
Elements of structure SLIDER_SKINFLEX_INFO

Data type	Element	Description
int	Width	With of the slider.
int	NumTicks	Number of ticks to be drawn.
int	Size	Length of the tick mark line.
int	IsPressed	1 if the slider is pressed, 0 if not.
int	IsVertical	0 if the slider is horizontal, 1 if it is vertical.

Table 22.127: SLIDER_SKINFLEX_INFO element list

22.19 SPINBOX_SKIN_FLEX

The following picture shows the details of the skin:



The SPINBOX skin consists of a rounded border and 2 rectangular inner areas which are drawn in dependence of the size of the EDIT widget. The background color of the EDIT widget is set to the set color of the inner area of the SPINBOX widget. The 2 buttons are drawn each with a gradient of 2 colors.

Detail	Description
A	Outer color of surrounding frame.
B	Inner color of surrounding frame.
C	Color of the displayed value.
D	Color of the text cursor (always inverse).
E	Color of the button frame.
F	2 color gradient of the upper button.
G	Arrow color.
H	2 color gradient of the lower button.
I	Background color.

Table 22.128: Skin details

22.19.1 Configuration structure

To set up the default appearance of the skin or to change it at run time, configuration structures of type `SPINBOX_SKINFLEX_PROPS` are used:

Elements of structure SPINBOX_SKINFLEX_PROPS

Data type	Element	Description
GUI_COLOR	aColorFrame[2]	[0] - Outer color of the surrounding frame. [1] - Inner color of the surrounding frame.
GUI_COLOR	aColorUpper[2]	[0] - Upper gradient color of the upper button. [1] - Lower gradient color of the upper button.
GUI_COLOR	aColorLower[2]	[0] - Upper gradient color of the lower button. [1] - Lower gradient color of the lower button.
GUI_COLOR	ColorArrow	Color of the button arrows.
GUI_COLOR	ColorBk	Color of the background.
GUI_COLOR	ColorText	Color of the text.
GUI_COLOR	ColorButtonFrame	Color of the button frame.

Table 22.129: SPINBOX_SKINFLEX_PROPS element list

22.19.2 Configuration options

The default appearance of the skin can be determined by setting custom configuration structures of the above type in `GUIConf.h`. The following table shows the available configuration options:

Macro	Description
SPINBOX_SKINPROPS_PRESSED	Defines the default attributes used for pressed state.
SPINBOX_SKINPROPS_FOCUSSED	Defines the default attributes used for focussed state.
SPINBOX_SKINPROPS_ENABLED	Defines the default attributes used for enabled state.
SPINBOX_SKINPROPS_DISABLED	Defines the default attributes used for disabled state.

Table 22.130: Configuration options

22.19.3 Skinning API

The table below lists the available routines in alphabetical order:

Routine	Description
SPINBOX_DrawSkinFlex()	Skinning callback function of SPINBOX_SKIN_FLEX. (Explained at the beginning of the chapter)
SPINBOX_GetSkinFlexProps()	Returns the properties of the given spinbox skin. (Explained at the beginning of the chapter)
SPINBOX_SetDefaultSkin()	Sets the default skin used for new spinbox widgets. (Explained at the beginning of the chapter)
SPINBOX_SetDefaultSkinClassic()	Sets the classical design as default for new spinbox widgets. (Explained at the beginning of the chapter)
SPINBOX_SetSkin()	Sets a skin for the given spinbox widget. (Explained at the beginning of the chapter)
SPINBOX_SetSkinClassic()	Sets the classical design for the given spinbox widget. (Explained at the beginning of the chapter)
SPINBOX_SetSkinFlexProps()	Sets the properties of the given spinbox skin.

Table 22.131: SPINBOX Skinning API list

SPINBOX_SetSkinFlexProps()

Before	After
	

Table 22.132: SPINBOX_SetSkinFlexProps() before after screenshotsssss

Description

The function can be used to change the properties of the skin.

Prototype

```
void SPINBOX_SetSkinFlexProps(const SPINBOX_SKINFLEX_PROPS * pProps,
                              int                            Index);
```

Parameter	Description
pProps	Pointer to a structure of type SPINBOX_SKINFLEX_PROPS.
Index	See table below.

Table 22.133: SPINBOX_SetSkinFlexProps() parameter list

Permitted values for parameter Index	
SPINBOX_SKINFLEX_PI_PRESSED	Properties for pressed state.
SPINBOX_SKINFLEX_PI_FOCUSSED	Properties for focussed state.
SPINBOX_SKINFLEX_PI_ENABLED	Properties for enabled state.
SPINBOX_SKINFLEX_PI_DISABLED	Properties for disabled state.

Additional information

The function passes a pointer to a SPINBOX_SKINFLEX_PROPS structure. It can be used to set up the colors and the radius of the skin.
The function SPINBOX_GetSkinFlexProps() can be used to get the current attributes of the skin.

22.19.4 List of commands

The skinning routine receives a pointer to a WIDGET_ITEM_DRAW_INFO structure. The Cmd member of this structure contains the command which needs to be processed. The following table shows all commands passed to the SPINBOX_SKIN_FLEX callback function:

Command	Description
WIDGET_ITEM_CREATE	Is sent immediately after creating the widget.
WIDGET_ITEM_DRAW_BACKGROUND	The skinning function should draw the background.
WIDGET_ITEM_DRAW_BUTTON_L	The skinning function should draw the upper button.
WIDGET_ITEM_DRAW_BUTTON_R	The skinning function should draw the lower button.
WIDGET_ITEM_DRAW_FRAME	The skinning function should draw the surrounding frame.

Table 22.134: List of commands

The WIDGET_ITEM_DRAW_INFO structure is explained at the beginning of the chapter.

WIDGET_ITEM_CREATE

The skinning routine should, if necessary, set up skin related properties like e.g. transparency or text alignment.

WIDGET_ITEM_DRAW_BACKGROUND

The background should be drawn.

Elements of structure WIDGET_ITEM_DRAW_INFO

Element	Description
hWin	Handle to the widget.
ItemIndex	See table below.
x0	Leftmost window coordinate, normally 0.
y0	Topmost window coordinate, normally 0.
x1	Rightmost window coordinate.
y1	Bottommost window coordinate.

Table 22.135: WIDGET_ITEM_DRAW_INFO element list

Permitted values for element ItemIndex	
SPINBOX_SKINFLEX_PI_PRESSED	The widget is pressed.
SPINBOX_SKINFLEX_PI_FOCUSSED	The widget is not pressed but focussed.
SPINBOX_SKINFLEX_PI_ENABLED	The widget is not focussed but enabled.
SPINBOX_SKINFLEX_PI_DISABLED	The widget is disabled.

WIDGET_ITEM_DRAW_BUTTON_L

The upper button should be drawn.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 933.

WIDGET_ITEM_DRAW_BUTTON_R

The lower button should be drawn.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 933.

WIDGET_ITEM_DRAW_FRAME

The surrounding frame should be drawn.

WIDGET_ITEM_DRAW_INFO

A detailed description of the elements can be found under “WIDGET_ITEM_DRAW_BACKGROUND” on page 933.

Chapter 23

Multiple Buffering

Multiple Buffering is a method of using more than one frame buffer. Basically it works as follows: With multiple buffers enabled there is a front buffer which is used by the display controller to generate the picture on the screen and one or more back buffers which are used for the drawing operations. After completing the drawing operations the back buffer becomes the visible front buffer.

With two buffers, one front and one back buffer, it is normally called ’double buffering’, with two back buffers and one front buffer it is called ’triple buffering’.

In general it is a method which is able to avoid several unwanted effects:

- The visible process of drawing a screen item by item
- Flickering effects caused by overlapping drawing operations
- Tearing effects caused by writing operations outside the vertical blanking period

The following section explains in detail how it works, the requirements to be able to use this feature, how to configure emWin and the advantage of ’triple buffering’ against ’double buffering’. Further it explains how to configure the optional Window Manager for automatic use of Multiple Buffering.

23.1 How it works

Multiple Buffering is the use of more than one frame buffer, so that the display ever shows a screen which is already completely rendered, even if a drawing operation is in process. When starting the process of drawing the current content of the front buffer is copied into a back buffer. After that all drawing operations take effect only on this back buffer. After the drawing operation has been completed the back buffer becomes the front buffer. Making the back buffer the visible front buffer normally only requires the modification of the frame buffer start address register of the display controller.
Now it should be considered that a display is being refreshed continuously by the display controller app. 60 times per second. After each period there is a vertical synchronization signal, normally known as VSYNC signal. The best moment to make the back buffer the new front buffer is this signal. If not considering the VSYNC signal tearing effects can occur.

Tearing effect

23.1.1 Double buffering

With double buffering only 2 buffers are available: One front and one back buffer. When starting the drawing operation the current content of the front buffer is copied into the back buffer. After completing the operation the back buffer should become the visible front buffer.
As explained above the best moment for doing this is reacting on the VSYNC signal of the display controller. Here the disadvantage of double buffering against triple buffering is revealed: Either the frame buffer start address is changed immediately at the end of the drawing operation or after waiting until the next VSYNC signal. This means that either tearing effects could occur or the performance slows down because of waiting for the next VSYNC signal.

23.1.2 Triple buffering

As the name implies there are 3 buffers available: One front and 2 back buffers. When starting the drawing operation the current content of the front buffer is copied into the first back buffer. After completing the operation the back buffer should become the visible front buffer. Contrary to the double buffer solution it is not required to switch to the buffer immediately. Switching to the new front buffer could be done on the next VSYNC signal of the display controller which can be achieved by an interrupt service routine (ISR). Most of the display controllers which are able to deal with more than one frame buffer provide the VSYNC signal as interrupt source. Within the ISR the pending front buffer should become visible. Until the pending front buffer becomes visible it is not used for further drawing operations. If a further drawing operation is initiated before the pending front buffer has become visible the second back buffer is used for the drawing operation. If a new buffer is ready until waiting for the VSYNC signal it becomes the new pending front buffer and so on. This always protects the front buffer against writing operations.
It should be mentioned that changing the display buffer start address on some display controllers only takes effect when drawing the next frame. In this case the solution without ISR works as well as with ISR. Only if changing the address takes effect directly an ISR is required to avoid tearing effects.

23.2 Requirements

The following list shows the requirements for using multiple buffers:

- The display controller should support multiple frame buffers.
- Enough video RAM for multiple frame buffers should be available.
- If tearing effects should be avoided it should be possible to react on the VSYNC signal of the display controller and triple buffering is recommended to achieve the best performance.

23.3 Limitations

Multiple Buffering can not be used with virtual screens.

23.4 Configuration

In general there are 2 routines in the configuration file `LCDConf.c` which need to be modified, the display configuration routine `LCD_X_Config()` and the driver callback function `LCD_X_DisplayDriver()`.

23.4.1 LCD_X_Config()

Basically one thing needs to be done here: Enabling the use of multiple buffers.

Basic configuration

The first thing which has to be done before creating the display driver device is configuring the multiple buffer interface. This is normally done in `LCD_X_Config()`. It is strictly required to enable Multiple Buffering before creating the display driver device as shown in the following code snippet:

```
void LCD_X_Config(void) {
  //
  // Initialize MultiBuffering
  //
  GUI_MULTIBUF_Config(NUM_BUFFERS);
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  ...
}
```


Custom callback routine for copying the buffers

Further a callback routine for copying the buffers can be set. As explained above at the beginning of the drawing operation it is required to copy the content of the current front buffer to the back buffer. Normally a simple memcpy operation is used to do this. But if the used display controller for example consists of a BitBLT-engine which is able to do the copy operation it could be desired to use it for the copy operation. Or a DMA based routine should be used to do the copy operation. In these cases a custom defined callback function can be used for this operation. It can be installed after creating the display driver device as shown in the following code snippet:

```
static void _CopyBuffer(int LayerIndex, int IndexSrc, int IndexDst) {
  unsigned long BufferSize, AddrSrc, AddrDst;

  //
  // Calculate the size of one frame buffer
  //
  BufferSize = (XSIZE * YSIZE * BITSPERPIXEL) / 8;
  //
  // Calculate source- and destination address
  //
  AddrSrc    = _VRamBaseAddr + BufferSize * IndexSrc;
  AddrDst    = _VRamBaseAddr + BufferSize * IndexDst;
  memcpy((void *)AddrDst, (void *)AddrSrc, BufferSize);
}

void LCD_X_Config(void) {
  //
  // Initialize multibuffering
  //
  GUI_MULTIBUF_Config(NUM_BUFFERS);
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Set custom callback function for copy operation
  //
  LCD_SetDevFunc(0, LCD_DEVFUNC_COPYBUFFER, (void (*)())_CopyBuffer);
}
```

The above sample implementation makes no sense, because the simple call of `memcpy()` equals the default behavior of the display driver. It makes only sense to use a custom copy buffer function if there is any possibility to accelerate the copy operation.

23.4.2 LCD_X_DisplayDriver()

After the drawing process has been completed the back buffer should become visible. The display driver sends a `LCD_X_SHOWBUFFER` command to the display driver callback function. The callback function then has to react on the command and should make sure that the buffer becomes visible. This can be done either by an ISR or by directly writing the right address into the frame buffer start address of the display controller.

With ISR

The following code snippet shows a sample implementation:

```
static void _ISR_EndOfFrame(void) {
  unsigned long Addr, BufferSize;

  if (_PendingBuffer >= 0) {
    //
    // Calculate address of the given buffer
    //
    BufferSize = (XSIZE * YSIZE * BITSPERPIXEL) / 8;
    Addr       = _VRamBaseAddr + BufferSize * pData->Index;
    //
    // Make the given buffer visible
    //
    AT91C_LCDC_BA1 = Addr;
    //
    // Send a confirmation that the buffer is visible now
    //
    GUI_MULTIBUF_Confirm(_PendingBuffer);
    _PendingBuffer = -1;
  }
}

int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * p) {
  LCD_X_SHOWBUFFER_INFO * pData;

  switch (Cmd) {
  ...
  case LCD_X_SHOWBUFFER:
    pData = (LCD_X_SHOWBUFFER_INFO *)p;
    //
    // Remember buffer index to be used by ISR
    //
    _PendingBuffer = pData->Index;
    break;
  }
}
```

The above implementation assumes the existence of an ISR which is executed at the next VSYNC signal.

Without ISR

If there is no ISR available alternatively the address can be set directly with the disadvantage that tearing effects could occur.
The following code snippet shows a sample implementation:

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * p) {
  LCD_X_SHOWBUFFER_INFO * pData;
  unsigned long           BufferSize;
  unsigned long           Addr;

  switch (Cmd) {
  ...
  case LCD_X_SHOWBUFFER: {
    pData = (LCD_X_SHOWBUFFER_INFO *)p;
    //
    // Calculate address of the given buffer
    //
     BufferSize = (XSIZE * YSIZE * BITSPERPIXEL) / 8;
     Addr       = _VRamBaseAddr + BufferSize * pData->Index;
    //
    // Make the given buffer visible
    //
    AT91C_LCDC_BA1 = Addr;
    //
    // Send a confirmation that the buffer is visible now
    //
    GUI_MULTIBUF_Confirm(pData->Index);
    break;
  }
}
```

23.5 Automatic use of multiple buffers with the WM

The optional Window Manager (WM) is able to use the multiple buffer feature automatically. The function `WM_MULTIBUF_Enable()` can be used to enable this function. If enabled the WM first switches to the back buffer before redrawing the invalid windows. After drawing all invalid windows the new screen becomes visible. This hides the process of drawing a screen window by window.

23.6 Multiple Buffering API

The following table lists the available routines of the multiple buffer support.

Routine	Description
GUI_MULTIBUF_Begin()	Needs be called before starting the drawing operation.
GUI_MULTIBUF_BeginEx()	Same as above except the parameter LayerIndex.
GUI_MULTIBUF_Config()	Needs to be called to configure the use of multiple buffers.
GUI_MULTIBUF_ConfigEx()	Same as above except the parameter LayerIndex.
GUI_MULTIBUF_Confirm()	Should be called immediately after the pending front buffer has become visible.
GUI_MULTIBUF_ConfirmEx()	Same as above except the parameter LayerIndex.
GUI_MULTIBUF_End()	Needs be called after completing the drawing operation.
GUI_MULTIBUF_EndEx()	Same as above except the parameter LayerIndex.
GUI_MULTIBUF_GetNumBuffers()	Returns the number of used buffers.
GUI_MULTIBUF_GetNumBuffersEx()	Same as above except the parameter LayerIndex.
GUI_MULTIBUF_UseSingleBuffer()	Lets the multi buffering use one frame for all layers.

Table 23.1: Multiple Buffering API list

(The interface of the above routines may be changed in a later version)

GUI_MULTIBUF_Begin()

Description

Needs to be called immediately before the drawing process.

Prototype

```
void GUI_MULTIBUF_Begin(void);
```

Additional information

This function makes sure that the current front buffer will be copied into the back buffer which then is used for all subsequent drawing operations. The copy operation is normally done by the display driver itself. As explained earlier this can also be achieved by a custom callback function.

GUI_MULTIBUF_BeginEx()

Description

Needs to be called immediately before the drawing process in the given layer.

Prototype

```
void GUI_MULTIBUF_BeginEx(int LayerIndex);
```

Parameter	Description
LayerIndex	Layer to be used.

Table 23.2: GUI_MULTIBUF_BeginEx() parameter list

GUI_MULTIBUF_Config()

Description

The function needs to be called during initialization in order to enable the use of Multiple Buffering. This is done typically from within `LCD_X_Config()`.

Prototype

```
void GUI_MULTIBUF_Config(int NumBuffers);
```

Parameter	Description
NumBuffers	Number of buffers to be used. The following numbers make sense: 2 - Double buffering 3 - Triple buffering

Table 23.3: GUI_MULTIBUF_Config() parameter list

Additional information

The function needs to be called before creating the display driver device.

GUI_MULTIBUF_ConfigEx()

Description

The function needs to be called during initialization in order to enable the use of Multiple Buffering for the given layer. This is done typically from within `LCD_X_Config()`.

Prototype

```
void GUI_MULTIBUF_ConfigEx(int LayerIndex, int NumBuffers);
```

Parameter	Description
LayerIndex	Layer to be used.
NumBuffers	Number of buffers to be used. The following numbers make sense: 2 - Double buffering 3 - Triple buffering

Table 23.4: GUI_MULTIBUF_ConfigEx() parameter list

GUI_MULTIBUF_Confirm()

Description

This function needs to be called immediately after a new buffer has become visible.

Prototype

```
void GUI_MULTIBUF_Confirm(int Index);
```

Parameter	Description
Index	Index of buffer which has been made visible.

Table 23.5: GUI_MULTIBUF_Confirm() parameter list

Additional information

The function is typically called by the ISR which switches to the new front buffer or by the display driver callback function.

GUI_MULTIBUF_ConfirmEx()

Description

This function needs to be called immediately after a new buffer has become visible in the given layer.

Prototype

```
void GUI_MULTIBUF_ConfirmEx(int LayerIndex, int BufferIndex);
```

Parameter	Description
LayerIndex	Layer to be used.
Index	Index of buffer which has been made visible.

Table 23.6: GUI_MULTIBUF_ConfirmEx() parameter list

GUI_MULTIBUF_End()

Description

This function needs to be called after the new screen has been completely drawn.

Prototype

```
void GUI_MULTIBUF_End(void);
```

Additional information

When calling this function the display driver sends an LCD_X_SHOWBUFFER command to the display driver callback routine which then has to make the given buffer the front buffer.

GUI_MULTIBUF_EndEx()

Description

This function needs to be called after the new screen has been completely drawn for the given layer.

Prototype

```
void GUI_MULTIBUF_EndEx(int LayerIndex);
```

Parameter	Description
LayerIndex	Layer to be used.

Table 23.7: GUI_MULTIBUF_EndEx() parameter list

GUI_MULTIBUF_GetNumBuffers()

Description

The function returns the number of buffers configured.

Prototype

```
int GUI_MULTIBUF_GetNumBuffers(void);
```

Return value

The number of buffers configured for the current layer.

GUI_MULTIBUF_GetNumBuffersEx()

Description

The function returns the number of buffers configured for the given layer.

Prototype

```
int GUI_MULTIBUF_GetNumBuffersEx(int LayerIndex);
```

Parameter	Description
LayerIndex	Layer to be used.

Table 23.8: GUI_MULTIBUF_GetNumBuffersEx() parameter list

Return value

The number of buffers configured for the specified layer.

GUI_MULTIBUF_UseSingleBuffer()

Description

Lets the multi buffering use one frame for all layers.

Prototype

```
void GUI_MULTIBUF_UseSingleBuffer(void);
```

Additional information

The function needs to be called before creating the display driver device.

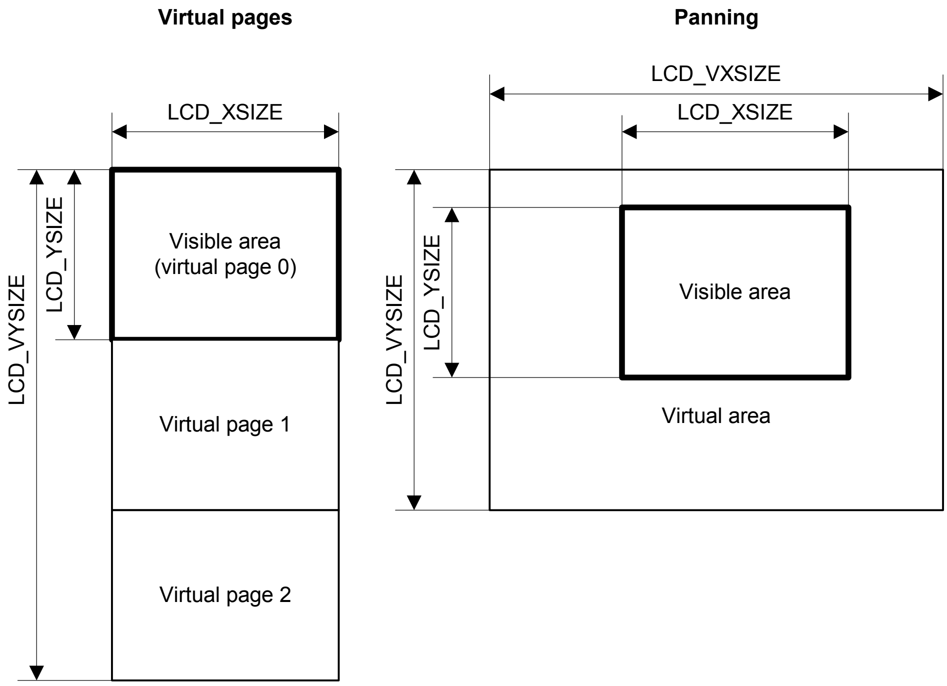
Chapter 24

Virtual screens / Virtual pages

A virtual screen means a display area greater than the physical size of the display. It requires additional video memory and allows instantaneous switching between different screens even on slow CPUs. The following chapter shows
- the requirements for using virtual screens,
- how to configure emWin
- and how to take advantage of virtual screens.

If a virtual display area is configured, the visible part of the display can be changed by setting the origin.

24.1 Introduction



The virtual screen support of emWin can be used for panning or for switching between different video pages.

Panning

If the application uses one screen which is larger than the display, the virtual screen API functions can be used to make the desired area visible.

Virtual pages

Virtual pages are a way to use the display RAM as multiple pages. If an application for example needs 3 different screens, each screen can use its own page in the display RAM. In this case, the application can draw the second and the third page before they are used. After that the application can switch very fast between the different pages using the virtual screen API functions of emWin. The only thing the functions have to do is setting the right display start address for showing the desired screen. In this case the virtual Y-size typically is a multiple of the display size in Y.

24.2 Requirements

The virtual screen feature requires hardware with more display RAM than required for a single screen and the ability of the hardware to change the start position of the display output.

Video RAM

The used display controller should support video RAM for the virtual area. For example if the display has a resolution of 320x240 and a color depth of 16 bits per pixel and 2 screens should be supported, the required size of the video RAM can be calculated as follows:

```
Size = LCD_XSIZE * LCD_YSIZE * LCD_BITSPERPIXEL / 8 * NUM_SCREENS
Size = 320 x 240 x 16 / 8 x 2
Size = 307200 Bytes
```

Configurable display start position

The used display controller needs a configurable display start position. This means the display driver even has a register for setting the frame buffer start address or it has a command to set the upper left display start position.

24.3 Configuration

Virtual screens should be configured during the initialization. The function `LCD_SetVSizeEx()` needs to be used to define the virtual display size. Further it is required to react on the command `LCD_X_SETORG` in the driver callback routine by setting the right frame buffer start address.

LCD_SetVSizeEx()

Description

Sets the virtual display size.

Prototype

```
int LCD_SetVSizeEx(int LayerIndex, int xSize, int ySize);
```

Parameter	Description
`LayerIndex`	Zero based layer index, typically 0 on single layer systems.
`xSize`	Horizontal resolution of virtual display.
`ySize`	Vertical resolution of virtual display.

Table 24.1: LCD_SetVSizeEx() parameter list

Return value

0 on success, 1 on error.

24.4 Examples

In the following a few examples are shown to make clear how to use virtual screens with emWin.

24.4.1 Basic example

The following example shows how to use a virtual screen of 128x192 and a display of 128x64 for instantaneous switching between 3 different screens.

Configuration

```
LCD_SetSizeEx (0, 128,  64);
LCD_SetVSizeEx(0, 128, 192);
```

Application

```
GUI_SetColor(GUI_RED);
GUI_FillRect(0,   0, 127,  63);
GUI_SetColor(GUI_GREEN);
GUI_FillRect(0,  64, 127, 127);
GUI_SetColor(GUI_BLUE);
GUI_FillRect(0, 128, 127, 191);
GUI_SetColor(GUI_WHITE);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringAt("Screen 0", 0,   0);
GUI_DispStringAt("Screen 1", 0,  64);
GUI_DispStringAt("Screen 2", 0, 128);
GUI_SetOrg(0,  64); /* Set origin to screen 1 */
GUI_SetOrg(0, 128); /* Set origin to screen 2 */
```


Output

The table below shows the output of the display:

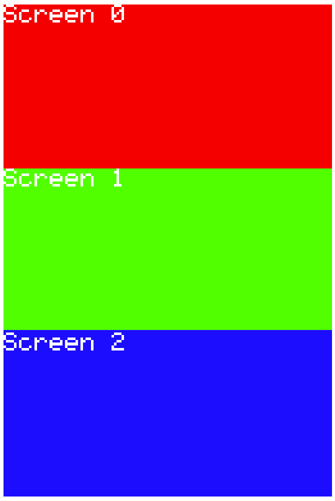
Description	Display output	Contents of virtual area
Before executing `GUI_SetOrg(0, 64)`		
After executing `GUI_SetOrg(0, 64)`		
After executing `GUI_SetOrg(0, 128)`		

Table 24.2: Basic example

24.4.2 Real time example using the Window Manager

The shipment of emWin contains an example which shows how to use virtual screens in a real time application. It can be found under `Sample\Tutorial\VSCREEN_RealTime.c`:

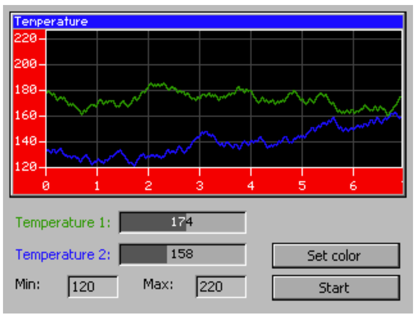
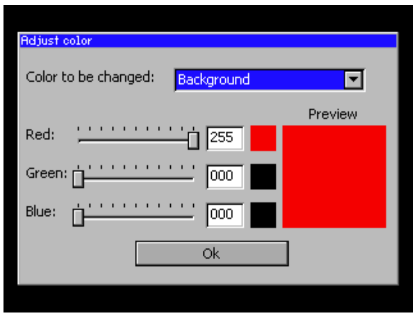
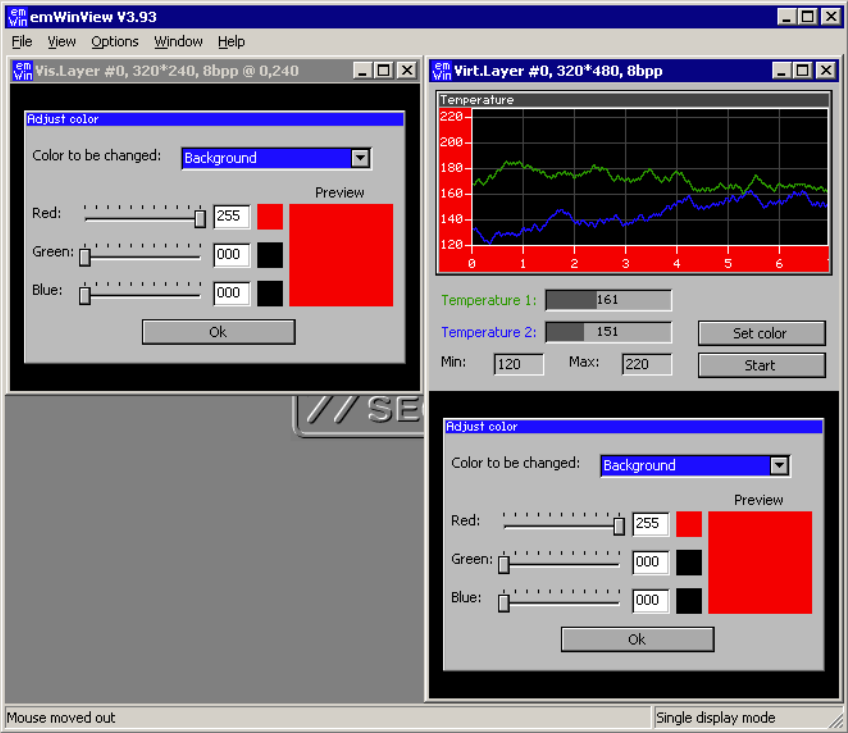
Screen 0 / Page 0	Screen 1 / Page 1
	

Table 24.3: Real time example

After showing a short introduction, the example creates 2 screens on 2 separate pages as shown above. The first screen shows a dialog which includes a graphical representation of 2 temperature curves. When pressing the 'Set color' button, the application switches instantaneously to the second screen, even on slow CPUs. After pressing the 'OK' button of the 'Adjust color' dialog, the application switches back to the first screen.
For more details, see the source code of the example.

Viewer Screenshot of the above example



If using the viewer both screens can be shown at the same time. The screenshot above shows the visible display at the left side and the contents of the whole configured virtual display RAM at the right side.

24.4.3 Dialog example using the Window Manager

The second advanced example is available in the folder `Sample\GUI\VSCREEN_MultiPage`. It uses the virtual screen to show 4 screens on 3 different video pages. The application consists of the following screens:

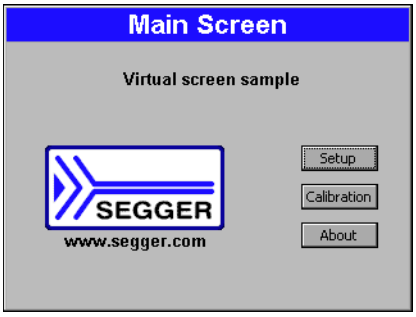
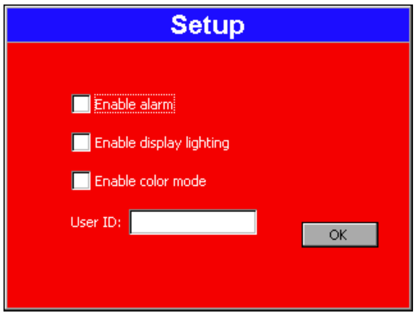
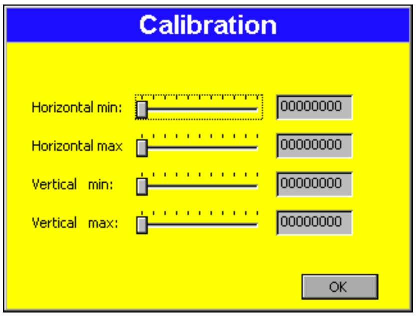
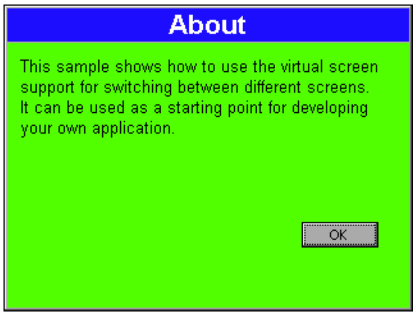
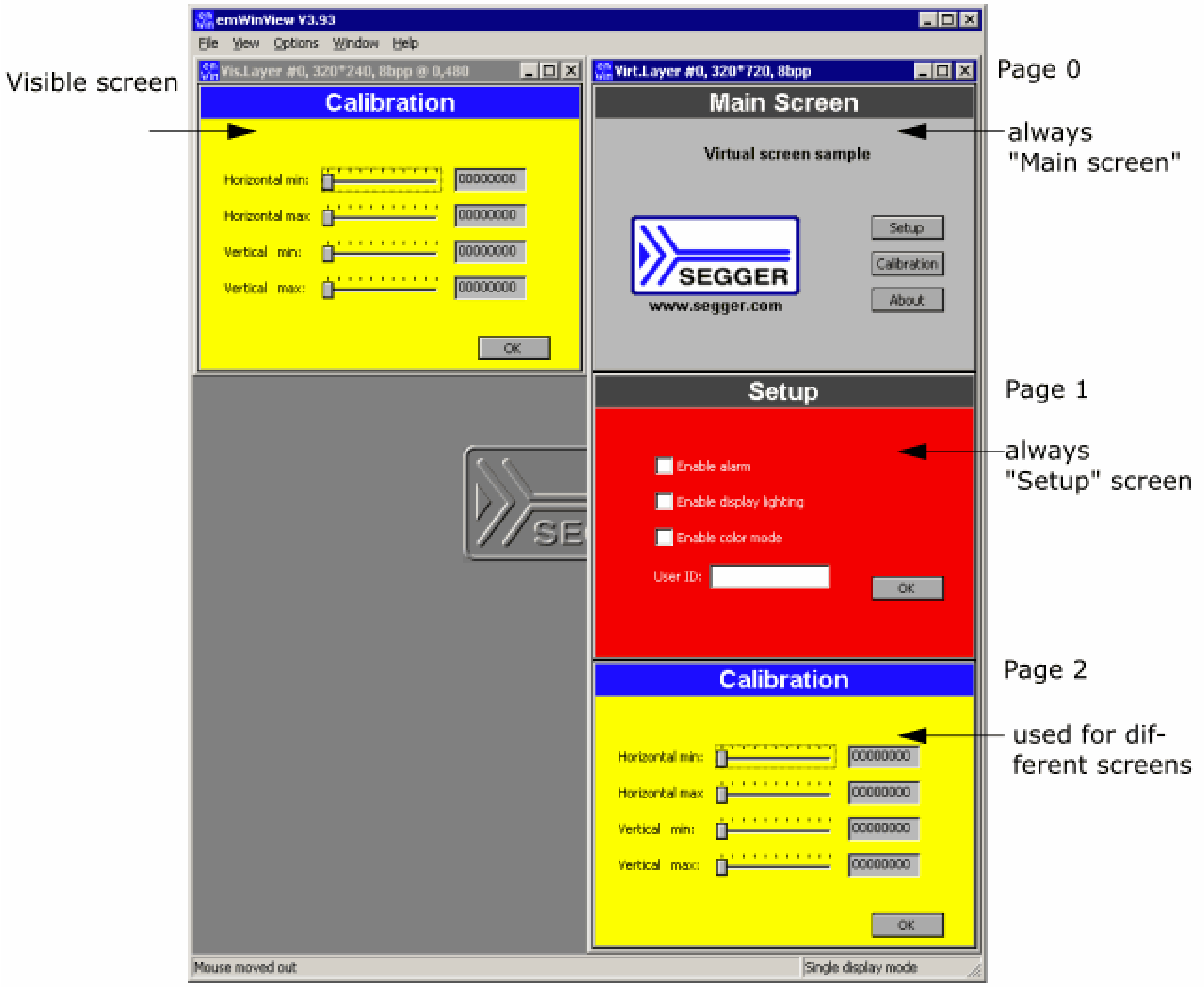
<table>
<tr><th>Main screen / Page 0</th><th>Setup screen / Page 1</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><th>Calibration screen / Page 2</th><th>About screen / Page 2</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Table 24.4: Dialog example

After a short intro screen the 'Main Screen' is shown on the display using page 0. After the 'Setup' button is pressed, the 'Setup' screen is created on page 1. After the screen has been created, the application makes the screen visible by switching to page 1. The 'Calibration' and the 'About' screen both use page 2. If the user presses one of the buttons 'Calibration' or 'About' the application switches to page 2 and shows the dialog.

Viewer Screenshot of the above example



The viewer can show all pages at the same time. The screenshot above shows the visible display at the left side and the contents of the whole layer (virtual display RAM) with the pages 0 - 2 at the right side.

24.5 Virtual screen API

The following table lists the available routines of the virtual screen support.

Routine	Description
GUI_GetOrg()	Returns the display start position.
GUI_SetOrg()	Sets the display start position.

Table 24.5: Virtual Screen API list

GUI_GetOrg()

Description

Returns the display start position.

Prototype

```
void GUI_GetOrg(int * px, int * py);
```

Parameter	Description
px	Pointer to variable of type int to store the X position of the display start position.
py	Pointer to variable of type int to store the Y position of the display start position.

Table 24.6: GUI_GetOrg() parameter list

Additional information

The function stores the current display start position into the variables pointed by the given pointers.

GUI_SetOrg()

Description

Sets the display start position.

Prototype

```
void GUI_SetOrg(int x, int y);
```

Parameter	Description
x	New X position of the display start position.
y	New Y position of the display start position.

Table 24.7: GUI_SetOrg() parameter list

Chapter 25

MultiLayer / MultiDisplay support

Multiple displays and multiple layers can be utilized via emWin MultiLayer support. If the hardware supports multiple layers, MultiLayer support can be used. If the hardware does not include such a function, multiple layers can be implemented using the emWin SoftLayer feature.
MultiLayer and MultiDisplay support work the same way. Each layer / display can be accessed with its own color settings, its own size and its own display driver. Initialization of more than one layer is quite simple: The maximum number of available layers `GUI_NUM_LAYERS` should be defined in `GUIConf.h` and each layer needs a display driver device which should be created during the initialization in the configuration routine `LCD_X_Config()`. There is no limitation regarding the maximum number of available layers.
All SoftLayers use an internal driver which works with 32bpp. The SoftLayer composite is converted to the color setting of the display. SoftLayers and MultiLayer support can not be used in combination. Therefor SoftLayers have to be configured slightly different from MultiLayers.

25.1 Introduction

This chapter deals with multiple hardware layers (MultiLayer), multiple software layers (SoftLayer) and multiple displays (MultiDisplay). Since a lot of information is valid for each of those features, the following sections will just refer to it as layers unless there are functional differences.
Windows and drawing operations can be placed and performed on any layer. emWinView can output every single layer and the composite view in a separate window.

25.1.1 Selecting a layer for drawing operations

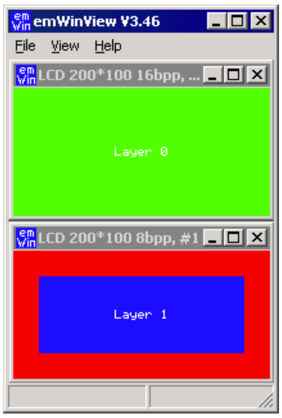
When drawing directly, per default layer 0 is used. Other layers can be selected by using the function `GUI_SelectLayer()`.

Example

The following example shows how to select a layer for drawing operations:

```
void MainTask(void) {
  GUI_Init();
  //
  // Draw something on default layer 0
  //
  GUI_SetBkColor(GUI_GREEN);
  GUI_Clear();
  GUI_DispStringHCenterAt("Layer 0", 100, 46);
  //
  // Draw something on layer 1
  //
  GUI_SelectLayer(1);
  GUI_SetBkColor(GUI_RED);
  GUI_Clear();
  GUI_SetColor(GUI_BLUE);
  GUI_FillRect(20, 20, 179, 79);
  GUI_SetColor(GUI_WHITE);
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_DispStringHCenterAt("Layer 1", 100, 46);
  while(1) {
    GUI_Delay(100);
  }
}
```

Screenshot of above example



25.1.2 Selecting a layer for a window

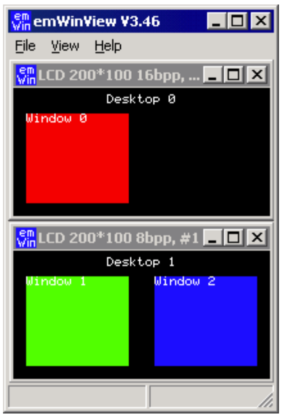
The Window Manager automatically keeps track of which window is located in which layer. This is done in a fairly easy way:
If the Window Manager is used, every layer has a top level (desktop) window. Any other window in this layer is visible only if it is a descendent of the according desktop window. Windows are connected to a certain layer depending on if they are a descendant of the layer’s desktop window.

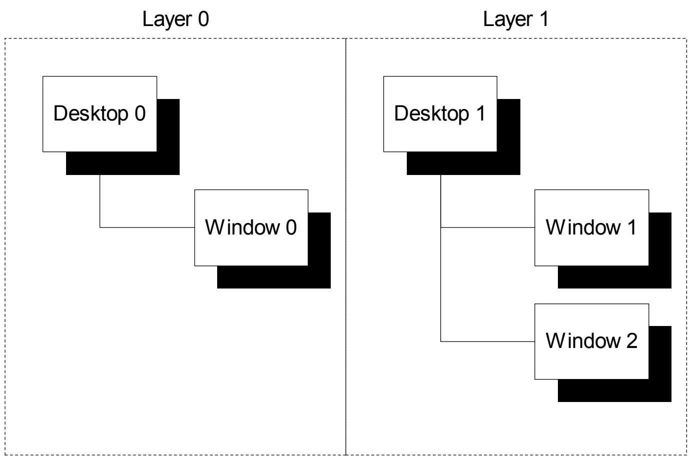
Example

The following example shows how to create 3 windows on 2 different desktop windows:

```
//
// Create 1 child window on destop 0
//
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                              WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0);
//
// Create 2 child windows on destop 1
//
hWin1 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                              WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin1, 0);
hWin2 = WM_CreateWindowAsChild(110, 20, 80, 70,
                              WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin2, 0);
```

The following table shows the screenshot and the window hierarchy of the above example:

<table>
<tr><th>Screenshot</th><th>Window hierarchy</th></tr>
<tr><td>


</td><td>


</td></tr>
</table>

Table 25.1: Selecting a window

25.1.2.1 Moving a window from one layer to an other

This can sometime be very desirable and can easily be accomplished: If a window is detached from its parent (The desktop window of one layer or any descendent of this desktop window) and attached to a window which lies in another layer, this window actually moves from one layer to another layer.

Example

The following example shows how to attach a window to a new parent window:

```
//
// Create 1 child window on destop 0
//
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                              WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0);
//
// Create 2 child windows on destop 1
//
hWin1 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                              WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin1, 0);
hWin2 = WM_CreateWindowAsChild(110, 20, 80, 70,
                              WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin2, 0);
GUI_Delay(1000);
//
// Detach window 2 from desktop 1 and attach it to desktop 0
//
WM_AttachWindow(hWin2, WM_GetDesktopWindowEx(0));
```


The following table shows the screenshot and the window hierarchy of the above example before attaching the window to the new parent:

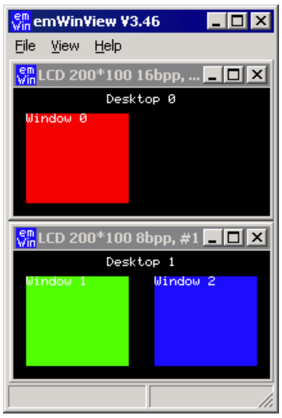
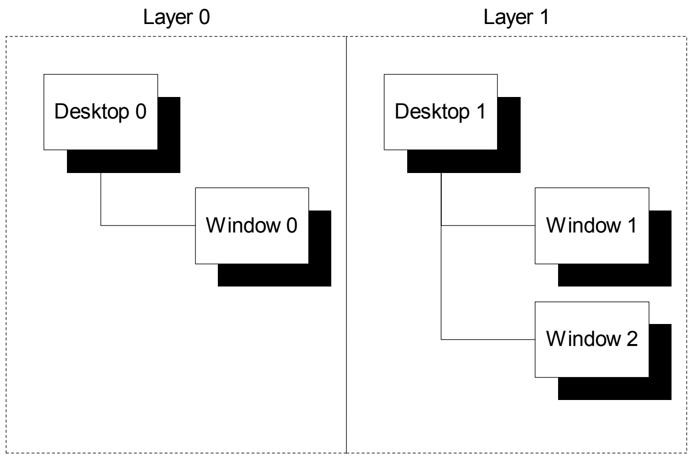
Screenshot	Window hierarchy
	

Table 25.2: Moving a window

The next table shows the screenshot and the window hierarchy of the above example after attaching the window to the new parent:

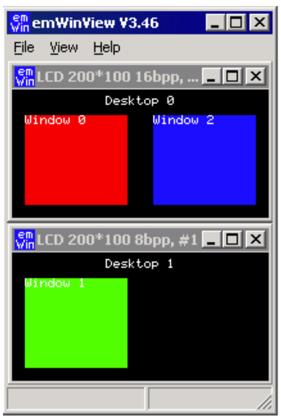
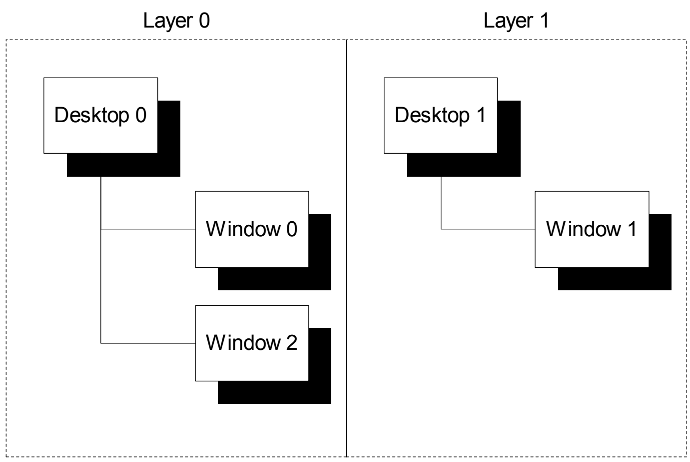
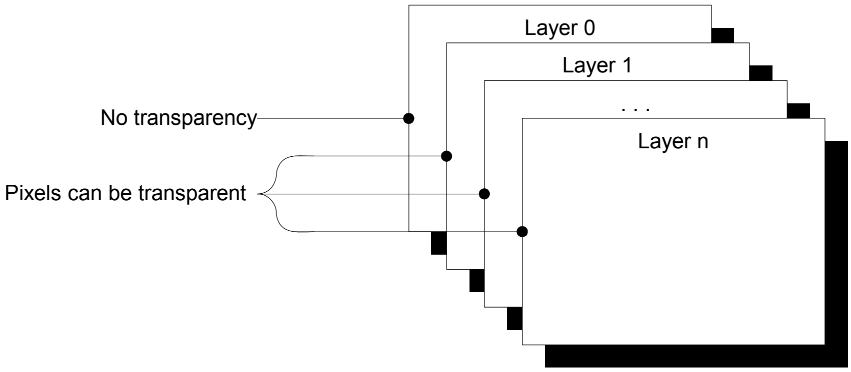
Screenshot	Window hierarchy
	

Table 25.3: Moving a window

25.2 Using MultiLayer support

emWin does not distinguish between multiple layers or multiple displays. When using multiple layers normally the size and the driver for each layer is the same. The viewer shows each layer in a separate window. The composite window of the viewer shows all layers; layers with higher index are on top of layers with lower index and can have transparent pixels:



25.2.1 Transparency

Transparency means that at the position of pixels with color index 0 in a layer > 0, the color of the background layer is visible. Since for all but layer 0 Index 0 means transparency, Index 0 can not be used to display colors. This also means that the color conversion should never yield 0 as best match for a color, since this would result in a transparent pixel. This means that only some fixed palette modes or a custom palette mode should be used and that you need to be careful when defining your own palette. You need to make sure that the color conversion (24 bit RGB -> Index) never yields 0 as result.

Fixed palette modes

The only available fixed palette modes including transparency (not Alpha Blending) are `GUICC_M1555I` and `GUICC_8666_1`. Details about fixed palette modes can be found in the chapter "Colors" on page 291.

Custom palette mode

If a custom palette should be used in a layer > 0, the first color should not be used from the color conversion routines. The following shows an example definition for a custom palette with 15 gray scales:

```
static const LCD_COLOR _aColors_16[] = {
  GUI_TRANSPARENT, 0x000000, 0x222222, 0x333333,
  0x444444, 0x555555, 0x666666, 0x777777,
  0x888888, 0x999999, 0xAAAAAA, 0xBBBBBB,
  0xCCCCCC, 0xDDDDDD, 0xEEEEEE, 0xFFFFFF
};

static const LCD_PHYSPALETTE _aPalette_16 = {
  16, _aColors_16

};
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  .
  .
  .
  //
  // Set user palette data (only required if no fixed palette is used)
  //
  LCD_SetLUTEx(1, _aPalette_16);
}
```

The description of the function `LCD_SetLUTEx()` can be found under "Custom palette mode" on page 312.

Example

The following example shows how to use transparency. It draws 3 color bars in layer 0. Layer 1 is filled with white and 3 transparent items are drawn.

```
GUI_SelectLayer(0);
GUI_SetColor(GUI_RED);
GUI_FillRect(0, 0, 199, 33);
GUI_SetColor(GUI_GREEN);
GUI_FillRect(0, 34, 199, 66);
GUI_SetColor(GUI_BLUE);
GUI_FillRect(0, 67, 199, 99);
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_WHITE);
GUI_Clear();
GUI_SetColor(GUI_BLACK);
GUI_DispStringHCenterAt("Layer 1", 100, 4);
GUI_SetColor(GUI_TRANSPARENT);
GUI_FillCircle(100, 50, 35);
GUI_FillRect(10, 10, 40, 90);
GUI_FillRect(160, 10, 190, 90);
```

Screenshots of the above example

The table below shows the contents of the separate layers and the composite view, as the result appears on the display:

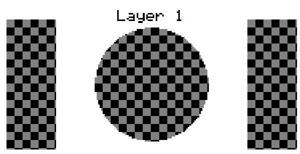
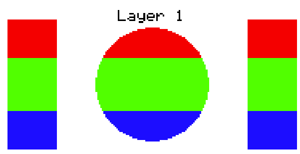
Layer 0	Layer 1	Display
		

Table 25.4: MultiLayer transparency

25.2.2 Alpha blending

Alpha blending is a method of combining two colors for transparency effects. Assumed 2 colors C_0 and C_1 should be combined with alpha blending A (a value between 0 and 1 where 0 means invisible and 1 means 100% visible) the resulting color C_r can be calculated as follows:

$$C_r = C_0 * (1 - A) + C_1 * A$$

Logical colors are handled internally as 32 bit values. The lower 24 bits are used for the color information and the alpha blending is managed in the upper 8 bits. An alpha value of 0x00 means opaque and 0xFF means completely transparent (invisible).

Different methods

There are 3 different methods of managing the alpha information:
- Layer alpha blending: On systems with layer alpha blending the alpha value is fixed to the layer and can be set with the function `LCD_SetAlphaEx()`.
- Lookup table (LUT) alpha blending: This kind of alpha blending uses the LUT for managing the alpha information.
- Pixel alpha blending: Each pixel of the layer which has to be combined with the background consists of alpha blending information.

Fixed palette modes

For LUT alpha blending the fixed palette modes 822216 and 84444 can be used. Pixel alpha blending is supported only in 32 bpp mode using the fixed palette mode 8888. For details about the fixed palette modes, refer to the chapter “Colors” on page 291.

Example

The following example shows how to use pixel alpha blending. It draws a circle in layer 0 and a yellow triangle build of horizontal lines with a vertical gradient of alpha values:


```
GUI_SetColor(GUI_BLUE);
GUI_FillCircle(100, 50, 49);
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_TRANSPARENT);
GUI_Clear();
for (i = 0; i < 100; i++) {
  U32 Alpha;
  Alpha = (i * 255 / 100) << 24;
  GUI_SetColor(GUI_YELLOW | Alpha);
  GUI_DrawHLine(i, 100 - i, 100 + i);
}
```

Screenshots of the above example

The table below shows the contents of the separate layers and the composite view, as the result appears on the display:

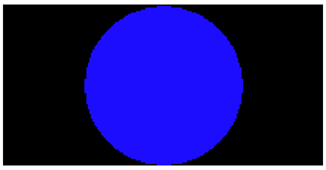
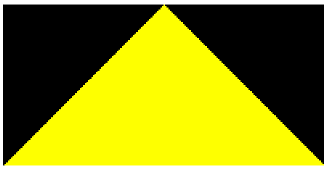
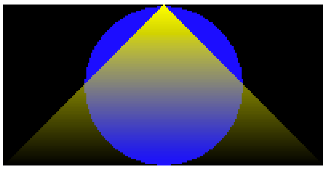
Layer 0	Layer 1	Display
		

Table 25.5: MultiLayer Alphablending

25.2.3 Hardware cursors

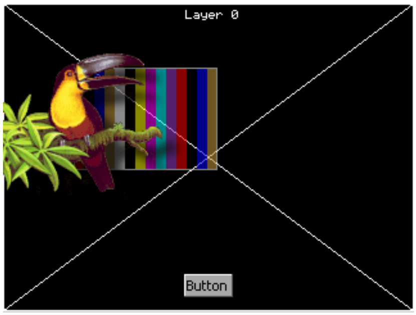
The term 'Hardware cursor' means the use of cursor images in a separate layer with a transparent background. If a hardware supports multiple layers and the ability of layer positioning emWin can be configured to use a separate layer for managing the cursor. The main advantages of this kind of cursor support are a better performance because only a few registers need to be changed on a movement and the ability of custom drawings in the cursor layer. For details about usage, refer to "GUI_AssignCursorLayer()" on page 965.

25.2.4 MultiLayer example

For information about a multi-layer example, see the chapter "Simulation" on page 51. Further, the `Sample` folder contains the following example which shows how to use multiple layer support:

- MULTILAYER_AlphaChromaMove.c

Screenshot of above example



25.2.5 Configuring MultiLayer support

LCD Configuration of the above MultiLayer example

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for first layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, // Display driver
                           GUICC_655,     // Color conversion
                           0, 0);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (0, 400,  234);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0xc00000); // Video RAM start address
  //
  // Set display driver and color conversion for second layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,  // Display driver
                           GUICC_86661,   // Color conversion
                           0, 1);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (1, 400,  234);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(1, (void *)0xc00000); // Video RAM start address
}
```

25.3 Using MultiDisplay support

Each display can be accessed with its own driver and with its own settings.

25.3.1 Enabling MultiDisplay support

To enable the MultiDisplay support you have to define the maximum number of layers in `GUIConf.h`:

```
#define GUI_NUM_LAYERS 2 /* Enables support for 2 displays/layers */
```

Further you have to create and configure a display driver device for each layer.

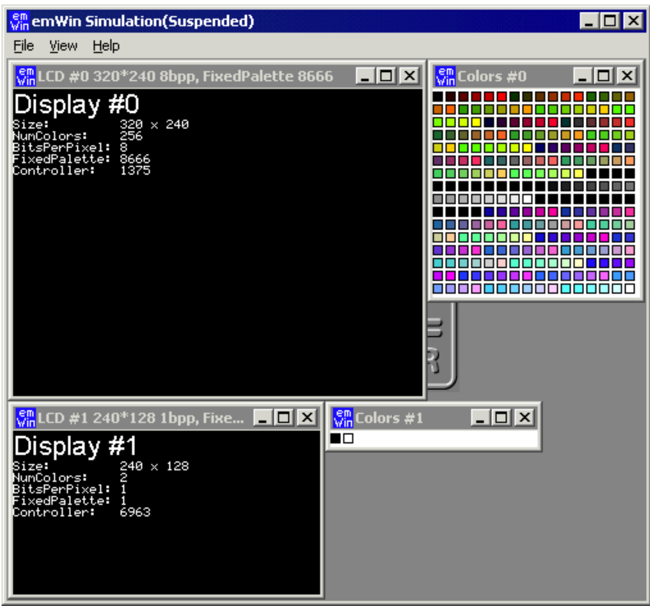
25.3.2 Run-time screen rotation

In some cases it may be necessary to change the display orientation at run-time. The MultiDisplay support allows to do this. In this case the file `LCDConf.c` should contain a display configuration for each required display orientation. Switching the display orientation then works as follows:

- Select the configuration with the required display orientation with `GUI_SelectLayer()`.
- If the rotation requires a reinitialization of the display controller the right driver function for reinitializing should be called. This is `LCD_L0_Init()` for layer 0 and `LCD_L0_x_Init()` for higher layers, where 'x' means the zero based index of the configuration.

25.3.3 MultiDisplay example

The example below shows a screenshot of the simulation with 2 displays. The fist display is a 8bpp color display with a size of 320 x 240 pixel. The driver is LCD13XX.c configured for an Epson S1D13705 LCD-controller. The second display is a 1bpp bw-display with a size of 240 x 128 pixels. The driver is LCDSlin.c configured for a Toshiba T6963 LCD-controller:



25.3.4 Configuring MultiDisplay support

Configuration of the above MultiDisplay example

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for first layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,  // Display driver
                           GUICC_8666,    // Color conversion
                           0, 0);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (0, 320,  240);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0xc00000); // Video RAM start address
  //
  // Set display driver and color conversion for second layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_1,  // Display driver
                           GUICC_1,       // Color conversion
                           0, 1);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (1, 240,  128);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(1, (void *)0x800000); // Video RAM start address
}
```


25.4 Using SoftLayers

In case multiple hardware layers are not supported, emWin offers the possibility to make use of software layers. The advantage of softlayers is they can be used on any target hardware meeting the memory requirements. The disadvantage is additional CPU load to render the layers. Rendering is done automatically by executing the GUI. Alternatively the function `GUI_SOFTLAYER_Refresh()` can be called to immediately refresh the layers.
emWin SoftLayers are highly optimized. Layers are refreshed only if there is at least one dirty area. Drawing operations are tracked automatically to create/extend according dirty areas. A refresh is processed by dividing dirty areas in subrectangles which affect the same layers for faster color calculation. This way processing all layers unnecessarily for all pixels is avoided.
If there are dirty areas, pixel data is processed from the bottom to the top starting with the top opaque layer. Layers below the top opaque layer would not have any impact on the result.

Mixing example (opaque and non-overlapping layers)

Assuming there is a configuration including 4 SoftLayers. Drawing operations have been done, so a dirty area exists. Layer 1 is opaque, layer 2 does not overlap the dirty area and layer 3 contains semi-transparent pixels. In this case the SoftLayer logic would ignore layer 0 and layer 2, so the only thing to do would be mixing the colors from layer 1 and 3.

Mixing example (composite color and several semi-transparent layers)

Assuming there is a configuration including 4 SoftLayers in which the layers contain either transparency, semi-transparency or are not affected by the dirty area at all, the composite color is used as "opaque layer". In this case the composite color is mixed with the content of layer 0. The result would be mixed with the content of layer 1. In turn the result would be mixed with the content of layer 2. And so on...

25.4.1 Using SoftLayers within a simulation environment

In a simulation environment SoftLayers have to be set up differently from HardLayers. Below information should be a help for configuring the simulation properly for the use of SoftLayers. The according function descriptions can be found in the chapter “Simulation” on page 51.

Composite color

SoftLayers consist of their own composite color which can be set using the function `GUI_SOFTLAYER_Enable()` or `GUI_SOFTLAYER_SetCompositeColor()`. The function `GUI_SIM_SetCompositeColor()` does not have an effect with SoftLayers.

Composite size

The function `SIM_GUI_SetCompositeSize()` can be used for setting the size of the composite view. Calling this function is required for SoftLayer use.

Transparency mode

In order to benefit from transparency effects, Layers can be configured for a certain transparency mode using the function `SIM_GUI_SetTransMode()`. Transparency can be implemented either via Zero-Transparency or via Pixel-Alpha.

SIMConf.c example

```
void SIM_X_Config() {
  SIM_GUI_SetCompositeSize(480, 272);  // Set size of composite window
  SIM_GUI_SetTransMode(1, GUI_TRANSMODE_PIXELALPHA);
  SIM_GUI_SetTransMode(2, GUI_TRANSMODE_PIXELALPHA);
  SIM_GUI_SetTransMode(3, GUI_TRANSMODE_PIXELALPHA);
}
```


25.4.2 Memory requirements

emWin SoftLayers require storing additional information in RAM which can be subdivided in display related memory and layer related memory. The memory is taken from the memory pool which is assigned to emWin using `GUI_ALLOC_AssignMemory()` in function `GUI_X_Config()` (`GUIConf.c`).

Required display related memory

Depending on the display size and color depth, SoftLayers require storing data for the following items:

- SoftLayer driver context.
- 32bpp buffer with the size of the display width.
- Frame buffer with the size and color depth of the display.

The required display related memory can be calculated using the following formula:

```
ReqMem  = 68 Bytes + xSizeDisp * 4 + xSizeDisp * ySizeDisp * BytesPerPixelDisp
```

Required layer related memory

Depending on the SoftLayer configuration, SoftLayers require additional memory to store one complete frame for each layer at a color depth of 32bpp.
The required display related memory can be calculated using the following formula:

```
ReqMem += xSize0 * ySize0 * 4 + xSize1 * ySize1 * 4 + ... (and so on)
```

Explanation of terms

Term	Explanation
68 Bytes	Required for context information in the SoftLayer driver.
xSizeDisp	X-size of the display.
ySizeDisp	Y-size of the display.
BytesPerPixelDisp	Number of bytes per pixel used by the display.
xSize0	X-size of layer 0.
ySize0	Y-size of layer 0.

Table 25.6: Explanation of terms used to calculate SoftLayer memory requirments

25.4.3 Configuring SoftLayers

Softlayers need to be configured different from hardware layers. In order to set up the desired layers a data structure of type `GUI_SOFTLAYER_CONFIG` needs to be filled and passed to the function `GUI_SOFTLAYER_Enable()`. SoftLayers do not require each layer to have a separate driver device. There is an internal SoftLayer driver which is automatically used for all layers.

LCDConf.c

```
void LCD_X_Config(void) {
  GUI_SOFTLAYER_CONFIG aConfig[] = {
    {   0,    0, 480, 272, 1 },
    {   0,    0, 120, 108, 1 },
    {   0,    0, 120,  74, 1 },
    {  30,   30, 420,  35, 1 },
  };

  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx    (0, XSIZE_PHYS,   YSIZE_PHYS);
  LCD_SetVSizeEx   (0, VXSIZE_PHYS,  VYSIZE_PHYS);
  LCD_SetVRAMAddrEx(0, (void *)VRAM_ADDR);
  //
  // SoftLayer activation after existing single layer configuration
  //
  GUI_SOFTLAYER_Enable(aConfig, GUI_COUNTOF(aConfig), GUI_DARKBLUE);
}
```

25.5 MultiLayer API

The table below lists the available MultiLayer related routines in alphabetical order. Detailed descriptions follow:

Routine	Description
General MultiLayer functions	
GUI_AssignCursorLayer()	Assigns a layer to be used to manage a hardware cursor.
GUI_GetLayerPosEx()	Gets the position of the given layer.
GUI_SelectLayer()	Selects a layer/display for output operations.
GUI_SetLayerAlphaEx()	Sets the layer alpha blending.
GUI_SetLayerPosEx()	Sets the position of the given layer.
GUI_SetLayerSizeEx()	Sets the size of the given layer.
GUI_SetLayerVisEx()	Sets the visibility of the given layer.
LCD_GetNumLayers()	Returns the number of layers.
SoftLayer specific functions	
GUI_SOFTLAYER_Enable()	Enables and configures SoftLayers.
GUI_SOFTLAYER_Refresh()	Refreshes layers by rerendering the dirty area.
GUI_SOFTLAYER_SetCompositeColor()	Sets the composite color.
GUI_SOFTLAYER_MULTIBUF_Enable()	Enables the automatic use of Multiple Buffering with SoftLayers.

Table 25.7: MultiLayer API list

GUI_AssignCursorLayer()

Description

The function assigns a layer to be used as cursor layer.

Prototype

```
void GUI_AssignCursorLayer(unsigned Index, unsigned CursorLayer);
```

Parameter	Description
Index	Layer index.
CursorLayer	Layer to be used to manage the cursor.

Table 25.8: GUI_AssignCursorLayer() parameter list

Additional information

Using a hardware cursor means a layer is used as cursor layer. Contrary to the default cursor handling, where the cursor is drawn in the same video memory area as all other items, a hardware cursor is drawn in a separate layer. In this case emWin makes sure the background color of the hardware cursor layer is set to transparency and the selected cursor will be drawn into the layer.
Whereas the default cursor management requires more or less calculation time to draw the cursor and to manage the background, moving a hardware cursor requires only the modification of a few registers.
Note that using this function requires that the display driver supports layer positioning.

GUI_GetLayerPosEx()

Description

Returns the X- and Y-position of the given layer.

Prototype

```
void GUI_GetLayerPosEx(unsigned Index, int * pxPos, int * pyPos);
```

Parameter	Description
Index	Layer index.
pxPos	Pointer to an integer to be used to return the X position of the given layer.
pyPos	Pointer to an integer to be used to return the Y position of the given layer.

Table 25.9: GUI_GetLayerPosEx() parameter list

Additional information

To be able to use this function the hardware and the used display driver need to support layer positioning. If the driver does not support this feature the function returns immediately.

GUI_SelectLayer()

Description

Selects a layer for drawing operations.

Prototype

```
unsigned int GUI_SelectLayer(unsigned int Index);
```

Parameter	Description
Index	Layer index.

Table 25.10: GUI_SelectLayer() parameter list

Return value

Index of the previously selected layer.

GUI_SetLayerAlphaEx()

Description

Sets the alpha blending of the given layer.

Prototype

```
int GUI_SetLayerAlphaEx(unsigned Index, int Alpha);
```

Parameter	Description
Index	Layer index.
Alpha	Alpha blending value of the given layer.

Table 25.11: GUI_SetLayerAlphaEx() parameter list

Return value

0 on success, 1 on error.

Additional information

To be able to use this function the hardware and the used display driver need to support layer alpha blending. If the driver does not support this feature the function returns immediately.
The usable range of alpha values depends on the hardware. In many cases the range of alpha values is limited, for example 0 - 0x3f. emWin does not know something about limitations and passes the given value to the driver. It is the responsibility of the application to make sure that the given value is in a legal range.

GUI_SetLayerPosEx()

Description

Sets the X- and Y-position of the given layer.

Prototype

```
void GUI_GetLayerPosEx(unsigned Index, int xPos, int yPos);
```

Parameter	Description
Index	Layer index.
xPos	New X position of the given layer.
yPos	New Y position of the given layer.

Table 25.12: GUI_SetLayerPosEx() parameter list

Additional information

To be able to use this function the hardware and the used display driver need to support layer positioning. If the driver does not support this feature the function returns immediately.

GUI_SetLayerSizeEx()

Description

Sets the X- and Y-size of the given layer.

Prototype

```
int GUI_SetLayerSizeEx(unsigned Index, int xSize, int ySize);
```

Parameter	Description
Index	Layer index.
xSize	New horizontal size in pixels of the given layer.
ySize	New vertical size in pixels of the given layer.

Table 25.13: GUI_SetLayerSizeEx() parameter list

Additional information

To be able to use this function the hardware and the used display driver need to support layer sizing. If the driver does not support this feature the function returns immediately.

GUI_SetLayerVisEx()

Description

Sets the visibility of the given layer.

Prototype

```
int GUI_SetLayerVisEx(unsigned Index, int OnOff);
```

Parameter	Description
Index	Layer index.
OnOff	1 if layer should be visible, 0 for invisible.

Table 25.14: GUI_SetLayerVisEx() parameter list

Additional information

To be able to use this function the hardware and the used display driver need to support this feature. If the driver does not support this feature the function returns immediately.

LCD_GetNumLayers()

Description

Returns the number of layers configured in your configuration.

Prototype

```
int LCD_GetNumLayers(void);
```

Return value

Number of layers configured in your configuration.

25.5.1 SoftLayer API

GUI_SOFTLAYER_Enable()

Description

Enables and configures SoftLayers. This function may be called only from within the function `LCD_X_Config()`. The function `LCD_X_Config()` is called automatically from `GUI_Init()`.

Prototype

```
int GUI_SOFTLAYER_Enable(GUI_SOFTLAYER_CONFIG * pConfig, int NumLayers,
GUI_COLOR CompositeColor);
```

Parameter	Description
pConfig	Pointer to the GUI_SOFTLAYER_CONFIG data structure is explained below.
NumLayers	Number of layers to create using the config data.
CompositeColor	The composite color is used in case certain areas do not contain opaque colors.

Table 25.15: GUI_SOFTLAYER_Enable() parameter list

Elements of structure GUI_SOFTLAYER_CONFIG

Data type	Element	Description
int	xPos	X-position
int	yPos	Y-position
int	xSize	X-size
int	ySize	Y-size
int	Visible	1 == visible, 0 == not visible

Table 25.16: GUI_SOFTLAYER_CONFIG element list

Return value

0 on success, 1 on error.

GUI_SOFTLAYER_Refresh()

Description

Refreshes layers by rerendering the dirty area. This function is called from the function `GUI_Exec1()` if SoftLayers are enabled. After a refresh was performed, the dirty area gets cleared.

Prototype

```
int GUI_SOFTLAYER_Refresh(void);
```

Return value

0 if nothing was done, 1 if something was done, 2 on error.

GUI_SOFTLAYER_SetCompositeColor()

Description

Sets the composite color.

Prototype

```
void GUI_SOFTLAYER_SetCompositeColor(U32 Color);
```

Parameter	Description
Color	Color to be set as composite color.

Table 25.17: GUI_SOFTLAYER_SetCompositeColor() parameter list

Additional information

The function SIM_GUI_SetCompositeColor() does not have an effect with SoftLayers.

GUI_SOFTLAYER_MULTIBUF_Enable()

Description

Enables the automatic use of Multiple Buffering with SoftLayers. If Multiple Buffering is enabled, the function GUI_SOFTLAYER_Refresh() automatically calls the function GUI_MULTIBUF_BeginEx() and GUI_MULTIBUF_EndEx().

Prototype

```
int GUI_SOFTLAYER_MULTIBUF_Enable(int OnOff);
```

Parameter	Description
OnOff	0 == Disable MultiBuffering support. 1, Enable MultiBuffering support.

Table 25.18: GUI_SOFTLAYER_MULTIBUF_Enable() parameter list

Return value

The function returns the previous setting.
0 if MultiBuffering has not been used, 1 if MultiBuffering has already been used.

Chapter 26

Pointer Input Devices

emWin provides support for pointer-input-devices. Pointer input devices can be touch-screen, mouse or joystick. The basic emWin package includes a driver for analog touch-screens, a PS2 mouse driver, as well as an example joystick driver. Other types of touch-panel and mouse devices can also be used with the appropriate drivers.
The software for input devices is located in the subdirectory `GUI\Core`.

26.1 Description

Pointer input devices are devices such as mice, touch-screens and joysticks. Multiple pointer input devices can be used in a single application to enable simultaneous mouse/touch-screen/joystick use. Basically all a PID driver does is calling the routine GUI_PID_StoreState() whenever an event (such as a moved mouse, or a pressed touch screen) has been detected.
PID events are stored in a FIFO which is processed by the Window Manager. If the Window Manager is not used (respectively deactivated), the application is responsible for reacting on PID events.

26.2 Pointer input device API

The table below lists the pointer input device routines in alphabetical order. Detailed descriptions follow.
Note: This API is used by the PID-driver; if you use a PID-driver shipped with emWin, your code does not need to call these routines.

Routine	Description
GUI_PID_GetCurrentState()	Returns the most recently stored state from the PID.
GUI_PID_GetState()	Returns the state of the PID.
GUI_PID_IsEmpty()	Returns if the PID buffer is empty.
GUI_PID_IsPressed()	Returns if the most recent state of the PID is pressed.
GUI_PID_StoreState()	Stores the current state of the PID.

Table 26.1: PID API list

Data structure

The structure of type GUI_PID_STATE referenced by the parameter pState is filled by the routine with the current values. The structure is defined as follows:

```
typedef struct {
  int x, y;
  U8  Pressed;
  U8  Layer;
} GUI_PID_STATE;
```

Elements of structure GUI_PID_STATE

Data type	Element	Description
int	x	X position of pointer input device.
int	y	Y position of pointer input device.
U8	Pressed	If using a touch screen this value can be 0 (unpressed) or 1 (pressed). If using a mouse bit 0 is used for the pressed state of the left button and bit 1 for the right button. The bits are 1 if the button is pressed and 0 if not.
U8	Layer	Describes the layer from which the PID state has been received

Table 26.2: GUI_PID_STATE element list

GUI_PID_GetCurrentState()

Description

Fills the given GUI_PID_STATE structure with the most recently stored PID state.

Prototype

```
void GUI_PID_GetCurrentState(GUI_PID_STATE * pState);
```

Parameter	Description
pState	Pointer to a GUI_PID_STATE structure.

Table 26.3: GUI_PID_GetCurrentState() parameter list

Additional information

This function performs a non-destructive read on the PID FIFO.

GUI_PID_GetState()

Description

Fills the given GUI_PID_STATE structure with the current state information and returns if the input device is currently pressed.

Prototype

```
int GUI_PID_GetState(GUI_PID_STATE * pState);
```

Parameter	Description
pState	Pointer to a GUI_PID_STATE structure.

Table 26.4: GUI_PID_GetState() parameter list

Additional information

This function does a destructive read on the PID FIFO:
If the FIFO contains unread values, it reads and eliminates the first value in the FIFO.
If the FIFO is empty, it returns the last value written to it. If no value has ever been written into the PID FIFO, all values in pState are set to 0.

Return value

1 if input device is currently pressed; 0 if not pressed.

Example

```
GUI_PID_STATE State;
GUI_PID_GetState(&State);
```

GUI_PID_IsEmpty()

Description

Returns if the PID buffer is empty.

Prototype

```
int GUI_PID_IsEmpty(void);
```

Return value

1, if the PID buffer is empty.
0, if entries were found in the PID buffer.

GUI_PID_IsPressed()

Description

Returns if the most recent state of the PID is pressed.

Prototype

```
int GUI_PID_IsPressed(void);
```

Additional information

This function does not modify the PID FIFO.

Return value

1 if input device is currently pressed; 0 if not pressed.

GUI_PID_StoreState()

Description

Stores the current state of the pointer input device.

Prototype

```
void GUI_PID_StoreState(const GUI_PID_STATE * pState);
```

Parameter	Description
pState	Pointer to a structure of type GUI_PID_STATE.

Table 26.5: GUI_PID_StoreState() parameter list

Additional information

This function can be used from an interrupt service routine.
The PID input manager of emWin contains a FIFO buffer which is able to hold up to 5 PID events per default. If a different size is required this value can be changed. Details can be found in the section "Advanced GUI configuration options" on page 1213.

26.3 Mouse driver

Mouse support consists of two "layers": a generic layer and a mouse driver layer. Generic routines refer to those functions which always exist, no matter what type of mouse driver you use. The available mouse driver routines, on the other hand, will call the appropriate generic routines as necessary, and may only be used with the PS2 mouse driver supplied with emWin. If you write your own driver, it is responsible for calling the generic routines.
The generic mouse routines will in turn call the corresponding PID routines.

26.3.1 Generic mouse driver API

The table below lists the generic mouse routines in alphabetical order. These functions may be used with any type of mouse driver. Detailed descriptions follow.

Routine	Description
GUI_MOUSE_GetState()	Return the current state of the mouse.
GUI_MOUSE_StoreState()	Store the current state of the mouse.

Table 26.6: Generic mouse driver API list

GUI_MOUSE_GetState()

Description

Returns the current state of the mouse.

Prototype

```
int GUI_MOUSE_GetState(GUI_PID_STATE * pState);
```

Parameter	Description
pState	Pointer to a structure of type GUI_PID_STATE.

Table 26.7: GUI_MOUSE_GetState() parameter list

Return value

1 if mouse is currently pressed; 0 if not pressed.

Additional information

This function will call `GUI_PID_GetState()`.

GUI_MOUSE_StoreState()

Description

Stores the current state of the mouse.

Prototype

```
void GUI_MOUSE_StoreState(const GUI_PID_STATE * pState);
```

Parameter	Description
pState	Pointer to a structure of type GUI_PID_STATE.

Table 26.8: GUI_MOUSE_StoreState() parameter list

Additional information

This function will call `GUI_PID_StoreState()`.
This function can be used from an interrupt service routine.

Example

```
GUI_PID_STATE State;
State.x = _MousepositionX; /* Screen position in X of mouse device */
State.y = _MousepositionY; /* Screen position in Y of mouse device */
State.Pressed = 0;
if (_LeftButtonPressed) {
  State.Pressed |= 1;       /* Set bit 0 if left button is pressed */
}
if (_RightButtonPressed) {
  State.Pressed |= 2;       /* Set bit 1 if right button is pressed */
}
GUI_MOUSE_StoreState(&State);
```

26.3.2 PS2 mouse driver

The driver supports any type of PS2 mouse.

26.3.2.1 Using the PS2 mouse driver

The driver is very easy to use. In the startup code, the init function `GUI_MOUSE_DRIVER_PS2_Init()` should be called. The application should somehow notice when a byte is received from the mouse. When this happens, the function `GUI_MOUSE_DRIVER_PS2_OnRx()` should be called and the byte received passed as parameter. The driver in turn then calls `GUI_PID_StoreState` as required. The reception of the byte is typically handled in an interrupt service routine.

An example ISR could look as follows: (Note that this is of course different for different systems)

```
void interrupt OnRx(void) {
  char Data;
  Data  = UART_REG;                 // Read data from the hardware
  GUI_MOUSE_DRIVER_PS2_OnRx(Data);  // Pass it on to the driver
}
```

26.3.2.2 PS2 mouse driver API

The table below lists the available mouse driver routines in alphabetical order.

Routine	Description
GUI_MOUSE_DRIVER_PS2_Init()	Initialize the mouse driver.
GUI_MOUSE_DRIVER_PS2_OnRx()	Called form receive interrupt routines.

Table 26.9: PS2 mouse driver API list

GUI_MOUSE_DRIVER_PS2_Init()

Description

Initializes the mouse driver.

Prototype

```
void GUI_MOUSE_DRIVER_PS2_Init(void);
```

GUI_MOUSE_DRIVER_PS2_OnRx()

Description

Must be called from receive interrupt routines.

Prototype

```
void GUI_MOUSE_DRIVER_PS2_OnRx(unsigned char Data);
```

Parameter	Description
Data	Byte of data received by ISR.

Table 26.10: GUI_MOUSE_DRIVER_PS2_OnRx()

Additional information

The PS2 mouse driver is a serial driver, meaning it receives 1 byte at a time.
You need to ensure that this function is called from your receive interrupt routine every time a byte (1 character) is received.

26.4 Touch screen driver

A touch screen driver will typically simply call `GUI_PID_StoreState()` as described earlier. Any type of touch screen can be supported this way. It is the responsibility of the user to write the driver code (which is usually fairly simple).
The most common way of interfacing a touch screen is the 4-pin analog interface, for which a driver is supplied.

26.4.1 Generic touch screen driver API

The generic touch screen API is used with any type of driver (analog, digital, etc.). A driver calls the appropriate routines as necessary. If you write your own driver, it has to call the generic routines.
The table below lists the generic touch-screen routines in alphabetical order. These functions may be used with any type of touch-screen driver. Detailed descriptions follow.

Routine	Description
GUI_TOUCH_GetState()	Return the current state of the touch-screen.
GUI_TOUCH_StoreState()	Store the current state of the touch-screen using X- and Y-coordinates.
GUI_TOUCH_StoreStateEx()	Store the current state of the touch-screen.

Table 26.11: Generic touch screen driver API list

GUI_TOUCH_GetState()

Description

Returns the current state of the touch-screen.

Prototype

`int GUI_TOUCH_GetState(GUI_PID_STATE * pState);`

Parameter	Description
pState	Pointer to a structure of type GUI_PID_STATE.

Table 26.12: GUI_TOUCH_GetState() parameter list

Return value

1 if touch-screen is currently pressed; 0 if not pressed.

GUI_TOUCH_StoreState()

Description

Stores the current state of the touch screen using X- and Y-coordinates as parameters.

Prototype

`void GUI_TOUCH_StoreState(int x, int y);`

Parameter	Description
x	X-position.
y	Y-position.

Table 26.13: GUI_TOUCH_StoreState() parameter list

Additional information

This function can be used from an interrupt service routine. It calls the function GUI_PID_StoreState(). It is assumed that the touch panel is not pressed, if this function is called with negative values.
A detailed example of a touch handling routine can be found in
`Sample\GUI_X\GUI_X_Touch_StoreState.c`.

Example

```
int x, y;
if (_TouchIsPressed) {
  x = _TouchPositionX;  /* Current position in X of touch device */
  y = _TouchPositionY;  /* Current position in Y of touch device */
} else {
  x = y = -1;           /* Use -1 if touch is not pressed */
}
GUI_TOUCH_StoreState(x, y);
```

GUI_TOUCH_StoreStateEx()

Description

Stores the current state of the touch screen.

Prototype

`void GUI_TOUCH_StoreStateEx(const GUI_PID_STATE * pState);`

Parameter	Description
pState	Pointer to a structure of type GUI_PID_STATE.

Table 26.14: GUI_TOUCH_StoreStateEx() parameter list

Additional information

This function will call `GUI_PID_StoreState()`.
A detailed example of a touch handling routine can be found in
`Sample\GUI_X\GUI_X_Touch_StoreState.c`.

Example

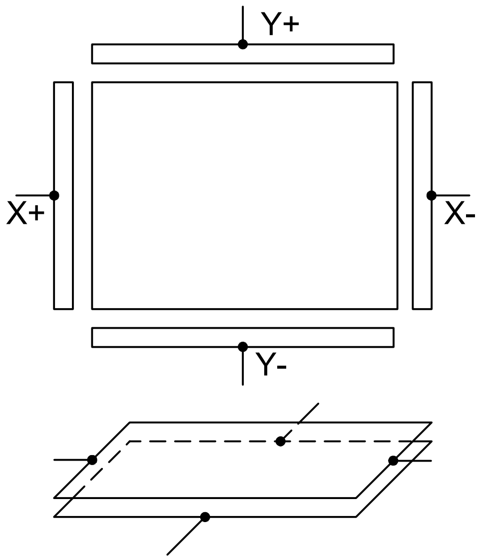
```
GUI_PID_STATE State;
State.x = _TouchPositionX;
State.y = _TouchPositionY;
if (_TouchIsPressed) {
  State.Pressed = 1;
} else {
  State.Pressed = 0;
}
GUI_TOUCH_StoreStateEx(&State);
```

26.4.2 The analog touch screen driver

The emWin touch-screen driver handles analog input (from an 8-bit or better A/D converter), debouncing and calibration of the touch-screen.
The touch-screen driver continuously monitors and updates the touch-panel through the use of the function `GUI_TOUCH_Exec()`, which calls the appropriate generic touch-screen API routines when it recognizes that an action has been performed or something has changed.

How an analog touch screen works

The touch panel consists of 2 thin conducting layers of glass, normally insulated from each other. If the user presses the touch panel, the two layers are connected at that point. If a voltage is applied to the Y-layer, when pressed, a voltage can be measured at the X+/X-terminals. This voltage depends on the touch position. The same thing holds true the other way round. If a voltage is applied to the X-layer, when pressed, a voltage can be measured at the Y+/Y-terminals.



26.4.2.1 Setting up the analog touch screen

Putting a touch panel into operation should be done in the following steps:
- Implementing the hardware routines
- Implementing regular calls to `GUI_TOUCH_Exec()`
- Verifying proper operation with the oscilloscope
- Using example to determine calibration values
- Adding a call of `GUI_TOUCH_Calibrate()` to the initialization routine `LCD_X_Config()` using the determined values

The following shows a detailed description of each step.

Implementing the hardware routines

The first step of implementing a touch screen should be filling the hardware routines with code. These routines are:

```
GUI_TOUCH_X_ActivateX(), GUI_TOUCH_X_ActivateY()
GUI_TOUCH_X_MeasureX(),  GUI_TOUCH_X_MeasureY()
```

A module `GUI_TOUCH_X.c` containing the empty routines is located in the folder `Sample\GUI_X`. You can use this module as a starting point.
The activate routines should prepare the measurement by switching on the measurement voltage. `GUI_TOUCH_X_ActivateX()` for example should prepare the measurement in Y by switching on the measurement voltage in X. Further it should switch of the voltage in Y and disable the measurement in X.

The measurement routines should return the measurement result of a A/D converter. Later in this chapter you will find an example implementation of the hardware routines.

Implementing regular calls to GUI_TOUCH_Exec()

The second step of implementing a touch screen is to make sure, that the function `GUI_TOUCH_Exec()` will be called in regular intervals. The application should call it about 100 times/second. If a real-time operating system is used, the easiest way to make sure this function is called is to create a separate task. When not using a multitasking system, an interrupt service routine may do the job. The function `GUI_TOUCH_Exec()` measures x- and y-axis in turns. So complete measurements are done once both axes were measured.

Verifying proper operation with the oscilloscope

After implementing the call of `GUI_TOUCH_Exec()` make sure the hardware works. The easiest way to do this is to measure the supply and measurement voltages of the touch panel with a oscilloscope. The following table shows a typical result. The first column shows the supply voltage of an axis, the second column shows the result of measuring the measurement voltage when pressing in the middle of the touch panel.

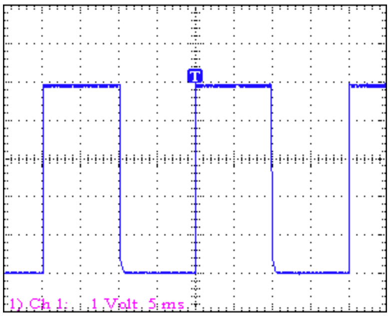
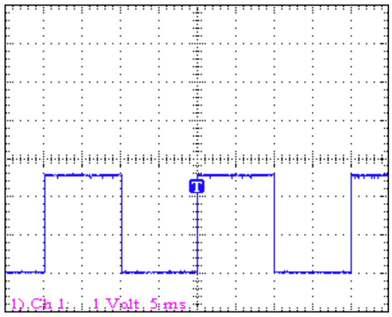
Supply voltage	Measurement voltage
	

Table 26.15: Touch screen voltage

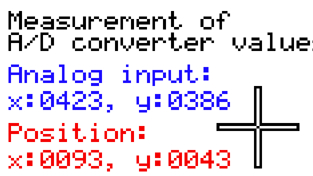
Use example to determine calibration values

The third step is to get the minimum and maximum values of the A/D converter. emWin needs this values to convert the measurement result to the touch position in pixels. These 4 values are:

Value	How to get them
GUI_TOUCH_AD_TOP	Press the touch at the top and write down the analog input value in Y.
GUI_TOUCH_AD_BOTTOM	Press the touch at the bottom and write down the analog input value in Y.
GUI_TOUCH_AD_LEFT	Press the touch at the left and write down the analog input value in X.
GUI_TOUCH_AD_RIGHT	Press the touch at the right and write down the analog input value in X.

Table 26.16: A/D converter value symbols

The example folder of emWin contains a small program which can be used to get these values from your touch panel. It is located in the folder `Sample\Tutorial` and its name is `TOUCH_Sample.c`. Run this example on your hardware. The output should be similar to the screenshot at the right side.



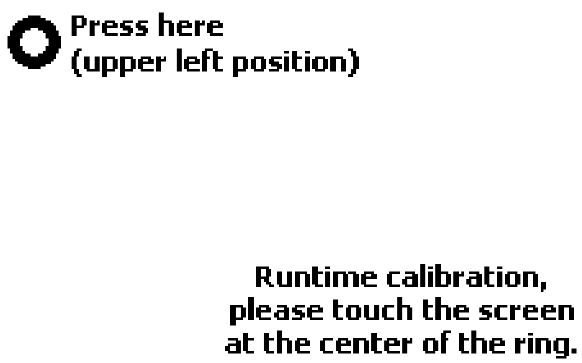
Use GUI_TOUCH_Calibrate() with the above values

The last step is adding a call to `GUI_TOUCH_Calibrate()` using the calibration values. The recommended location for calibrating the touch screen is the initialization routine `LCD_X_Config()` which is located in `LCDConf.c`. similar to following example:

```
#define GUI_TOUCH_AD_TOP     877
#define GUI_TOUCH_AD_BOTTOM  273
#define GUI_TOUCH_AD_LEFT    232
#define GUI_TOUCH_AD_RIGHT   918
.
.
.
void LCD_X_Config(void) {
  //
  // Initialize display driver
  //
  .
  .
  .
  //
  // Set orientation of touch screen (only required when using
  //
  TouchOrientation = (GUI_MIRROR_X * LCD_GetMirrorX()) |
                     (GUI_MIRROR_Y * LCD_GetMirrorY()) |
                     (GUI_SWAP_XY  * LCD_GetSwapXY()) ;
  GUI_TOUCH_SetOrientation(TouchOrientation);
  //
  // Calibrate touch screen
  //
  GUI_TOUCH_Calibrate(GUI_COORD_X, 0, 240, GUI_TOUCH_AD_TOP , GUI_TOUCH_AD_BOTTOM);
  GUI_TOUCH_Calibrate(GUI_COORD_Y, 0, 320, GUI_TOUCH_AD_LEFT, GUI_TOUCH_AD_RIGHT);
}
```

26.4.2.2 Runtime calibration

In practice the exact values for the configuration file can be determined only for one touch panel. Because there are small differences between the parts of a series it could be very needful to calibrate each device at run-time. This can be done by using the function `GUI_TOUCH_Calibrate()`. The `Sample` folder contains the example `TOUCH_Calibrate.c` which shows, how a touch screen can be calibrated at run time:



26.4.2.3 Hardware routines

The following four hardware-dependent functions need to be added to your project if you use the driver supplied with emWin, as they are called by `GUI_TOUCH_Exec()` when polling the touch-panel. A suggested place is in the file `GUI_X.c`. These functions are as follows:

Routine	Description
`GUI_TOUCH_X_ActivateX()`	Prepares measurement for Y-axis.
`GUI_TOUCH_X_ActivateY()`	Prepares measurement for X-axis.
`GUI_TOUCH_X_MeasureX()`	Returns the X-result of the A/D converter.
`GUI_TOUCH_X_MeasureY()`	Returns the Y-result of the A/D converter.

Table 26.17: Hardware routines

GUI_TOUCH_X_ActivateX()

GUI_TOUCH_X_ActivateY()

Description

These routines are called from `GUI_TOUCH_Exec()` to activate the measurement of the X- and the Y-axes. `GUI_TOUCH_X_ActivateX()` switches on the measurement voltage to the X-axis; `GUI_TOUCH_X_ActivateY()` switches on the voltage to the Y-axis. Switching on the voltage in X means the value for the Y-axis can be measured and vice versa.

Prototypes

```
void GUI_TOUCH_X_ActivateX(void);
void GUI_TOUCH_X_ActivateY(void);
```

GUI_TOUCH_X_MeasureX()

GUI_TOUCH_X_MeasureY()

Description

These routines are called from `GUI_TOUCH_Exec()` to return the measurement values from the A/D converter for the X- and the Y-axes.

Prototypes

```
int GUI_TOUCH_X_MeasureX(void);
int GUI_TOUCH_X_MeasureY(void);
```

Example implementation

The following shows an example implementation of the touch hardware routines for a Renesas M16C/80 controller:

```
void GUI_TOUCH_X_ActivateX(void) {
  U8 Data;
  asm("fclr i");                  /* Disable interrupts            */
  Data  = P10;                    /* Read port data                */
  Data |=   (1 << 2) | (1 << 3);  /* Switch on  power in X
                                     and  enable measurement in Y  */
  Data &= ~((1 << 4) | (1 << 5)); /* Switch off power in Y
                                     and disable measurement in X  */
  P10   = Data;                   /* Write port data               */
  asm("fset i");                  /* Enable interrupts             */
}

void GUI_TOUCH_X_ActivateY(void) {
  U8 Data;
  asm("fclr i");                  /* Disable interrupts            */
  Data  = P10;                    /* Read port data                */
  Data |=   (1 << 5) | (1 << 4);  /* Switch on power in Y
                                     and  enable measurement in X  */
  Data &= ~((1 << 3) | (1 << 2)); /* Switch off power in X
```



```
                                        and disable measurement in Y   */
  P10    = Data;                     /* Write port data                */
  asm("fset i");                     /* Enable interrupts              */
}

static void ReadADCx(int channel) {
  ADCON0   = channel                 /* Select channel 0-7             */
           | (0 << 3)                /* One shot mode                  */
           | (0 << 6)                /* A-D conversion start (0=stop) */
           | (0 << 7);               /* FAD/4 select                   */
  ADCON1   =  (0 << 0)               /* A-D sweep select (XX)          */
           | (0 << 2)                /* No sweep mode                  */
           | (0 << 3)                /* 8 bit mode                     */
           | (0 << 4)                /* FAD4 select                    */
           | (1 << 5)                /* VRef connected                 */
           | (0 << 6);               /* Anex0/1 not used               */
  ADCON2   =  (1 << 0);              /* Use example and hold           */
  ADIC     = 0;                      /* Reset IR flag                  */
  ADCON0 |= (1 << 6);                /* Start conversion               */
  while ((ADIC & (1 << 3)) == 0);    /* Wait for end of conversion     */
  ADCON0 &= ~(6 << 0);               /* Start conversion = 0           */
}

int GUI_TOUCH_X_MeasureX(void) {
  ReadADCx(0);
  return AD0;
}

int GUI_TOUCH_X_MeasureY(void) {
  ReadADCx(1);
  return AD1;
}
```

26.4.2.4 Driver API for analog touch screens

The table below lists the available analog touch screen driver routines in alphabetical order. These functions only apply if you are using the driver included with emWin.

Routine	Description
GUI_TOUCH_Calibrate()	Changes the calibration.
GUI_TOUCH_Exec()	Activates the measurement of the X- and Y-axes; needs to be called about 100 times/second.
GUI_TOUCH_GetxPhys()	Returns the x coordintate given from the A/D converter.
GUI_TOUCH_GetyPhys()	Returns the y coordintate given from the A/D converter.
GUI_TOUCH_SetOrientation()	Sets the logical display orientation.

Table 26.18: Driver API for analog touch screens

GUI_TOUCH_Calibrate()

Description

Changes the calibration at runtime.

Prototype

```
int GUI_TOUCH_Calibrate(int Coord, int Log0, int Log1,
                        int Phys0, int Phys1);
```

Parameter	Description
Coord	GUI_COORD_X for X-axis, GUI_COORD_Y for Y-axis.
Log0	Logical value 0 in pixels.
Log1	Logical value 1 in pixels.
Phys0	A/D converter value for Log0.
Phys1	A/D converter value for Log1.

Table 26.19: GUI_TOUCH_Calibrate() parameter list

Additional information

The function takes as parameters the axis to be calibrated, two logical values in pixels for this axis and two corresponding physical values of the A/D converter. Since the logical value `Log0` usually is set to 0, `Log1` should contain the (x- or y-)size decreased by 1.

GUI_TOUCH_Exec()

Description

Polls the touch-screen by calling the `TOUCH_X` routines to activate the measurement of the X- and Y-axes. It is required that this function is called for about 100 times per second, since there is only one axis measured per call. Therefore a complete measurement of the touch screen is done with 2 calls of `GUI_TOUCH_Exec()`.

Prototype

```
void GUI_TOUCH_Exec(void);
```

Additional information

If you are using a real-time operating system, the easiest way to make sure this function is called is to create a separate task. When not using a multitask system, you can use an interrupt service routine to do the job.
This function calls `GUI_TOUCH_StoreState()`.

GUI_TOUCH_GetxPhys()

GUI_TOUCH_GetyPhys()

Description

Returns a measurement value of the x- or y-coordinate given from the A/D-converter.

Prototype

```
void GUI_TOUCH_GetxPhys(void);

void GUI_TOUCH_GetyPhys(void);
```

Additional information

A sample which shows how to use these functions is located in the folder `Sample\Tutorial` and its name is `TOUCH_Sample.c`. Run this example on your hardware.

GUI_TOUCH_SetOrientation()

Description

The function configures the touch screen orientation. If the touch screen for example already has been configured to work with the default orientation and the display now needs to be turned or mirrored, this function can be used to configure the touch driver to use the same orientation as the display without changing anything at the hardware routines.

Prototype

```
void GUI_TOUCH_SetOrientation(unsigned Orientation);
```

Parameter	Description
`Orientation`	One or more "OR" combined values of the table below.

Table 26.20: GUI_TOUCH_SetOrientation() parameter list

Permitted values for parameter Orientation	
GUI_MIRROR_X	Mirroring the X-axis
GUI_MIRROR_Y	Mirroring the Y-axis
GUI_SWAP_XY	Swapping X- and Y-axis

26.4.2.5 Configuring the analog touch-screen driver

The touch screen driver is completely run-time configurable. GUI_TOUCH_Calibrate() should be used to specify the physical values returned by the A/D converter for 2 positions per axis. If the display needs to be turned or mirrored, GUI_TOUCH_SetOrientation() can be used to set a new orientation without changing anything at the hardware routines.
Configuring the touch screen should be done before emWin manages any touch input.

Example

```
#define TOUCH_AD_LEFT   0x3c0
#define TOUCH_AD_RIGHT  0x034
#define TOUCH_AD_TOP    0x3b0
#define TOUCH_AD_BOTTOM 0x034

Orientation = (GUI_MIRROR_X * LCD_GetMirrorXEx(0)) |
              (GUI_MIRROR_Y * LCD_GetMirrorYEx(0)) |
              (GUI_SWAP_XY  * LCD_GetSwapXYEx (0));
GUI_TOUCH_SetOrientation(Orientation);
GUI_TOUCH_Calibrate(GUI_COORD_X, 0, 319, TOUCH_AD_LEFT, TOUCH_AD_RIGHT);
GUI_TOUCH_Calibrate(GUI_COORD_Y, 0, 239, TOUCH_AD_TOP,  TOUCH_AD_BOTTOM);
```


26.5 Joystick input example

The following example shows how the pointer input device API can be used to process the input from a joystick:

```
/*********************************************************************
*
*         _JoystickTask
*
* Purpose:
*   Periodically read the Joystick and inform emWin using
*   GUI_PID_StoreState.
*   It supports dynamic acceleration of the pointer.
*   The Joystick is a simple, standard 5 switch (digital) type.
*
*/
static void _JoystickTask(void) {
  GUI_PID_STATE State;
  int Stat;
  int StatPrev = 0;
  int TimeAcc = 0;   // Dynamic acceleration value
  int xMax, yMax;

  xMax = LCD_GetXSize() - 1;
  yMax = LCD_GetYSize() - 1;
  while (1) {
    Stat = HW_ReadJoystick();
    //
    // Handle dynamic pointer acceleration
    //
    if (Stat == StatPrev) {
      if (TimeAcc < 10) {
        TimeAcc++;
      }
    } else {
      TimeAcc = 1;
    }
    if (Stat || (Stat != StatPrev)) {
      //
      // Compute the new coordinates
      //
      GUI_PID_GetState(&State);
      if (Stat & JOYSTICK_LEFT) {
        State.x -= TimeAcc;
      }
      if (Stat & JOYSTICK_RIGHT) {
        State.x += TimeAcc;
      }
      if (Stat & JOYSTICK_UP) {
        State.y -= TimeAcc;
      }
      if (Stat & JOYSTICK_DOWN) {
        State.y += TimeAcc;
      }
      //
      // Make sure coordinates are still in bounds
      //
      if (State.x < 0) {
        State.x = 0;
      }
      if (State.y < 0) {
        State.y = 0;
      }
      if (State.x >= xMax) {
        State.x = xMax;
      }
      if (State.y > yMax) {
        State.y = yMax;
      }
      //
      // Inform emWin
      //
      State.Pressed = (Stat & JOYSTICK_ENTER) ? 1: 0;
      GUI_PID_StoreState(&State);
      StatPrev = Stat;
    }
    OS_Delay(40);
  }
}
```


Chapter 27

MultiTouch support (MT)

Initially the concept of emWin was based on a single touch and keyboard interface. Since smartphones with MultiTouch capabilities became more and more attractive, it was a need to implement MultiTouch capabilities also to emWin. The emWin implementation is able to recognize up to 10 touch points, wheareas the maximum number of touch points is limited by the target hardware.

Single touch screens usually consist of resistive touch panels. Most of the MultiTouch panels are capacitive panels which behave different to resistive panels. Whereas a resistive touch panel needs a noticable pressure, a capacitive panel just requires a smooth touch for recognizing the touch event.

MultiTouch support is an add-on and not part of the emWin basic package. It has to be purchased separately. The emWin Simulation supports the MultiTouch feature, so it is possible to evaluate MultiTouch samples which are provided on www.segger.com. To do so it is necessary to have a MultiTouch display connected to the computer.

The following chapter shows the implementation of MultiTouch functionality in emWin.

27.1 Introduction

A MultiTouch panel enables the application to react on multiple touch point inputs simultaneously. The implementation of MT support in emWin offers different consecutive levels of MT access:

- Basic buffer access
- Gesture support (requires the Window Manager)
- Automatic window animation (requires the Window Manager)

Gesture support requires basic buffer access, window animation is based on gesture support.

Basic buffer access

The MT buffer is able to store a configurable number of MT events. The buffer access API functions consists of functions for storing new events, polling the buffer, setting the touch screen orientation and basically enabling MT support. A detailed description of the available API functions follows later.

Gesture support

This level of MT support is responsible for gesture recognition and requires the window manager. If a gesture is detected, a `WM_GESTURE` message with more detailed information is send to the according window. It can be used to modify any kind of data. Detailed descriptions how to use the gesture messages follow later.

Window animation

emWin also offers the possibility for automatic window animation via gesture support. Windows can be moved and resized automatically by gesture input. This can be achieved simply by setting the according flags when creating the window. It does not include automatic resizing of fonts and objects shown in the window. This need to be done by the application based on a factor which is passed to/from the application. Details follow later in this chapter.

27.2 Getting started

Only a few things need to be considered to be able to use MT support. It needs to be enabled, the MT buffer needs to be filled and it must be ensured that the buffer is polled by emWin if gesture support or window animation is required.

Enabling MT support

To be able to use MT support it needs to be enabled once. It is recommended to do that immediately after the initialization:

```
void MainTask(void) {
  GUI_Init();
  GUI_MTOUCH_Enable(1);
}
```

Filling the MT buffer

Further it is required to fill the MT buffer with MT events. That can be done either by an existing MT driver like `GUIMTDRV_TangoC32` or by filling the buffer by a custom driver. In case of using a custom driver the function `GUI_MTOUCH_StoreEvent()` needs to be used to do that. The function will be explained later in detail.

Polling the MT buffer

Once MT support has been enabled, the window manager (WM) automatically polls the MT buffer. That is done when executing an emWin update function (typically `GUI_Exec()`, `GUI_Delay()` or `WM_Exec()`). If no gesture support is required the buffer can also be polled manually by the functions `GUI_MTOUCH_GetEvent()` and

GUI_MTOUCH_GetTouchInput(). In case of automatic polling by emWin the gesture detecting module will automatically send WM_GESTURE messages to the according window.

27.3 Using basic buffer access

The functions explained later under 'Basic buffer access API' can be used for that. Polling works as follows:

Polling am MT event from the buffer

GUI_MTOUCH_GetEvent() should be used to get an existing MT event from the buffer. It passes a pointer to a GUI_MTOUCH_EVENT structure to the function to be filled with information like the number of touch points of the current event. If the function fails there is no existing event. Otherwise the GUI_MTOUCH_EVENT structure contains the number of touch points associated to the event.

Getting the touch points of an MT event

GUI_MTOUCH_GetTouchInput() should be used for getting the point information for each single touch point. It passes a pointer to a GUI_MTOUCH_INPUT structure to the function to be filled by the function. That information comprises position, ID and flags.

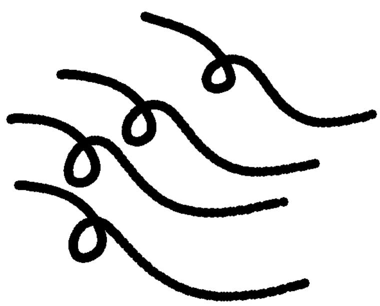
Example 1

The following code shows a very simple function which continuously polls the MT buffer and draws each touch point:

```
#include "GUI.h"

void MainTask(void) {
  GUI_MTOUCH_EVENT Event;
  GUI_MTOUCH_INPUT Input;
  unsigned i;

  GUI_Init();
  GUI_MTOUCH_Enable(1);
  GUI_SetPenSize(5);
  do {
    if (GUI_MTOUCH_GetEvent(&Event) == 0) {
      for (i = 0; i < Event.NumPoints; i++) {
        GUI_MTOUCH_GetTouchInput(&Event, &Input, i);
        GUI_DrawPoint(Input.x, Input.y);
      }
    }
    GUI_Delay(1);
  } while (1);
}
```

Screenshot of above sample

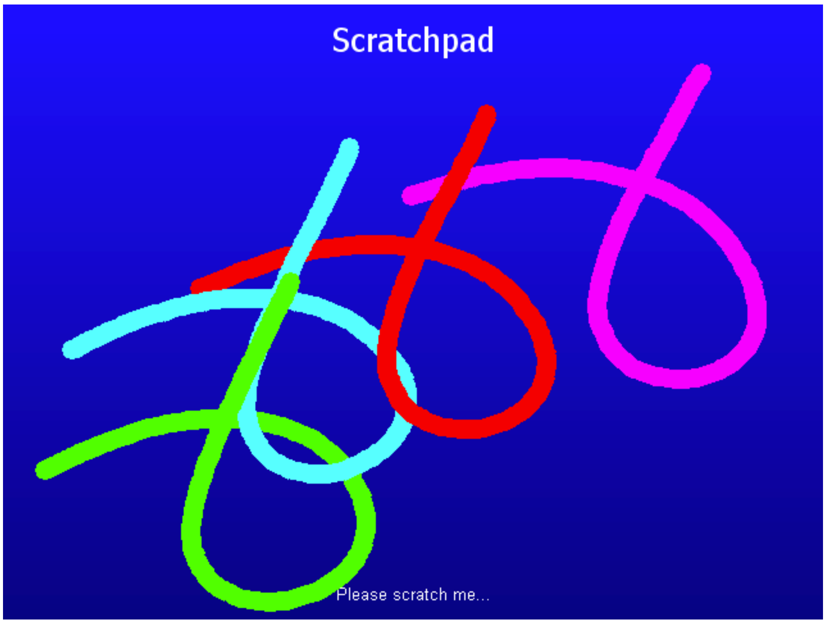
When scratching over the screen with multiple touch points or fingers:



Example 2

The sample folder of emWin contains the sample MTOUCH_ScratchAndGestures.c. It can be used to get more familiar with processing basic buffer access and MT support. It contains a scratchpad sub sample which detects multiple points and uses their IDs for assigning a unique color.

Screenshot



27.4 Using gestures

Gestures in emWin are based on motion detection via MultiTouch panel. Each gesture starts by detecting the first touch input and ends on releasing it.

Requirements

To be able to use gesture support, the following needs to be considered:
- MT support needs to be enabled.
- Gesture support needs to be enabled via `WM_GESTURE_Enable()`.
- The flag `WM_CF_GESTURE` needs to be set by the according window.

Supported gestures

The following table gives an overview of the currently supported gestures:

Gesture	Touch points	Description
`Panning`	1	Dependent on the first motion detection the gesture supports motion on one axis (X or Y) or both simultaneous.
`Zooming`	2	This gesture is started when detecting a relative motion between 2 touch points and can be used to scale an object. Can be combined with rotating and panning.
`Rotating`	2	When detecting initially a change of the angle between 2 touch points rotation gesture is started. Can be combined with zooming and panning.

Table 27.1: Supported gestures

Processing gesture input

Gesture input is send to the according window by a `WM_GESTURE` message. For detailed gesture information a pointer to a structure of type `WM_GESTURE_INFO` (explained in detail below) is passed to the window. The '`Flags`' element of that structure is used to specify the kind of information passed to the window. The following table shows the currently supported flags:

Flag	Description
`WM_GF_BEGIN`	This flag is set when sending the first message for the gesture.
`WM_GF_PAN`	A panning gesture is detected. The element '`Point`' of `WM_GESTURE_INFO` contains the relative movement in pixels to be processed by the application.
`WM_GF_ROTATE`	Rotation is active. The element '`Angle`' of `WM_GESTURE_INFO` contains the relative movement in degrees (<< 16) to be processed by the application. To be able to achieve a smooth rotation the value is passed in 1/65536 degrees. If movement should be considered simultaneously the element '`Point`' contains also the relative movement.
`WM_GF_ZOOM`	Zooming is active. When starting a zooming gesture the element '`Factor`' of `WM_GESTURE_INFO` has to be set to the initial value to be used by the gesture. During the gesture the same element contains the updated value to be processed by the application. If movement should be considered simultaneously the element '`Point`' contains also the relative movement.
`WM_GF_END`	Set when releasing a touch point at the end of a gesture.

Table 27.2: Gesture flags

Elements of structure WM_GESTURE_INFO

Data type	Element	Description
int	Flags	Already explained above.
GUI_POINT	Point	Relative movement to be processed by the application.
I32	Angle	Relative angle difference to be processed by the application.
I32	Factor	When starting a zoom gesture the application has to set the element to the initial value for the gesture. After that during the gesture it contains the updated value to be processed by the application.

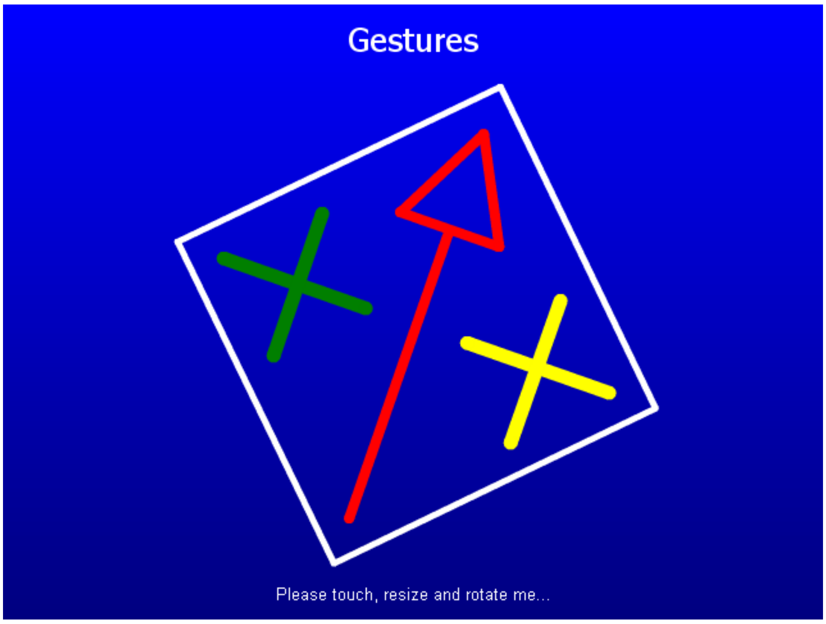
Table 27.3: WM_GESTURE_INFO element list

The above table explains all elements of the structure which are relevant for processing gesture messages. All other elements should not be used by the application.

Example

The sample folder of emWin contains the sample `MTOUCH_ScratchAndGestures.c`. It can be used to get more familiar with processing gestures and MT support. It contains a sub sample which animates a small vector graphic by gesture functions.

Screenshot



27.5 Window animation

(preliminary)
Automatic window animation can be used to scale and pan windows automatically by gesture input. Currently that feature can be used for bare windows and can not be used with any of the widgets. Scaling of objects shown within scaled windows need to be done by the application. For that purpose the element 'Factor' of the WM_GESTURE_INFO structure can be used as explained in the following.

Requirements

To be able to use automatic window animation, the following needs to be considered:
- MT support needs to be enabled.
- Gesture support needs to be enabled via WM_GESTURE_Enable().
- The window flags WM_CF_GESTURE and WM_CF_ZOOM need to be set.
- The window has to pass a pointer to a WM_ZOOM_INFO structure on demand.

Reacting on gestures

Processing automatic window animation is similar to bare gesture support with the difference, that the object to be animated is the window itself which is modified automatically by emWin. But to be able to do this automatic animation the window manager needs additional information. This is done by passing a pointer to a WM_ZOOM_INFO structure to the WM. It is passed by an additional element of the already explained structure WM_GESTURE_INFO.

Additional element of WM_GESTURE_INFO

Data type	Element	Description
WM_ZOOM_INFO *	pZoomInfo	Pointer to be set to a valid location of a WM_ZOOM_INFO structure. The application should keep sure, that the location remains valid during the gesture.

Table 27.4: WM_GESTURE_INFO additional element

Elements of structure WM_ZOOM_INFO

Data type	Element	Description
I32	FactorMin	Minimum factor to be used (<< 16).
I32	FactorMax	Maximum factor to be used (<< 16).
U32	xSize	Native xSize of window to be zoomed in pixels.
U32	ySize	Native ySize of window to be zoomed in pixels.

Table 27.5: WM_ZOOM_INFO element list

The factor is shifted by 16 bits to be able to achieve a smooth result. The native size needs to be passed by the application to the WM because otherwise it is not possible to get the exact size of the window in unscaled state.

Limits

The window to be animated can be moved and scaled by using the parent window as a kind of viewport. If the size of the animated window is larger than the parent the WM makes sure that the zoomed window covers the complete area of the parent window. If it is smaller the WM makes sure, that it is not moved over the parents border.

Scalable fonts

Text can be rendered at different sizes using scalable TrueType fonts. This requires a fast CPU and a reasonable amount of RAM and ROM. Details can be found in the chapter “TrueType Font (TTF) format” on page 213.

27.6 Basic buffer access API

The table below lists the available MT buffer access routines in alphabetical order. Detailed descriptions follow:

Routine	Description
GUI_MTOUCH_Enable()	Enables use of MT buffer.
GUI_MTOUCH_GetEvent()	Polls an event from the MT buffer.
GUI_MTOUCH_GetTouchInput()	Returns detailed information about a dedicated touchpoint.
GUI_MTOUCH_IsEmpty()	Evaluates if the MT buffer is empty.
GUI_MTOUCH_SetOrientation()	Can be used to set the touch screen orientation.
GUI_MTOUCH_StoreEvent()	Puts an event into the MT buffer.

Table 27.6: Basic buffer access API list

GUI_MTOUCH_Enable()

Description

This routine needs to be called to enable the MT buffer.

Prototype

```
void GUI_MTOUCH_Enable(int OnOff);
```

Parameter	Description
OnOff	1 for enabling the buffer, 0 for disabling.

Table 27.7: GUI_MTOUCH_Enable() parameter list

GUI_MTOUCH_GetEvent()

Description

Returns an event and removes it from the buffer.

Prototype

```
int GUI_MTOUCH_GetEvent(GUI_MTOUCH_EVENT * pEvent);
```

Parameter	Description
pEvent	Pointer to a structure of type GUI_MTOUCH_EVENT to be filled by the function.

Table 27.8: GUI_MTOUCH_GetEvent() parameter list

Elements of structure GUI_MTOUCH_EVENT

Data type	Element	Description
int	LayerIndex	Layer index of touched layer (normally 0)
unsigned	NumPoints	Number of available touch points.
GUI_TIMER_TIME	TimeStamp	Time stamp in ms of that event.

Table 27.9: GUI_MTOUCH_EVENT element list

Return value

0 on success, 1 if the function fails. Can be used to check if an event was available or not.

Additional information

The most important information returned by that function is the availability of an event and how many touch points are available. Further the time stamp could be of interest which is filled automatically on storing an event into the buffer.

GUI_MTOUCH_GetTouchInput()

Description

This function is responsible for getting information of a dedicated touch point. It requires a pointer to a GUI_MTOUCH_EVENT structure previously filled by GUI_MTOUCH_GetEvent() and a pointer to a GUI_MTOUCH_INPUT structure to be filled by this function with the touchpoint details.

Prototype

```
int GUI_MTOUCH_GetTouchInput(GUI_MTOUCH_EVENT * pEvent,
                             GUI_MTOUCH_INPUT * pBuffer,
                             unsigned Index);
```

Parameter	Description
pEvent	Pointer to the event structure filled by GUI_MTOUCH_GetEvent()
pBuffer	Pointer to a structure of type GUI_MTOUCH_INPUT to be filled by the function.
Index	Index of the requested touch point.

Table 27.10: GUI_MTOUCH_GetTouchInput() parameter list

Elements of structure GUI_MTOUCH_INPUT

Data type	Element	Description
I32	x	X-position in pixels of the touch point.
I32	y	Y-position in pixels of the touch point.
U32	Id	Unique Id, should be provided by the touch driver.
U16	Flags	(see table below)

Table 27.11: GUI_MTOUCH_INPUT element list

Permitted values for element Flags	
GUI_MTOUCH_FLAG_DOWN	New touch point has touched the surface.
GUI_MTOUCH_FLAG_MOVE	Touch point has been moved.
GUI_MTOUCH_FLAG_UP	Touch point has released the surface.

Return value

0 on success, 1 on failure.

Additional information

The parameter Index needs to be < the available number of touch points.
A unique Id is normally managed the touch controller which is passed by the Id element of GUI_MTOUCH_INPUT.

GUI_MTOUCH_IsEmpty()

Description

Returns if the MT buffer is empty or not.

Prototype

```
int GUI_MTOUCH_IsEmpty(void);
```

Return value

1 if buffer is empty, 0 if it contains MT events.

GUI_MTOUCH_SetOrientation()

Description

If the display does not operate by the default orientation or if the display orientation is different to the MT orientation, that function can be used to change the touch screen orientation.

Prototype

```
void GUI_MTOUCH_SetOrientation(int Orientation);
```

Parameter	Description
Orientation	One or more "OR" combined values of the table below

Table 27.12: GUI_MTOUCH_SetOrientation() parameter list

Permitted values for parameter Orientation	
GUI_MIRROR_X	Mirroring the X-axis
GUI_MIRROR_Y	Mirroring the Y-axis
GUI_SWAP_XY	Swapping X- and Y-axis

GUI_MTOUCH_StoreEvent()

Description

Routine to be used by the touch screen driver to store a new event with associated touch points into the MT buffer. The number of available touch points is passed by the NumPoints element of the structure pointed by pEvent. pInput then points to an array of GUI_MTOUCH_INPUT structures containing the information of each touch point.

Prototype

```
void GUI_MTOUCH_StoreEvent(GUI_MTOUCH_EVENT * pEvent,
                           GUI_MTOUCH_INPUT * pInput);
```

Parameter	Description
pEvent	Pointer to a structure of type GUI_MTOUCH_EVENT.
pInput	Pointer to the first element of an array of GUI_MTOUCH_INPUT structures.

Table 27.13: GUI_MTOUCH_StoreEvent() parameter list

Additional information

The number of possible touch points is limited per default to 10.
The new event will automatically get a time stamp information which can be used later.

Chapter 28

Keyboard Input

emWin provides support for any kind of keyboards. Any type of keyboard driver is compatible with emWin.
The software for keyboard input is located in the subdirectory `GUI\Core` and part of the basic package.

28.1 Description

A keyboard input device uses ASCII character coding in order to be able to distinguish between characters. For example, there is only one "A" key on the keyboard, but an uppercase "A" and a lowercase "a" have different ASCII codes (0x41 and 0x61, respectively).

emWin predefined character codes

emWin also defines character codes for other "virtual" keyboard operations. These codes are listed in the table below, and defined in an identifier table in `GUI.h`. A character code in emWin can therefore be any extended ASCII character value or any of the following predefined emWin values.

Predefined virtual key code	Description
GUI_KEY_BACKSPACE	Backspace key.
GUI_KEY_TAB	Tab key.
GUI_KEY_BACKTAB	Shift + Tab key.
GUI_KEY_ENTER	Enter/return key.
GUI_KEY_LEFT	Left arrow key.
GUI_KEY_UP	Up arrow key.
GUI_KEY_RIGHT	Right arrow key.
GUI_KEY_DOWN	Down arrow key.
GUI_KEY_HOME	Home key (move to beginning of current line).
GUI_KEY_END	End key (move to end of current line).
GUI_KEY_SHIFT	Shift key.
GUI_KEY_CONTROL	Control key.
GUI_KEY_ESCAPE	Escape key.
GUI_KEY_INSERT	Insert key.
GUI_KEY_DELETE	Delete key.
GUI_KEY_SPACE	Space key.
GUI_KEY_PGUP	Page up key.
GUI_KEY_PGDOWN	Page down key.

Table 28.1: Predefined character codes

28.2 Driver layer API

The keyboard driver layer handles keyboard messaging functions. These routines notify the Window Manager when specific keys (or combinations of keys) have been pressed or released. The table below lists the driver-layer keyboard routines in alphabetical order. Detailed descriptions follow.

Routine	Description
GUI_StoreKeyMsg()	Stores a key message in the keyboard buffer.
GUI_SendKeyMsg()	Sends a key message to the currently focussed window.

Table 28.2: Driver layer API list

GUI_StoreKeyMsg()

Description

Stores a key message in the keyboard buffer.

Prototype

```
void GUI_StoreKeyMsg(int Key, int Pressed);
```

Parameter	Description
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined emWin character code.
Pressed	Key state. See table below.

Table 28.3: GUI_StoreKeyMsg() parameter list

Permitted values for parameter Pressed	
1	Pressed state.
0	Released (unpressed) state.

Additional information

This function can be used from an interrupt service routine.
The keyboard input manager of emWin contains a FIFO buffer which is able to hold up to 10 keyboard events per default. If a different size is required this value can be changed. Details can be found in the section "Advanced GUI configuration options" on page 1213.
The Window Manager polls the keyboard buffer automatically and sends according keyboard messages to the currently focussed window.

GUI_SendKeyMsg()

Description

Sends a key message to the window with the input focus. If no window has the input focus, the function `GUI_StoreKeyMsg()` is called to store the data to the input buffer.

Prototype

```
void GUI_SendKeyMsg(int Key, int Pressed);
```

Parameter	Description
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined emWin character code.
Pressed	Key state (see GUI_StoreKeyMsg()).

Table 28.4: GUI_SendKeyMsg() parameter list

Additional information

This function should not be called from an interrupt service routine.

28.3 Application layer API

The table below lists the application-layer keyboard routines in alphabetical order. Detailed descriptions follow.

Routine	Description
GUI_ClearKeyBuffer()	Clear the key buffer.
GUI_GetKey()	Return the contents of the key buffer.
GUI_GetKeyState()	Returns the current key state.
GUI_StoreKey()	Store a key in the buffer.
GUI_WaitKey()	Wait for a key to be pressed.

Table 28.5: Application layer API list

GUI_ClearKeyBuffer()

Description

Clears the key buffer.

Prototype

```
void GUI_ClearKeyBuffer(void);
```

GUI_GetKey()

Description

Returns the current content of the key buffer.

Prototype

```
int GUI_GetKey(void);
```

Return value

Codes of characters in the key buffer; 0 if no key is buffered.

GUI_GetKeyState()

Description

Returns the current key state.

Prototype

```
void GUI_GetKeyState(GUI_KEY_STATE * pState);
```

Parameter	Description
pState	This structure is filled by the function. See elements below.

Table 28.6: GUI_GetKeyState() parameter list

Elements of structure GUI_KEY_STATE

Data type	Element	Description
int	Key	Key code.
int	Pressed	1, if the key is pressed. 0, if the key is not pressed. -1, if the state could not be determined.

Table 28.7: GUI_KEY_STATE element list

GUI_StoreKey()

Description

Stores a key in the buffer.

Prototype

```
void GUI_StoreKey(int Key);
```

Parameter	Description
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined emWin character code.

Table 28.8: GUI_StoreKey() parameter list

Additional information

This function is typically called by the driver and not by the application itself.

GUI_WaitKey()

Description

Waits for a key to be pressed.

Prototype

```
int GUI_WaitKey(void);
```

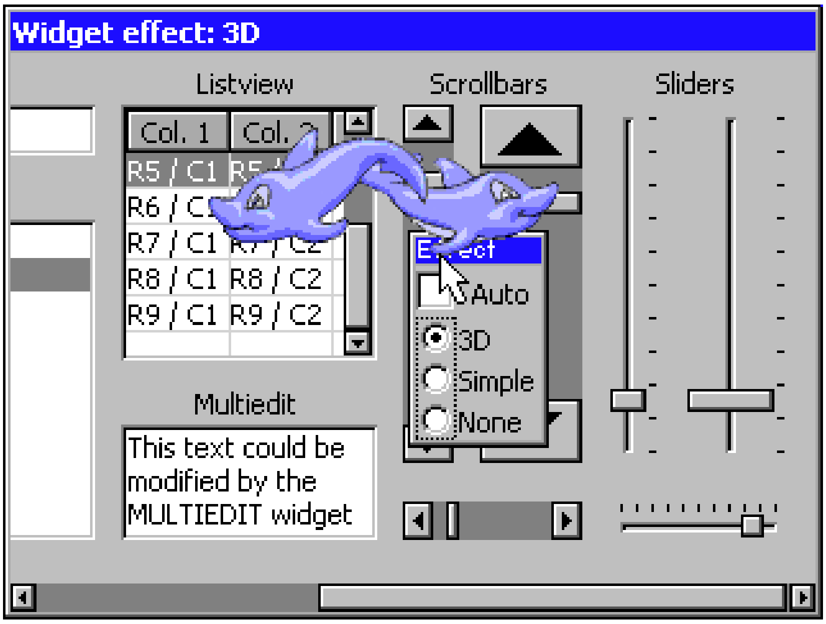
Additional information

The application is "blocked", meaning it will not return until a key is pressed.

Chapter 29

Sprites

A 'Sprite' is an image which can be shown above all other graphics on the screen. A Sprite preserves the screen area it covers. It can be moved or removed at any time, fully restoring the screen content. Animation by use of multiple images is possible.
Sprites are completely independent from all other drawing operations as well as window operations: Sprites do not affect drawing or window operations; drawing or window operations do not affect Sprites.
Sprites can be seen as objects which are sitting "on top" of the screen, similar to cursors.



29.1 Introduction

emWin Sprites are implemented as a pure software solution. No additional hardware is required to use emWin Sprites. They can be shown, moved and deleted without affecting already visible graphics.

Memory requirements

Each Sprite needs a memory area for saving the display data 'behind' the Sprite to be able to restore the background on moving operations or on removing the Sprite. Further a memory area for a color cache is required. The size of the color cache depends on the number of colors used in the Sprite image. So the complete number of bytes required for a Sprite can be calculated as follows:

```
SizeOfSpriteObject (~30 bytes) +
(XSize * YSize + NumberOfBitmapColors) * REQUIRED_BYTES_PER_PIXEL
```

Maximum number of Sprites

The number of simultaneous visible Sprites is not limited by emWin. It depends only on the available memory.

Performance

Note that drawing a Sprite is more computer-bound than drawing a simple bitmap, because it has to manage the background data and intersections with other Sprites.

Z-order

Z-order is an ordering of overlapping two-dimensional objects, in this case the Sprites. When two Sprites overlap, their Z-order determines which one appears on top of the other. The Sprite created at last is the topmost Sprite.

29.2 Sprite API

The table below lists the available Sprite-related routines in alphabetical order. Detailed descriptions follow:

Routine	Description
GUI_SPRITE_Create()	Creates a Sprite.
GUI_SPRITE_CreateAnim()	Creates an animated Sprite.
GUI_SPRITE_CreateEx()	Creates a Sprite in the given layer.
GUI_SPRITE_CreateExAnim()	Creates an animated Sprite in the given layer.
GUI_SPRITE_CreateHidden()	Creates a hidden Sprite.
GUI_SPRITE_CreateHiddenEx()	Creates a hidden Sprite in the given layer.
GUI_SPRITE_Delete()	Deletes a Sprite.
GUI_SPRITE_GetState()	Return if the Sprite is visible or not.
GUI_SPRITE_Hide()	Hides a Sprite.
GUI_SPRITE_SetBitmap()	Sets a new bitmap of a Sprite.
GUI_SPRITE_SetBitmapAndPosition()	Sets a new bitmap and the position of a Sprite.
GUI_SPRITE_SetLoop()	Enables/Disables infinite animation of the given Sprite.
GUI_SPRITE_SetPosition()	Sets the position of a Sprite.
GUI_SPRITE_Show()	Shows the given Sprite.
GUI_SPRITE_StartAnim()	Starts the animation of the given Sprite.
GUI_SPRITE_StopAnim()	Stops the animation of the given Sprite.

Table 29.1: Sprite API list

GUI_SPRITE_Create()

Description

Creates a Sprite at the given position in the current layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_Create(const GUI_BITMAP * pBM, int x, int y);
```

Parameter	Description
pBM	Pointer to a bitmap structure to be used for drawing the Sprite.
x	X-position of the Sprite in screen coordinates.
y	Y-position of the Sprite in screen coordinates.

Table 29.2: GUI_SPRITE_Create() parameter list

Return value

Handle of the new Sprite, 0 on failure.

Additional information

The bitmap addressed by the parameter `pBM` needs to agree with the following requirements:

- It should not be compressed.
- It needs to be transparent.
- It needs to be a palette based bitmap with 1, 2, 4 or 8bpp or, if semi transparency is required, a true color bitmap.

Other bitmaps or insufficient memory cause the function to fail.

GUI_SPRITE_CreateAnim()

Description

Creates an animated Sprite at the given position in the current layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_CreateAnim(const GUI_BITMAP ** ppBm,
                                  int x, int y, unsigned Period,
                                  const unsigned * pPeriod,
                                  int NumItems);
```

Parameter	Description
ppBM	Pointer to an array of bitmap pointers to be used for drawing the Sprite.
x	X-position of the Sprite in screen coordinates.
y	Y-position of the Sprite in screen coordinates.
Period	Period to be used to switch between the images.
pPeriod	Pointer to an array containing the periods to be used to switch between the images.
NumItems	Number of images.

Table 29.3: GUI_SPRITE_CreateAnim() parameter list

Return value

Handle of the new Sprite, 0 on failure.

Additional information

The bitmaps addressed by the parameter `ppBM` needs to agree with the following requirements:

- They need to have exactly the same X- and Y-size.
- They should not be compressed.
- They need to be transparent.
- They need to be palette based bitmaps with 1, 2, 4 or 8bpp or, if semi transparency is required, true color bitmaps.

Using bitmaps which do not match above criteria causes the function to fail as well as insufficient memory.
The parameter `pPeriod` is required, only if the periods for the images are different. If the same period should be used for all images the parameter `Period` should be used. In this case `pPeriod` can be `NULL`.
In case `pPeriod` is used, the animation will stop at the according image if one of the timer values is 0.

GUI_SPRITE_CreateEx()

Description

Creates a Sprite at the given position in the desired layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_CreateEx(const GUI_BITMAP * pBM,
                                int x, int y, int Layer);;
```

Parameter	Description
pBM	Pointer to a bitmap structure to be used for drawing the Sprite.
x	X-position of the Sprite in screen coordinates.
y	Y-position of the Sprite in screen coordinates.
Layer	Layer of Sprite.

Table 29.4: GUI_SPRITE_CreateEx() parameter list

Return value

Handle of the new Sprite, 0 on failure.

Additional information

Additional information can be found under "GUI_SPRITE_Create()" on page 1005.

GUI_SPRITE_CreateExAnim()

Description

Creates an animated Sprite at the given position in the current layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_CreateAnim(const GUI_BITMAP ** ppBm,
                                  int x, int y, unsigned Period,
                                  const unsigned * pPeriod,
                                  int NumItems, int LayerIndex);
```

Parameter	Description
ppBM	Pointer to an array of bitmap pointers to be used for drawing the Sprite.
x	X-position of the Sprite in screen coordinates.
y	Y-position of the Sprite in screen coordinates.
Period	Period to be used to switch between the images.

Table 29.5: GUI_SPRITE_CreateExAnim() parameter list

Parameter	Description
pPeriod	Pointer to an array containing values to be used to switch between the images.
NumItems	Number of images.
LayerIndex	Layer of Sprite.

Table 29.5: GUI_SPRITE_CreateExAnim() parameter list

Return value

Handle of the new Sprite, 0 on failure.

Additional information

Additional information can be found under “GUI_SPRITE_CreateAnim()” on page 1005.

GUI_SPRITE_CreateHidden()

Description

Creates a hidden Sprite at the given position in the current layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_CreateHidden(const GUI_BITMAP * pBM, int x, int y);
```

Parameter	Description
pBM	Pointer to a bitmap structure to be used for drawing the Sprite.
x	X-position of the Sprite in screen coordinates.
y	Y-position of the Sprite in screen coordinates.

Table 29.6: GUI_SPRITE_CreateHidden() parameter list

Return value

Handle of the new Sprite, 0 on failure.

Additional information

More details can be found in the description of “GUI_SPRITE_Create()” on page 1005.

GUI_SPRITE_CreateHiddenEx()

Description

Creates a hidden Sprite at the given position in the given layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_CreateHiddenEx(const GUI_BITMAP * pBM,
                                      int x, int y, int Layer);
```

Parameter	Description
pBM	Pointer to a bitmap structure to be used for drawing the Sprite.
x	X-position of the Sprite in screen coordinates.
y	Y-position of the Sprite in screen coordinates.
Layer	Layer to be used.

Table 29.7: GUI_SPRITE_CreateHiddenEx() parameter list

Return value

Handle of the new Sprite, 0 on failure.

Additional information

More details can be found in the description of “GUI_SPRITE_Create()” on page 1005.

GUI_SPRITE_Delete()

Description

Deletes the given Sprite.

Prototype

```
void GUI_SPRITE_Delete(GUI_HSPRITE hSprite);
```

Parameter	Description
hSprite	Handle of Sprite to be deleted.

Table 29.8: GUI_SPRITE_Delete() parameter list

Additional information

The function deletes the Sprite from the memory and restores its background automatically.

GUI_SPRITE_GetState()

Description

Returns if the given Sprite is visible or not.

Prototype

```
int GUI_SPRITE_GetState(GUI_HSPRITE hSprite);
```

Parameter	Description
hSprite	Handle of Sprite.

Table 29.9: GUI_SPRITE_GetState() parameter list

Return value

1 if it is visible, 0 if not.

GUI_SPRITE_Hide()

Description

Hides the given Sprite.

Prototype

```
void GUI_SPRITE_Hide(GUI_HSPRITE hSprite);
```

Parameter	Description
hSprite	Handle of Sprite to hide.

Table 29.10: GUI_SPRITE_Hide() parameter list

Additional information

The function removes the given Sprite from the list of visible Sprites.

GUI_SPRITE_SetBitmap()

Description

Sets a new image for drawing the Sprite.

Prototype

```
int GUI_SPRITE_SetBitmap(GUI_HSPRITE hSprite,
```



```
const GUI_BITMAP * pBM);
```

Parameter	Description
hSprite	Handle of Sprite.
pBM	Pointer to a bitmap structure to be used for drawing the Sprite.

Table 29.11: GUI_SPRITE_SetBitmap() parameter list

Return value

0 on success, 1 if the routine fails.

Additional information

The new bitmap must have exact the same size as the previous one. Passing a pointer to a bitmap of a different size causes the function to fail.
The function immediately replaces the visible Sprite image on the screen. No further operation is required for showing the new image.

GUI_SPRITE_SetBitmapAndPosition()

Description

Sets the position and the image at once.

Prototype

```
int GUI_SPRITE_SetBitmapAndPosition(GUI_HSPRITE hSprite,
                                    const GUI_BITMAP * pBM,
                                    int x, int y);
```

Parameter	Description
hSprite	Handle of Sprite.
pBM	Pointer to the new bitmap structure to be used to draw the Sprite.
x	New X-position in screen coordinates.
y	New Y-position in screen coordinates.

Table 29.12: GUI_SPRITE_SetBitmapAndPosition() parameter list

Additional information

It makes a difference on using the functions `GUI_SPRITE_SetBitmap()` and `GUI_SPRITE_SetPosition()` one after another or using this function. Whereas the image on the screen will be rendered twice on calling `GUI_SPRITE_SetBitmap()` and `GUI_SPRITE_SetPosition()` it is rendered only once on using this function, which can be used very well in animations.

GUI_SPRITE_SetLoop()

Description

Enables/Disables infinite animation of the given Sprite.

Prototype

```
int GUI_SPRITE_SetLoop(GUI_HSPRITE hSprite, int OnOff);
```

Parameter	Description
hSprite	Handle of the Sprite.
OnOff	1 to enable infinite animation.0 to disable it.

Table 29.13: GUI_SPRITE_SetLoop() parameter list

Return value

-1, if the function failed.
0, if infinite animation was not previously set.
1, if infinite animation was previously set.

GUI_SPRITE_SetPosition()

Description

Moves the Sprite to the new position.

Prototype

```
void GUI_SPRITE_SetPosition(GUI_HSPRITE hSprite, int x, int y);
```

Parameter	Description
hSprite	Handle of Sprite.
x	New X-position in screen coordinates.
y	New Y-position in screen coordinates.

Table 29.14: GUI_SPRITE_SetPosition() parameter list

Additional information

The function moves the given Sprite to the new position.

GUI_SPRITE_Show()

Description

Shows the given Sprite.

Prototype

```
void GUI_SPRITE_Show(GUI_HSPRITE hSprite);
```

Parameter	Description
hSprite	Handle of Sprite.

Table 29.15: GUI_SPRITE_Show() parameter list

Additional information

The function adds the given Sprite to the list of visible Sprites.

GUI_SPRITE_StartAnim()

Description

Starts the animation of the given Sprite.

Prototype

```
int GUI_SPRITE_StartAnim(GUI_HSPRITE hSprite);
```

Parameter	Description
hSprite	Handle of the Sprite.

Table 29.16: GUI_SPRITE_StartAnim() parameter list

Return value

0 on success. 1 on error.

GUI_SPRITE_StopAnim()

Description

Stops the animation of the given Sprite.

Prototype

```
void GUI_SPRITE_Show(GUI_HSPRITE hSprite);
```

Parameter	Description
hSprite	Handle of the Sprite.

Table 29.17: GUI_SPRITE_StopAnim() parameter list

Return value

0 on success. 1 on error.

Chapter 30

Cursors

emWin includes a system-wide cursor which may be changed to other, predefined styles. Also automatically animated cursors are supported. Although the cursor always exists, it is hidden by default. It will not be visible until a call is made to show it, and may be hidden again at any point.

30.1 Available cursors

The following cursor styles are currently available. If a call to `GUI_CURSOR_Show()` is made and no style is specified with `GUI_CURSOR_Select()`, the default cursor will be a medium arrow.

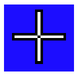
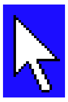
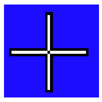
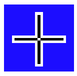
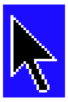
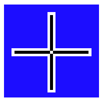
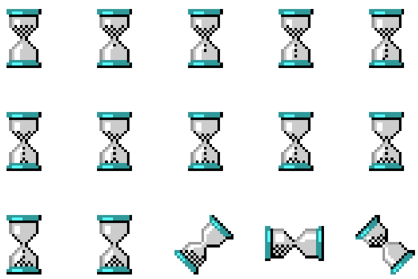
Arrow cursors		**Cross cursors**	
GUI_CursorArrowS Small arrow		GUI_CursorCrossS Small cross	
GUI_CursorArrowM Medium arrow (default cursor)		GUI_CursorCrossM Medium cross	
GUI_CursorArrowL Large arrow		GUI_CursorCrossL Large cross	
Inverted arrow cursors		**Inverted cross cursors**	
GUI_CursorArrowSI Small inverted arrow		GUI_CursorCrossSI Small inverted cross	
GUI_CursorArrowMI Medium inverted arrow		GUI_CursorCrossMI Medium inverted cross	
GUI_CursorArrowLI Large inverted arrow		GUI_CursorCrossLI Large inverted cross	
Animated cursors			
GUI_CursorAnimHourglassM Medium animated hourglass			

Table 30.1: Available cursors

30.2 Cursor API

The table below lists the available cursor-related routines in alphabetical order. Detailed descriptions follow:

Routine	Description
GUI_CURSOR_GetState()	Returns if the cursor is visible or not.
GUI_CURSOR_Hide()	Hides the cursor.
GUI_CURSOR_Select()	Sets a specified cursor.
GUI_CURSOR_SelectAnim()	Sets an animated cursor.
GUI_CURSOR_SetPosition()	Sets the cursor position.
GUI_CURSOR_Show()	Shows the cursor.

Table 30.2: Cursor API list

GUI_CURSOR_GetState()

Description

Returns if the cursor is currently visible or not.

Prototype

```
int GUI_CURSOR_GetState(void);
```

Return value

1 if the cursor is visible and 0 if not.

GUI_CURSOR_Hide()

Description

Hides the cursor.

Prototype

```
void GUI_CURSOR_Hide(void);
```

Additional information

This is the default cursor setting. If the cursor should be visible, the function `GUI_CURSOR_Show()` needs to be called.

GUI_CURSOR_Select()

Description

Sets a specified cursor style.

Prototype

```
void GUI_CURSOR_Select(const GUI_CURSOR * pCursor);
```

Parameter	Description
pCursor	Pointer to the cursor to be selected.

Table 30.3: GUI_CURSOR_Select() parameter list

Permitted values for parameter pCursor (Predefined cursors)	
GUI_CursorArrowS	Small arrow.
GUI_CursorArrowM	Medium arrow.
GUI_CursorArrowL	Large arrow.
GUI_CursorArrowSI	Small inverted arrow.
GUI_CursorArrowMI	Medium inverted arrow.

Permitted values for parameter pCursor (Predefined cursors)	
GUI_CursorArrowLI	Large inverted arrow.
GUI_CursorCrossS	Small cross.
GUI_CursorCrossM	Medium cross.
GUI_CursorCrossL	Large cross.
GUI_CursorCrossSI	Small inverted cross.
GUI_CursorCrossMI	Medium inverted cross.
GUI_CursorCrossLI	Large inverted cross.

Additional information

If this function is not called, the default cursor is a medium arrow.

GUI_CURSOR_SelectAnim()

Description

Sets an animated cursor.

Prototype

```
int GUI_CURSOR_SelectAnim(const GUI_CURSOR_ANIM * pCursorAnim);
```

Parameter	Description
pCursorAnim	Pointer to a GUI_CURS_ANIM structure used for the animation.

Table 30.4: GUI_CURSOR_SelectAnim() parameter list

Permitted values for parameter pCursorAnim (Predefined cursors)	
GUI_CursorAnimHourglassM	Animated hourglass, medium size.

Elements of structure GUI_CURSOR_ANIM

Data type	Element	Description
const GUI_BITMAP **	ppBm	Pointer to an array of pointers to bitmaps to be used for the animated cursor.
int	xHot	X-position of hot spot. Details can be found below.
int	yHot	Y-position of hot spot. Details can be found below.
unsigned	Period	Period to be used to switch between the images.
unsigned *	pPeriod	Pointer to an array containing the periods to be used to switch between the images.
int	NumItems	Number of images used for the animation.

Table 30.5: GUI_CURSOR_ANIM element list

Additional information

The bitmaps addressed by ppBM needs to agree with the following requirements:

- They need to have exactly the same X- and Y-size.
- They should not be compressed.
- They need to be transparent.
- They need to be palette based bitmaps with 1, 2, 4 or 8bpp.

Other bitmaps or insufficient memory cause the function to fail.
The pPeriod is only required if the periods for the images are different. If the same period should be used for all images Period should be used instead of pPeriod. In this case pPeriod should be NULL.
xHot and yHot determine the hot spot position of the cursor. This means the relative position in X and Y from the upper left corner of the image to the position of the pointer input device.

Customized cursors can be realized by passing a pointer to a custom defined GUI_CURSOR_ANIM structure.

GUI_CURSOR_SetPosition()

Description

Sets the cursor position.

Prototype

```
void GUI_CURSOR_SetPosition(int x, int y);
```

Parameter	Description
x	X-position of the cursor.
y	Y-position of the cursor.

Table 30.6: GUI_CURSOR_SetPosition() parameter list

Additional information

Normally this function is called internally by the Window Manager and does not need to be called from the application.

GUI_CURSOR_Show()

Description

Shows the cursor.

Prototype

```
void GUI_CURSOR_Show(void);
```

Additional information

The default setting for the cursor is hidden; therefore this function must be called if you want the cursor to be visible.

Chapter 31

Antialiasing

Lines are approximated by a series of pixels that must lie at display coordinates. They can therefore appear jagged, particularly lines which are nearly horizontal or nearly vertical. This jaggedness is called aliasing.
Antialiasing is the smoothing of lines and curves. It reduces the jagged, stair-step appearance of any line that is not exactly horizontal or vertical. emWin supports different antialiasing qualities, antialiased fonts and high-resolution coordinates.
Support for antialiasing is a separate software item and is not included in the emWin basic package. The software for antialiasing is located in the subdirectory `GUI\AntiAlias`.

31.1 Introduction

Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground. The higher the number of shades used between background and foreground colors, the better the antialiasing result (and the longer the computation time).

31.1.1 Quality of antialiasing

The quality of antialiasing is set by the routine `GUI_AA_SetFactor()`, which is explained later in this chapter. For an idea of the relationship between the antialiasing factor and the corresponding result, take a look at the image pictured.
The first line is drawn without antialiasing (factor 1). The second line is drawn antialiased using factor 2. This means that the number of shades from foreground to background is 2 x 2 = 4. The next line is drawn with an antialiasing factor of 3, so there are 3 x 3 = 9 shades, and so on. Factor 4 should be sufficient for most applications. Increasing the antialiasing factor further does not improve the result significantly, but increases the calculation time dramatically.

31.1.2 Antialiased fonts

Two types of antialiased fonts, low-quality (2bpp) and high-quality (4bpp), are supported. The routines required to display these fonts are automatically linked when using them. The following table shows the effect on drawing the character C without antialiasing and with both types of antialiased fonts:

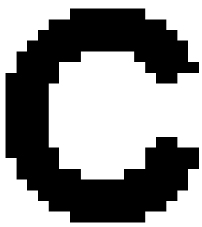
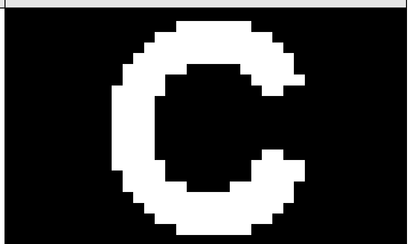
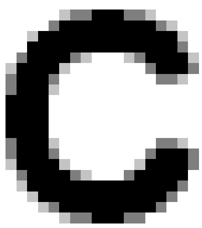
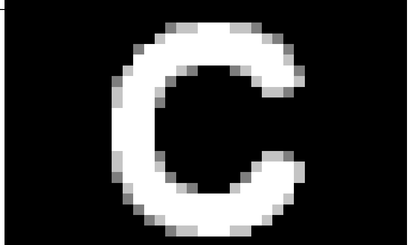
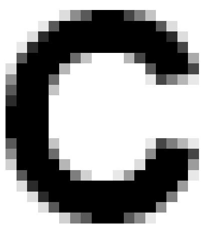
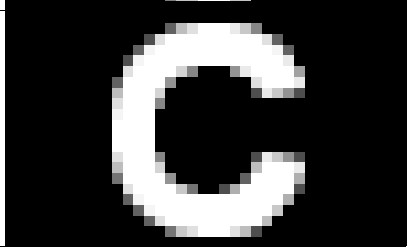
Font type	Black on white	White on black
Standard (no antialiasing) 1 bpp 2 shades		
Low-quality (antialiased) 2 bpp 4 shades		
High-quality (antialiased) 4 bpp 16 shades		

Table 31.1: Antialiased fonts

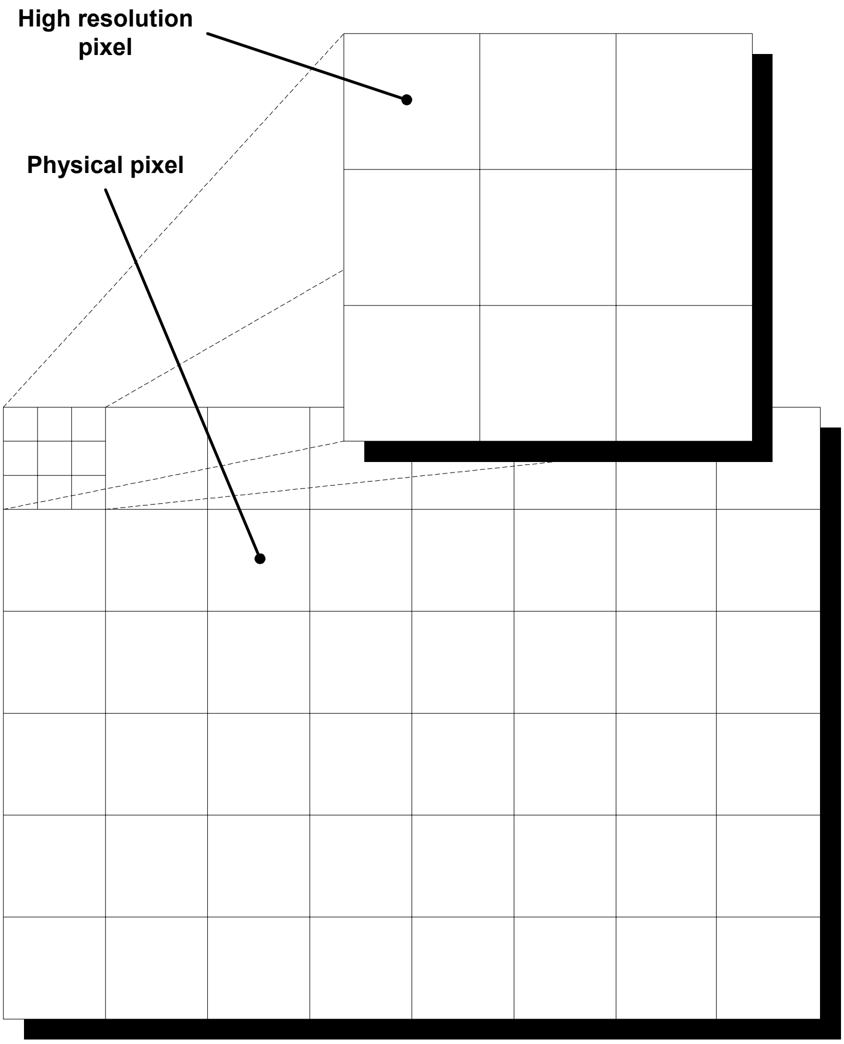
Antialiased fonts can be created using the Font Converter. The general purpose of using antialiased fonts is to improve the appearance of text. While the effect of using high-quality antialiasing will be visually more pleasing than low-quality antialiasing, computation time and memory consumption will increase proportionally. Low-quality (2bpp) fonts require twice the memory of non-antialiased (1bpp) fonts; high-quality (4bpp) fonts require four times the memory.

31.1.3 High-resolution coordinates

When drawing items using antialiasing, the same coordinates are used as for regular (non-antialiasing) drawing routines. This is the default mode. It is not required to consider the antialiasing factor in the function arguments. An antialiased line from (50, 100) to (100, 50) would be drawn with the following function call:

```
GUI_AA_DrawLine(50, 100, 100, 50);
```

The high-resolution feature of emWin lets you use the virtual space determined by the antialiasing factor and your display size. The advantage of using high-resolution coordinates is that items can be placed not only at physical positions of your display but also "between" them.
The virtual space of a high-resolution pixel is illustrated below based on an antialiasing factor of 3:



To draw a line from pixel (50, 100) to (100, 50) in high-resolution mode with antialiasing factor 3, you would write:

```
GUI_AA_DrawLine(150, 300, 300, 150);
```

High-resolution coordinates must be enabled with the routine GUI_AA_EnableHiRes(), and may be disabled with GUI_AA_DisableHiRes(). Both functions are explained later in the chapter. For example programs using the high-resolution feature, see the examples at the end of the chapter.

31.2 Antialiasing API

The table below lists the available routines in the antialiasing package, in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Description
Control functions	
GUI_AA_DisableHiRes()	Disables high-resolution coordinates.
GUI_AA_EnableHiRes()	Enables high-resolution coordinates.
GUI_AA_GetFactor()	Returns the current antialiasing factor.
GUI_AA_SetFactor()	Sets the current antialiasing factor.
Drawing functions	
GUI_AA_DrawArc()	Draws an antialiased arc.
GUI_AA_DrawLine()	Draws an antialiased line.
GUI_AA_DrawPolyOutline()	Draws the outline of an antialiased polygon of max. 10 points.
GUI_AA_DrawPolyOutlineEx()	Draws the outline of an antialiased polygon.
GUI_AA_DrawRoundedRect()	Draws the outline of an antialiased rectangle with rounded corners.
GUI_AA_DrawRoundedRectEx()	Draws the outline of an antialiased rectangle with rounded corners.
GUI_AA_FillCircle()	Draws a filled antialiased circle.
GUI_AA_FillEllipse()	Draws a filled antialiased ellipse.
GUI_AA_FillPolygon()	Draws a filled antialiased polygon.
GUI_AA_FillRoundedRect()	Draws a filled antialiased rectangle with rounded corners.
GUI_AA_FillRoundedRectEx()	Draws a filled antialiased rectangle with rounded corners.
GUI_AA_SetDrawMode()	Determines how the background color is fetched for mixing.

Table 31.2: Antialiasing API list

31.2.1 Control functions

GUI_AA_DisableHiRes()

Description

Disables high-resolution coordinates.

Prototype

```
void GUI_AA_DisableHiRes(void);
```

Additional information

High-resolution coordinates are disabled by default.

GUI_AA_EnableHiRes()

Description

Enables high-resolution coordinates.

Prototype

```
void GUI_AA_EnableHiRes(void);
```


GUI_AA_GetFactor()

Description

Returns the current antialiasing quality factor.

Prototype

```
int GUI_AA_GetFactor(void);
```

Return value

The current antialiasing factor.

GUI_AA_SetFactor()

Description

Sets the antialiasing quality factor.

Prototype

```
void GUI_AA_SetFactor(int Factor);
```

Parameter	Description
Factor	Antialiasing quality factor to use. Minimum: 1 (no antialiasing); default: 3; maximum: 6.

Table 31.3: GUI_AA_SetFactor() parameter list

Additional information

For good quality and performance, it is recommended to use an antialiasing quality factor of 2-4.

31.2.2 Drawing functions

GUI_AA_DrawArc()

Description

Displays an antialiased arc at a specified position in the current window, using the current pen size and the current pen shape.

Prototype

```
void GUI_AA_DrawArc(int x0, int y0, int rx, int ry, int a0, int a1);
```

Parameter	Description
x0	Horizontal position of the center.
y0	Vertical position of the center.
rx	Horizontal radius.
ry	Vertical radius.
a0	Starting angle (degrees).
a1	Ending angle (degrees).

Table 31.4: GUI_AA_DrawArc() parameter list

Limitations

Currently the `ry` parameter is not available. The `rx` parameter is used instead.

Additional information

If working in high-resolution mode, position and radius must be in high-resolution coordinates. Otherwise they must be specified in pixels.

GUI_AA_DrawLine()

Description

Displays an antialiased line at a specified position in the current window, using the current pen size and the current pen shape.

Prototype

```
void GUI_AA_DrawLine(int x0, int y0, int x1, int y1);
```

Parameter	Description
x0	X-starting position.
y0	Y-starting position.
x1	X-end position.
y1	Y-end position.

Table 31.5: GUI_AA_DrawLine() parameter list

Additional information

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

GUI_AA_DrawPolyOutline()

Description

Displays the outline of an antialiased polygon defined by a list of points, at a specified position in the current window and with a specified thickness. The number of points is limited to 10.

Prototype

```
void GUI_AA_DrawPolyOutline(const GUI_POINT * pPoint,
                            int               NumPoints,
                            int               Thickness,
                            int               x,
                            int               y);
```

Parameter	Description
pPoint	Pointer to the polygon to display.
NumPoints	Number of points specified in the list of points.
Thickness	Thickness of the outline.
x	X-position of origin.
y	Y-position of origin.

Table 31.6: GUI_AA_DrawPolyOutline() parameter list

Additional information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. The starting point must not be specified a second time as an endpoint.
If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.
Per default the number of points processed by this function is limited to 10. If the polygon consists of more than 10 points the function `GUI_AA_DrawPolyOutlineEx()` should be used.

Example

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

static GUI_POINT aPoints[] = {
  {  0,  0 },
  { 15, 30 },
  {  0, 20 },
  {-15, 30 }
};
```



```
void Sample(void) {
  GUI_AA_DrawPolyOutline(aPoints, countof(aPoints), 3, 150, 40);
}
```

Screenshot of above example

GUI_AA_DrawPolyOutlineEx()

Description

Displays the outline of an antialiased polygon defined by a list of points, at a specified position in the current window and with a specified thickness.

Prototype

```
void GUI_AA_DrawPolyOutlineEx(const GUI_POINT * pPoint,
                              int               NumPoints,
                              int               Thickness,
                              int               x,
                              int               y,
                              GUI_POINT       * pBuffer);
```

Parameter	Description
pPoint	Pointer to the polygon to display.
NumPoints	Number of points specified in the list of points.
Thickness	Thickness of the outline.
x	X-position of origin.
y	Y-position of origin.
pBuffer	Pointer to a buffer of GUI_POINT elements.

Table 31.7: GUI_AA_DrawPolyOutlineEx() parameter list

Additional information

The number of polygon points is not limited by this function. Internally the function needs a buffer of GUI_POINT elements for calculation purpose. The number of points of the buffer needs to be >= the number of points of the polygon.
For more details, refer to "GUI_AA_DrawPolyOutline()" on page 1024.

GUI_AA_DrawRoundedRect()

Description

Draws the outline of an antialiased rectangle with rounded corners using the current pen size.

Prototype

```
void GUI_AA_DrawRoundedRect(int x0, int y0, int x1, int y1, int r);
```

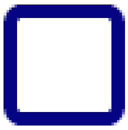
Parameter	Description
x0	X-position of the upper left corner.
y0	Y-position of the upper left corner.
x1	X-position of the lower right corner.
y1	Y-position of the lower right corner.
r	Radius to be used for the rounded corners.

Table 31.8: GUI_AA_DrawRoundedRect() parameter list

Example

```
#include "GUI.h"

void MainTask(void) {
  GUI_Init();
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetColor(GUI_DARKBLUE);
  GUI_SetPenSize(5);
  GUI_AA_DrawRoundedRect(10, 10, 50, 50, 5);
}
```

Screenshot of the example above



GUI_AA_DrawRoundedRectEx()

Description

Draws the outline of an antialiased rectangle with rounded corners using the current pen size.

Prototype

```
void GUI_AA_DrawRoundedRectEx(GUI_RECT * pRect, int r);
```

Parameter	Description
pRect	Pointer to the rectangle to draw.
r	Radius to be used for the rounded corners.

Table 31.9: GUI_AA_DrawRoundedRectEx() parameter list

Example

See "GUI_AA_DrawRoundedRect()" on page 1025.

GUI_AA_FillCircle()

Description

Displays a filled, antialiased circle at a specified position in the current window.

Prototype

```
void GUI_AA_FillCircle(int x0, int y0, int r);
```

Parameter	Description
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
r	Radius of the circle (half of the diameter).

Table 31.10: GUI_AA_FillCircle() parameter list

Additional information

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

GUI_AA_FillEllipse()

Description

Displays a filled, antialiased ellipse at a specified position in the current window.

Prototype

```
void GUI_AA_FillEllipse(int x0, int y0, int rx, int ry);
```

Parameter	Description
x0	X-position of the center of the ellipse in pixels of the client window.
y0	Y-position of the center of the ellipse in pixels of the client window.
rx	Radius in X of the ellipse.
ry	Radius in Y of the ellipse.

Table 31.11: GUI_AA_FillEllipse() parameter list

Additional information

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

GUI_AA_FillPolygon()

Description

Fills an antialiased polygon defined by a list of points, at a specified position in the current window.

Prototype

```
void GUI_AA_FillPolygon(const GUI_POINT * pPoint, int NumPoints,
                        int               x,      int y);
```

Parameter	Description
pPoint	Pointer to the polygon to display.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

Table 31.12: GUI_AA_FillPolygon() parameter list

Additional information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. The starting point must not be specified a second time as an endpoint.
If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

GUI_AA_FillRoundedRect()

Description

Draws a filled and antialiased rectangle with rounded corners.

Prototype

```
void GUI_AA_FillRoundedRect(int x0, int y0, int x1, int y1, int r);
```

Parameter	Description
x0	X-position of the upper left corner.
y0	Y-position of the upper left corner.
x1	X-position of the lower right corner.
y1	Y-position of the lower right corner.
r	Radius to be used for the rounded corners.

Table 31.13: GUI_AA_FillRoundedRect() parameter list

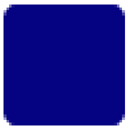
Example

```
#include "GUI.h"
```



```
void MainTask(void) {
  GUI_Init();
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetColor(GUI_DARKBLUE);
  GUI_AA_FillRoundedRect(10, 10, 54, 54, 5);
}
```

Screenshot of the example above



GUI_AA_FillRoundedRectEx()

Description

Draws a filled and antialiased rectangle with rounded corners.

Prototype

```
void GUI_AA_FillRoundedRectEx(GUI_RECT * pRect, int r);
```

Parameter	Description
pRect	Pointer to the rectangle to draw.
r	Radius to be used for the rounded corners.

Table 31.14: GUI_AA_FillRoundedRectEx() parameter list

Example

See “GUI_AA_FillRoundedRect()” on page 1027.

GUI_AA_SetDrawMode()

Description

This function determines how the background color is fetched for mixing.

Prototype

```
int GUI_AA_SetDrawMode(int Mode);
```

Parameter	Description
Mode	Mode to be used (see table below)

Table 31.15: GUI_AA_SetDrawMode() parameter list

Permitted values for parameter Mode	
GUI_AA_TRANS	Default behavior. Antialiased pixels are mixed with the current content of the frame buffer.
GUI_AA_NOTRANS	Antialiased pixels are mixed with the current background color set with GUI_SetBkColor().

Return value

0 on success. 1, if Mode did not contain a permitted value.

Additional information

The default behavior of antialiasing in emWin is mixing pixels with the current content of the frame buffer. But under certain circumstances using the currently set background color (GUI_SetBkColor()) for mixing may be an advantage. This makes it possible to redraw antialiased items completely without having to redraw the background.

31.3 Examples

Different antialiasing factors

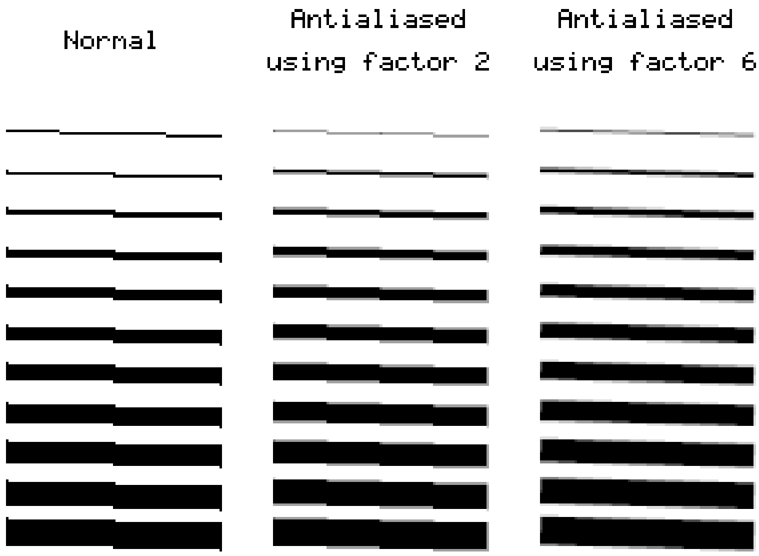
The following example creates diagonal lines with and without antialiasing. The source code is available as `AA_Lines.c` in the examples shipped with emWin.

```
/*********************************************************************
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                    emWin example code                              *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        : AA_Lines.c
Purpose     : Shows lines with different antialiasing qualities
----------------------------------------------------------------------
*/

#include "GUI.h"

/*******************************************************************
*
*        Show lines with different antialiasing qualities
*
********************************************************************
*/
static void DemoAntialiasing(void) {
  int i, x1, x2, y;

  y = 2;
  //
  // Set drawing attributes
  //
  GUI_SetColor(GUI_BLACK);
  GUI_SetBkColor(GUI_WHITE);
  GUI_SetPenShape(GUI_PS_FLAT);
  GUI_Clear();
  x1 = 10;
  x2 = 90;
  //
  // Draw lines without antialiasing
  //
  GUI_DispStringHCenterAt("\nNormal", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
  x1 = 110;
  x2 = 190;
  //
  // Draw lines with antialiasing quality factor 2
  //
  GUI_AA_SetFactor(2);
  GUI_DispStringHCenterAt("Antialiased\n\nusing factor 2", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
  x1 = 210;
  x2 = 290;
  //
  // Draw lines with antialiasing quality factor 6
  //
  GUI_AA_SetFactor(6);
  GUI_DispStringHCenterAt("Antialiased\n\nusing factor 6", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
}

/*******************************************************************
*
*        MainTask
*
********************************************************************
```



```
*/
void MainTask(void) {
  GUI_Init();
  DemoAntialiasing();
  while(1)
    GUI_Delay(100);
}
```

Screenshot of above example



Lines placed on high-resolution coordinates

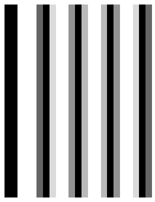
This example shows antialiased lines placed on high-resolution coordinates. It is available as `AA_HiResPixels.c`.

```
/*********************************************************************
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                    emWin example code                              *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        : AA_HiResPixels.c
Purpose     : Demonstrates high resolution pixels
----------------------------------------------------------------------
*/

#include "GUI.H"

/*******************************************************************
*
*        Show lines placed on high resolution pixels
*
********************************************************************
*/
static void ShowHiResPixels(void) {
  int i, Factor;

  Factor = 5;
  GUI_SetBkColor(GUI_WHITE);
  GUI_SetColor(GUI_BLACK);
  GUI_Clear();
  GUI_SetLBorder(50);
  GUI_DispStringAt("This example uses high resolution pixels.\n", 50, 10);
  GUI_DispString  ("Not only the physical pixels are used.\n");
  GUI_DispString  ("Enabling high resolution simulates more\n");
  GUI_DispString  ("pixels by using antialiasing.\n");
  GUI_DispString  ("Please take a look at the magnified output\n");
  GUI_DispString  ("to view the result.\n");
  GUI_SetPenSize(2);
  GUI_SetPenShape(GUI_PS_FLAT);
  GUI_AA_EnableHiRes();        /* Enable high resolution */
  GUI_AA_SetFactor(Factor);    /* Set quality factor */
```



```
  //
  // Drawing lines using high resolution pixels
  //
  for (i = 0; i < Factor; i++) {
    int x = (i + 1) * 5 * Factor + i - 1;
    GUI_AA_DrawLine(x, 50, x, 199);
  }
}

/*********************************************************************
*
*       MainTask
*
**********************************************************************
*/
void MainTask(void) {
  GUI_Init();
  ShowHiResPixels();
  while(1) {
    GUI_Delay(100);
  }
}
```

Magnified screenshot of above example



Moving pointer using high-resolution antialiasing

This example illustrates the use of high-resolution antialiasing by drawing a rotating pointer that turns 0.1 degrees with each step. There is no screenshot of this example because the effects of high-resolution antialiasing are only visible in the movement of the pointers. Without high-resolution the pointer appears to make short "jumps", whereas in high-resolution mode there is no apparent jumping.
The example can be found as `AA_HiResAntialiasing.c`.

```
/*********************************************************************
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                    emWin example code                              *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        : AA_HiResAntialiasing.c
Purpose     : Demonstrates high resolution antialiasing
----------------------------------------------------------------------
*/

#include "GUI.H"

/*********************************************************************
*
*       Data
*
**********************************************************************
*/
#define countof(Obj) (sizeof(Obj)/sizeof(Obj[0]))

static const GUI_POINT aPointer[] = {
  {  0,  3 },
  { 85,  1 },
  { 90,  0 },
  { 85, -1 },
  {  0, -3 }
};

static GUI_POINT aPointerHiRes[countof(aPointer)];
```

```
typedef struct {
  GUI_AUTODEV_INFO AutoInfo;
  GUI_POINT aPoints[countof(aPointer)];
  int Factor;
} PARAM;

/*******************************************************************
*
*        Drawing routines
*
********************************************************************
*/
static void DrawHiRes(void * p) {
  PARAM * pParam = (PARAM *)p;
  if (pParam->AutoInfo.DrawFixed) {
    GUI_ClearRect(0, 0, 99, 99);
  }
  GUI_AA_FillPolygon(pParam->aPoints,    countof(aPointer),
                     5 * pParam->Factor, 95 * pParam->Factor);
}

static void Draw(void * p) {
  PARAM * pParam = (PARAM *)p;
  if (pParam->AutoInfo.DrawFixed) {
    GUI_ClearRect(100, 0, 199, 99);
  }
  GUI_AA_FillPolygon(pParam->aPoints, countof(aPointer), 105, 95);
}

/*******************************************************************
*
*        Demonstrate high resolution by drawing rotating pointers
*
********************************************************************
*/
static void ShowHiresAntialiasing(void) {
  GUI_AUTODEV aAuto[2];
  PARAM       Param;
  int         i;

  Param.Factor = 3;
  GUI_DispStringHCenterAt("Using\nhigh\nresolution\nmode", 50, 120);
  GUI_DispStringHCenterAt("Not using\nhigh\nresolution\nmode", 150, 120);
  //
  // Create GUI_AUTODEV objects
  //
  for (i = 0; i < countof(aAuto); i++) {
    GUI_MEMDEV_CreateAuto(&aAuto[i]);
  }
  //
  // Calculate pointer for high resolution
  //
  for (i = 0; i < countof(aPointer); i++) {
    aPointerHiRes[i].x = aPointer[i].x * Param.Factor;
    aPointerHiRes[i].y = aPointer[i].y * Param.Factor;
  }
  GUI_AA_SetFactor(Param.Factor); /* Set antialiasing factor */
  while(1) {
    for (i = 0; i < 1800; i++) {
      float Angle = (i >= 900) ? 1800 - i : i;
      Angle *= 3.1415926f / 1800;
      //
      // Draw pointer with high resolution
      //
      GUI_AA_EnableHiRes();
      GUI_RotatePolygon(Param.aPoints, aPointerHiRes, countof(aPointer), Angle);
      GUI_MEMDEV_DrawAuto(&aAuto[0], &Param.AutoInfo, DrawHiRes, &Param);
      //
      // Draw pointer without high resolution
      //
      GUI_AA_DisableHiRes();
      GUI_RotatePolygon(Param.aPoints, aPointer, countof(aPointer), Angle);
      GUI_MEMDEV_DrawAuto(&aAuto[1], &Param.AutoInfo, Draw, &Param);
      GUI_Delay(2);
    }
  }
}

/*******************************************************************
*
*        MainTask
```



```
*
*******************************************************************
*/
void MainTask(void) {
  GUI_Init();
  ShowHiresAntialiasing();
}
```


Chapter 32

Language Support

Text written in a language like Arabic, Thai or Chinese contains characters, which are normally not part of the fonts shipped with emWin.
This chapter explains the basics like the Unicode standard, which defines all available characters worldwide and the UTF-8 encoding scheme, which is used by emWin to decode text with Unicode characters.
It also explains how to enable Arabic language support and how to render text with Shift-JIS (Japanese Industry Standard) encoding.

32.1 Unicode

Unicode is an international standard defined by the Unicode consortium. For each meaningful character or text element of all known cultures it contains a unique digital code point. Further it contains the bidirectional algorithm for right-to-left scripts like Hebrew and Arabic, which are also supported by emWin.
The Unicode Standard defines a codespace of 1,114,112 code points in the range from 0 to 0x10FFFF containing a repertoire of more than 110,000 characters covering more than 100 scripts. This codespace is divided into seventeen planes, numbered 0 to 16. emWin supports here the complete 'Basic Multilingual Plane' (BMP, plane 0) which covers the code points from 0x0000 to 0xFFFF. This BMP contains characters for almost all modern languages, and a large number of special characters. Characters outside the BMP are currently not supported.

32.1.1 UTF-8 encoding

ISO/IEC 10646-1 defines a multi-octet character set called the Universal Character Set (UCS) which encompasses most of the world's writing systems. Multi-octet characters, however, are not compatible with many current applications and protocols, and this has led to the development of a few UCS transformation formats (UTF), each with different characteristics.
UTF-8 has the characteristic of preserving the full ASCII range, providing compatibility with file systems, parsers and other software that rely on ASCII values but are transparent to other values.
In emWin, UTF-8 characters are encoded using sequences of 1 to 3 octets. If the high-order bit is set to 0, the remaining 7 bits being used to encode the character value. In a sequence of n octets, n>1, the initial octet has the n higher-order bits set to 1, followed by a bit set to 0. The remaining bit(s) of that octet contain bits from the value of the character to be encoded. The following octet(s) all have the higher-order bit set to 1 and the following bit set to 0, leaving 6 bits in each to contain bits from the character to be encoded.
The following table shows the encoding ranges:

Character range	UTF-8 Octet sequence
0000 - 007F	0xxxxxxx
0080 - 07FF	110xxxxx 10xxxxxx
0800 - FFFF	1110xxxx 10xxxxxx 10xxxxxx

Table 32.1: UTF-8 encoding

Encoding example

The text "Halöle" contains ASCII characters and European extensions. The following hex dump shows this text as UTF-8 encoded text:

```
48 61 6C C3 B6 6C 65
```

Programming examples

If we want to display a text containing non-ASCII characters, we can do this by manually computing the UTF-8 codes for the non-ASCII characters in the string.
However, if your compiler supports UTF-8 encoding (Sometimes called multi-byte encoding), even non-ASCII characters can be used directly in strings.

```
//
// Example using ASCII encoding:
//
GUI_UC_SetEncodeUTF8();        /* required only once to activate UTF-8*/
GUI_DispString("Hal\xc3\xb6le");

//
// Example using UTF-8 encoding:
//
GUI_UC_SetEncodeUTF8();        /* required only once to activate UTF-8*/
GUI_DispString("Halöle");
```

32.1.2 Unicode characters

The character output routine used by emWin (`GUI_DispChar()`) does always take an unsigned 16-bit value (U16) and has the basic ability to display a character defined by Unicode. It simply requires a font which contains the character you want to display.

32.1.3 UTF-8 strings

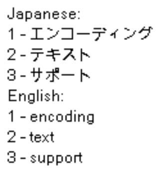
This is the most recommended way to display Unicode. You do not have to use special functions to do so. If UTF-8-encoding is enabled each function of emWin which handles with strings decodes the given text as UTF-8 text.

32.1.3.1 Using U2C.exe to convert UTF-8 text into C code

The `Tool` subdirectory of emWin contains the tool `U2C.exe` to convert UTF-8 text to C code. It reads an UTF-8 text file and creates a C file with C strings. The following steps show how to convert a text file into C strings and how to display them with emWin:

Step 1: Creating a UTF-8 text file

Save the text to be converted in UTF-8 format. You can use `Notepad.exe` to do this. Load the text under Notepad.exe:


Japanese:
1 - エンコーディング
2 - テキスト
3 - サポート
English:
1 - encoding
2 - text
3 - support

Choose "File/Save As...". The file dialog should contain a combo box to set the encoding format. Choose "UTF-8" and save the text file.

Step 2: Converting the text file into a C-code file

Start `U2C.exe`. After starting the program you need to select the text file to be converted. After selecting the text file the name of the C file should be selected. Output of `U2C.exe`:

```
"Japanese:"
"1 - \xe3\x82\xa8\xe3\x83\xb3\xe3\x82\xb3\xe3\x83\xbc"
    "\xe3\x83\x87\xe3\x82\xa3\xe3\x83\xb3\xe3\x82\xb0"
"2 - \xe3\x83\x86\xe3\x82\xad\xe3\x82\xb9\xe3\x83\x88"
"3 - \xe3\x82\xb5\xe3\x83\x9d\xe3\x83\xbc\xe3\x83\x88"
"English:"
"1 - encoding"
"2 - text"
"3 - support"
```

Step 3: Using the output in the application code

The following example shows how to display the UTF-8 text with emWin:

```
#include "GUI.h"

static const char * _apStrings[] = {
  "Japanese:",
  "1 - \xe3\x82\xa8\xe3\x83\xb3\xe3\x82\xb3\xe3\x83\xbc"
      "\xe3\x83\x87\xe3\x82\xa3\xe3\x83\xb3\xe3\x82\xb0",
  "2 - \xe3\x83\x86\xe3\x82\xad\xe3\x82\xb9\xe3\x83\x88",
  "3 - \xe3\x82\xb5\xe3\x83\x9d\xe3\x83\xbc\xe3\x83\x88",
  "English:",
  "1 - encoding",
  "2 - text",
  "3 - support"
};

void MainTask(void) {
  int i;
  GUI_Init();
  GUI_SetFont(&GUI_Font16_1HK);
  GUI_UC_SetEncodeUTF8();
  for (i = 0; i < GUI_COUNTOF(_apStrings); i++) {
    GUI_DispString(_apStrings[i]);
    GUI_DispNextLine();
  }
  while(1) {
    GUI_Delay(500);
  }
}
```

32.1.4 Unicode API

The table below lists the available routines in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Description
UTF-8 functions	
GUI_UC_ConvertUC2UTF8()	Converts a Unicode string into UTF-8 format.
GUI_UC_ConvertUTF82UC()	Converts a UTF-8 string into Unicode format.
GUI_UC_EnableBIDI()	Enables/Disables the support for bidirectional fonts.
GUI_UC_Encode()	Encodes the given character with the current encoding.
GUI_UC_GetCharCode()	Returns the decoded character.
GUI_UC_GetCharSize()	Returns the number of bytes used to encode the given character.
GUI_UC_SetEncodeNone()	Disables encoding.
GUI_UC_SetEncodeUTF8()	Enables UTF-8 encoding.
Double byte functions	
GUI_UC_DispString()	Displays a double byte string.

Table 32.2: Unicode API list

32.1.4.1 UTF-8 functions

GUI_UC_ConvertUC2UTF8()

Description

Converts the given double byte Unicode string into UTF-8 format.

Prototype

```
int GUI_UC_ConvertUC2UTF8(const U16 * s, int Len,
                          char * pBuffer, int BufferSize);
```

Parameter	Description
s	Pointer to Unicode string to be converted.
Len	Number of Unicode characters to be converted.
pBuffer	Pointer to a buffer to write in the result.
BufferSize	Buffer size in bytes.

Table 32.3: GUI_UC_ConvertUC2UTF8() parameter list

Return value

The function returns the number of bytes written to the buffer.

Additional information

UTF-8 encoded characters can use up to 3 bytes. To be on the save side the recommended buffer size is: Number of Unicode characters * 3.
If the buffer is not big enough for the whole result, the function returns when the buffer is full.

GUI_UC_ConvertUTF82UC()

Description

Converts the given UTF-8 string into Unicode format.

Prototype

```
int GUI_UC_ConvertUTF82UC(const char * s, int Len,
                          U16 * pBuffer, int BufferSize);
```

Parameter	Description
s	Pointer to UFT-8 string to be converted.
Len	Length in bytes of the string to be converted.
pBuffer	Pointer to a buffer to write in the result.
BufferSize	Buffer size in words.

Table 32.4: GUI_UC_ConvertUTF82UC() parameter list

Return value

The function returns the number of Unicode characters written to the buffer.

Additional information

If the buffer is not big enough for the whole result, the function returns when the buffer is full.

GUI_UC_EnableBIDI()

Description

This function enables support for bidirectional text.

Prototype

```
int GUI_UC_EnableBIDI(int OnOff);
```

Parameter	Description
OnOff	1 to enable BIDI support, 0 to disable it.

Table 32.5: GUI_UC_EnableBIDI() parameter list

Return value

The previous state of BIDI support.

Additional information

Once this function is linked approximately 60 KBytes of ROM are additionally used.

GUI_UC_Encode()

Description

This function encodes a given character with the current encoding settings.

Prototype

```
int GUI_UC_Encode(char * s, U16 Char);
```

Parameter	Description
s	Pointer to a buffer to store the encoded character.
Char	Character to be encoded.

Table 32.6: GUI_UC_Encode() parameter list

Return value

The number of bytes stored to the buffer.

Additional information

The function assumes that the buffer has at least 3 bytes for the result.

GUI_UC_GetCharCode()

Description

This function decodes a character from a given text.

Prototype

```
U16 GUI_UC_GetCharCode(const char * s);
```

Parameter	Description
s	Pointer to the text to be encoded.

Table 32.7: GUI_UC_GetCharCode() parameter list

Return value

The encoded character.

Related topics

`GUI_UC_GetCharSize()`

GUI_UC_GetCharSize()

Description

This function returns the number of bytes used to encode the given character.

Prototype

```
int GUI_UC_GetCharSize(const char * s);
```

Parameter	Description
s	Pointer to the text to be encoded.

Table 32.8: GUI_UC_GetCharSize() parameter list

Return value

Number of bytes used to encode the given character

Additional information

This function is used to determine how much bytes a pointer has to be incremented to point to the next character. The following example shows how to use the function:

```
static void _Display2Characters(const char * pText) {
  int Size;
  U16 Character;

  Size       = GUI_UC_GetCharSize(pText);  // Size to increment pointer
  Character  = GUI_UC_GetCharCode(pText);  // Get first character code
  GUI_DispChar(Character);                 // Display first character
  pText     += Size;                       // Increment pointer
  Character  = GUI_UC_GetCharCode(pText);  // Get next character code
  GUI_DispChar(Character);                 // Display second character
}
```

GUI_UC_SetEncodeNone()

Description

Disables character encoding.

Prototype

```
void GUI_UC_SetEncodeNone(void);
```

Additional information

After calling this function each byte of a text will be handled as one character. This is the default behavior of emWin.

GUI_UC_SetEncodeUTF8()

Description

Enables UTF-8 encoding.

Prototype

```
void GUI_UC_SetEncodeUTF8(void);
```

Additional information

After calling `GUI_UC_SetEncodeUTF8` each string related routine of emWin encodes a given sting in accordance to the UTF-8 transformation.

32.1.4.2 Double byte functions

GUI_UC_DispString()

Description

This function displays the given double byte string.

Prototype

```
void GUI_UC_DispString(const U16 * s);
```

Parameter	Description
s	Pointer to double byte string.

Table 32.9: GUI_UC_DispString() parameter list

Additional information

If you need to display double byte strings you should use this function. Each character has to be defined by a 16 bit value.

32.2 Text- and language resource files

To be able to change the text of an application without modifying one line of code the text- and language resource file API functions can be used. They offer the possibility to use one or more simple text files or one CSV (**C**omma **S**eparated **V**alue) file containing text in multiple languages. These files can reside in addressable RAM or at any non addressable medium like NAND flash or a file system.

32.2.1 Unicode support

If the used range of characters exceeds the ASCII set the text files should contain UTF-8 text. Other encodings like UC16 are not supported by this module.

32.2.2 Loading files from RAM

When using the files directly from RAM emWin does not allocate the required strings again. It uses the RAM location of the files directly. But because text- and CSV files do not contain zero delimited strings, emWin modifies the given text by replacing the line delimiters (CRLF) (text files) or field delimiters (CSV files) by a zero byte. Therefor the files have to reside in RAM, not in ROM.

32.2.3 Loading files from non addressable areas

It is also possible to use the files from non addressable areas or any other location in ROM. In these cases emWin uses a `GetData` function for getting the file data. In the first step (`GUI_LANG_LoadTextEx(), GUI_LANG_LoadCSVEx()`) emWin only remembers size and file offset of the text locations within the files. Only when accessing the text with `GUI_LANG_GetText()` the text will be allocated in RAM, read from the file and converted in a legal zero delimited string.

32.2.4 Rules for CSV files

Because the term 'CSV file' does not exactly determines the file format, here are the rules which have to be observed:

- Each record is located on a separate line, delimited by a line break (CRLF).
- The last record in the file may or may not have an ending line break.
- Within each record, there may be one or more fields, separated by delimiters. Each line should contain the same number of fields throughout the file. Spaces are considered part of a field. The last field in the record must not be followed by a delimiter.
- Default field delimiter is a comma. This may be changed using the function `GUI_LANG_SetSep()`.
- Each field may or may not be enclosed in double quotes. If fields are not enclosed with double quotes, then double quotes may not appear inside the fields.
- Fields containing line breaks (CRLF), double quotes, and commas should be enclosed in double-quotes.
- If double-quotes are used to enclose fields, then a double-quote appearing inside a field must be escaped by preceding it with another double quote.

32.2.5 Rules for text files

A text file is a simple file where each line contains one text element. Rules to be observed:

- Each line contains one text item.
- Each line must be delimited by a line break (CRLF).
- Text items containing line breaks are not supported.

32.2.6 Text- and language resource file API

The table below shows the available routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
	Text file functions
GUI_LANG_LoadText()	Loads a simple text file from RAM.
GUI_LANG_LoadTextEx()	Loads a simple text file from a non addressable area.
	CSV file functions
GUI_LANG_LoadCSV()	Loads a CSV file from RAM.
GUI_LANG_LoadCSVEx()	Loads a CSV file from a non addressable area.
	Common functions
GUI_LANG_GetNumItems()	Returns the number of items available for the given language.
GUI_LANG_GetText()	Returns a pointer to the requested text in the current language.
GUI_LANG_GetTextBuffered()	Copies the requested text of the current language into the given buffer.
GUI_LANG_GetTextBufferedEx()	Copies the requested text of the given language into the given buffer.
GUI_LANG_GetTextEx()	Returns a pointer to the requested text.
GUI_LANG_SetLang()	Sets the current language.
GUI_LANG_SetMaxNumLang()	Sets the maximum of available languages. Default is 10.
GUI_LANG_SetSep()	Sets the separator to be used for reading CSV files.

Table 32.10: Text- and language resource file API list

GUI_LANG_LoadText()

Description

Loads a text file from a RAM location.

Prototype

```
int GUI_LANG_LoadText(U8 * pFileData, U32 FileSize, int IndexLang);
```

Parameter	Description
pFileData	Pointer to the first byte of the file.
FileSize	Size of the given file in bytes.
IndexLang	Index of the language.

Table 32.11: GUI_LANG_LoadText() parameter list

Additional information

The given file needs to reside in RAM. As explained at the beginning of the chapter emWin converts the given text items into zero delimited strings.

GUI_LANG_LoadTextEx()

Description

Loads a text file using the given GetData function from any area.

Prototype

```
int GUI_LANG_LoadTextEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                        int IndexLang);
```


Parameter	Description
pfGetData	Pointer to a _GetData() function to be used for file access.
p	Pointer passed to the _GetData() function.
IndexLang	Index of the language.

Table 32.12: GUI_LANG_LoadTextEx() parameter list

Additional information

Data is accessed by the given GetData function. The pointer p can be used by the application.

Prototype of the 'GetData' function

```
int GUI_GET_DATA_FUNC(void * p, const U8 ** ppData, unsigned NumBytesReq,
                      U32 Off);
```

Parameter	Description
p	Application defined void pointer.
ppData	The location the pointer points to has to be filled by the 'GetData' function.
NumBytesReq	Number of requested bytes.
Off	Offset to be used to address the requested bytes within the file.

Table 32.13: GUI_GET_DATA_FUNC() parameter list

Sample

The following shows a sample implementation of the GetData function for WIN32:

```
static int _GetData(void * pVoid, const U8 ** ppData, unsigned NumBytes, U32 Off) {
  DWORD  NumBytesRead;
  HANDLE hFile;
  U8   * pData;

  pData = (U8 *)*ppData;
  hFile = *(HANDLE *)pVoid;
  if (SetFilePointer(hFile, Off, 0, FILE_BEGIN) == 0xFFFFFFFF) {
    return 0;
  }
  if (!ReadFile(hFile, pData, NumBytes, &NumBytesRead, 0)) {
    return 0;
  }
  if (NumBytesRead != NumBytes) {
    return 0;
  }
  return NumBytesRead;
}
```

GUI_LANG_LoadCSV()

Description

Loads a CSV file from a RAM location.

Prototype

```
int GUI_LANG_LoadCSV(char * pFileData, U32 FileSize);
```

Parameter	Description
pFileData	Pointer to the first byte of the file.
FileSize	Size of the given file in bytes.

Table 32.14: GUI_LANG_LoadCSV() parameter list

Return value

The function returns the number of available languages of the given file.

Additional information

The given file needs to reside in RAM. As explained at the beginning of the chapter emWin converts the given text items to zero delimited strings.
This function call first deletes all existing text resources. It is not possible to use a text file including one language and a CSV file including further languages. Either text files or CSV files should be used.

GUI_LANG_LoadCSVEx()

Description

Loads a CSV file from any location by using a `GetData` function.

Prototype

```
int GUI_LANG_LoadCSVEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Description
pfGetData	Pointer to a _GetData() function to be used for file access.
p	Pointer passed to the _GetData() function.

Table 32.15: GUI_LANG_LoadCSVEx() parameter list

Return value

The function returns the number of available languages.

Additional information

As explained at the beginning of the chapter emWin converts the given text items to zero delimited strings.
This function call first deletes all existing text resources. It is not possible to use a text file including one language and a CSV file including further languages. Either text files or CSV files should be used.

GUI_LANG_GetNumItems()

Description

Returns the number of available text items of the given language.

Prototype

```
int GUI_LANG_GetNumItems(int IndexLang);
```

Parameter	Description
IndexLang	Index of the given language.

Table 32.16: GUI_LANG_GetNumItems() parameter list

Return value

Number of available text items of the given language.

GUI_LANG_GetText()

Description

Returns a pointer to the requested text item of the current language.

Prototype

```
const char * GUI_LANG_GetText(int IndexText);
```

Parameter	Description
IndexText	Index of the text item to be returned.

Table 32.17: GUI_LANG_GetText() parameter list

Return value

Pointer to the requested text item.

Additional information

If a `GetData` function is used, the first time a text item is requested it will be allocated, read and converted once. In case of using a `GetData` function this could save memory if not all text items are used by the application.

GUI_LANG_GetTextBuffered()

Description

Copies the requested text of the current language into the given buffer.

Prototype

```
int GUI_LANG_GetTextBuffered(int IndexText, char * pBuffer,
                             int SizeOfBuffer);
```

Parameter	Description
IndexText	Index of the text item to be returned.
pBuffer	Pointer to an application defined buffer.
SizeOfBuffer	Size of the application defined buffer.

Table 32.18: GUI_LANG_GetTextBuffered() parameter list

Return value

0, on success.
1, if the text could not be found.

GUI_LANG_GetTextBufferedEx()

Description

Copies the requested text of the given language into the given buffer.

Prototype

```
int GUI_LANG_GetTextBufferedEx(int    IndexText, int IndexLang,
                               char * pBuffer,   int SizeOfBuffer);
```

Parameter	Description
IndexText	Index of the text item to be returned.
IndexLang	Index of the language.
pBuffer	Pointer to an application defined buffer.
SizeOfBuffer	Size of the application defined buffer.

Table 32.19: GUI_LANG_GetTextBufferedEx() parameter list

Return value

0, on success.
1, if the text could not be found.

GUI_LANG_GetTextEx()

Description

Returns a pointer to the requested text item.

Prototype

```
const char * GUI_LANG_GetTextEx(int IndexText, int IndexLang);
```

Parameter	Description
IndexText	Index of the text item to be returned.
IndexLang	Index of the requested language.

Table 32.20: GUI_LANG_GetTextEx() parameter list

Return value

Pointer to the requested text item.

Additional information

If a GetData function is used, the first time a text item is requested it will be allocated, read and converted once. In case of using a GetData function this could save memory if not all text items are used by the application.

GUI_LANG_SetLang()

Description

Sets the language to be used by the function GUI_LANG_GetText().

Prototype

```
int GUI_LANG_SetLang(int IndexLang);
```

Parameter	Description
IndexLang	Index of the language to be used.

Table 32.21: GUI_LANG_SetLang() parameter list

Return value

Previous index of the language.

GUI_LANG_SetMaxNumLang()

Description

Sets the maximum number of languages to be used.

Prototype

```
unsigned GUI_LANG_SetMaxNumLang(unsigned MaxNumLang);
```

Parameter	Description
MaxNumLang	Maximum number of languages

Table 32.22: GUI_LANG_SetMaxNumLang() parameter list

Return value

Previous maximum number of languages.

Additional information

This function has to be called before any other function of the language module is called. A good place for the function call would be GUI_X_Config().

GUI_LANG_SetSep()

Description

Sets the separator to be used when reading a CSV file.

Prototype

```
U16 GUI_LANG_SetSep(U16 Sep);
```

Parameter	Description
Sep	Separator to be used for CSV files.

Table 32.23: GUI_LANG_SetSep() parameter list

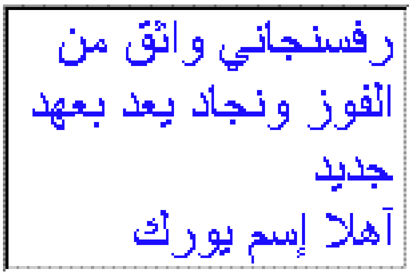
Return value

Previously used separator.

Additional information

The default separator is a comma. Some applications use TABs or semicolons as separator. This function can be used to change the separator. It does not check if the given separator makes sense. So it is the applications responsibility to set the right value. The function has no effect on reading text files.

32.3 Arabic support



The basic difference between western languages and Arabic is, that Arabic scripts are written from the right to the left and that it does not know uppercase and lowercase characters. Further the character codes of the text are not identical with the character index in the font file used to render the character, because the notation forms of the characters depend on the positions in the text.

32.3.1 Notation forms

The Arabic base character set is defined in the Unicode standard within the range from 0x0600 to 0x06FF. Unfortunately these character codes can not directly be used to get the character of the font for drawing it, because the notation form depends on the character position in the text. One character can have up to 4 different notation forms:

- One, if it is at the beginning of a word (initial)
- One, if it is at the end of a word (final)
- One, if it is in the middle of a word (medial)
- One, if the character stands alone (isolated)

But not each character is allowed to be joined to the left and to the right (double-joined). The character 'Hamza' for example always needs to be separated and 'Alef' is only allowed at the end or separated. Character combinations of the letters 'Lam' and 'Alef' should be transformed to a 'Ligature'. This means one character substitutionally for the combination of 'Lam' and 'Alef'.
The above description shows, that the notation form is normally not identical to the character code of the text. The following table shows how emWin transforms the characters to the notation form in dependence of the text position:

Base	Isolated	Final	Initial	Medial	Character
0x0621	0xFE80	-	-	-	Hamza
0x0622	0xFE81	0xFE82	-	-	Alef with Madda above
0x0623	0xFE83	0xFE84	-	-	Alef with Hamza above
0x0624	0xFE85	0xFE86	-	-	Waw with Hamza above
0x0625	0xFE87	0xFE88	-	-	Alef with Hamza below
0x0626	0xFE89	0xFE8A	0xFE8B	0xFE8C	Yeh with Hamza above
0x0627	0xFE8D	0xFE8E	-	-	Alef
0x0628	0xFE8F	0xFE90	0xFE91	0xFE92	Beh
0x0629	0xFE93	0xFE94	-	-	Teh Marbuta
0x062A	0xFE95	0xFE96	0xFE97	0xFE98	Teh
0x062B	0xFE99	0xFE9A	0xFE9B	0xFE9C	Theh
0x062C	0xFE9D	0xFE9E	0xFE9F	0xFEA0	Jeem
0x062D	0xFEA1	0xFEA2	0xFEA3	0xFEA4	Hah
0x062E	0xFEA5	0xFEA6	0xFEA7	0xFEA8	Khah
0x062F	0xFEA9	0xFEAA	-	-	Dal
0x0630	0xFEAB	0xFEAC	-	-	Thal
0x0631	0xFEAD	0xFEAE	-	-	Reh

Table 32.24: Notation forms

Base	Isolated	Final	Initial	Medial	Character
0x0632	0xFEAF	0xFEB0	-	-	Zain
0x0633	0xFEB1	0xFEB2	0xFEB3	0xFEB4	Seen
0x0634	0xFEB5	0xFEB6	0xFEB7	0xFEB8	Sheen
0x0635	0xFEB9	0xFEBA	0xFEBB	0xFEBC	Sad
0x0636	0xFEBD	0xFEBE	0xFEBF	0xFEC0	Dad
0x0637	0xFEC1	0xFEC2	0xFEC3	0xFEC4	Tah
0x0638	0xFEC5	0xFEC6	0xFEC7	0xFEC8	Zah
0x0639	0xFEC9	0xFECA	0xFECB	0xFECC	Ain
0x063A	0xFECD	0xFECE	0xFECF	0xFED0	Ghain
0x0641	0xFED1	0xFED2	0xFED3	0xFED4	Feh
0x0642	0xFED5	0xFED6	0xFED7	0xFED8	Qaf
0x0643	0xFED9	0xFEDA	0xFEDB	0xFEDC	Kaf
0x0644	0xFEDD	0xFEDE	0xFEDF	0xFEE0	Lam
0x0645	0xFEE1	0xFEE2	0xFEE3	0xFEE4	Meem
0x0646	0xFEE5	0xFEE6	0xFEE7	0xFEE8	Noon
0x0647	0xFEE9	0xFEEA	0xFEEB	0xFEEC	Heh
0x0648	0xFEED	0xFEEE	-	-	Waw
0x0649	0xFEEF	0xFEF0	-	-	Alef Maksura
0x064A	0xFEF1	0xFEF2	0xFEF3	0xFEF4	Yeh
0x067E	0xFB56	0xFB57	0xFB58	0xFB59	Peh
0x0686	0xFB7A	0xFB7B	0xFB7C	0xFB7D	Tcheh
0x0698	0xFB8A	0xFB8B	-	-	Jeh
0x06A9	0xFB8E	0xFB8F	0xFB90	0xFB91	Keheh
0x06AF	0xFB92	0xFB93	0xFB94	0xFB95	Gaf
0x06CC	0xFBFC	0xFBFD	0xFBFE	0xFBFF	Farsi Yeh

Table 32.24: Notation forms

32.3.2 Ligatures

Character combinations of ’Lam’ and ’Alef’ needs to be transformed to ligatures. The following table shows how emWin transforms these combinations into ligatures, if the first letter is a ’Lam’ (code 0x0644):

Second letter	Ligature (final)	Ligature (elsewhere)
0x622, Alef with Madda above	0xFEF6	0xFEF5
0x623, Alef with Hamza above	0xFEF8	0xFEF7
0x625, Alef with Hamza below	0xFEFA	0xFEF9
0x627, Alef	0xFEFC	0xFEFB

Table 32.25: Ligatures

32.3.3 Bidirectional text alignment

As mentioned above Arabic is written from the right to the left (RTL). But if for example the Arabic text contains numbers build of more than one digit these numbers should be written from left to right. And if Arabic text is mixed with European text a couple of further rules need to be followed to get the right visual alignment of the text.

The Unicode consortium has defined these rules in the Unicode standard. If bidirectional text support is enabled, emWin follows up most of these rules to get the right visual order before drawing the text.

emWin also supports mirroring of neutral characters in RTL aligned text. This is important if for example Arabic text contains parenthesis. The mirroring is done by replacing the code of the character to be mirrored with the code of a mirror partner

whose image fits to the mirrored image. This is done by a fast way using a table containing all characters with existing mirror partners. Note that support for mirroring further characters is not supported.
The following example shows how bidirectional text is rendered by emWin:

UTF-8 text	Rendering
`\xd8\xb9\xd9\x84\xd8\xa7 1, 2, 345 \xd8\xba\xd9\x86\xd9\x8a XYZ \xd8\xa3\xd9\x86\xd8\xa7`	علا 1, 2, 345 غني XYZ أنا

Table 32.26: Bidirectional text alignment

32.3.4 Requirements

Arabic support is part of the emWin basic package. emWin standard fonts do not contain Arabic characters. Font files containing Arabic characters can be created using the Font Converter.

Memory

The bidirectional text alignment and Arabic character transformation uses app. 60 KB of ROM and app. 800 bytes of additional stack.

32.3.5 How to enable Arabic support

Per default emWin writes text always from the left to the right and there will be no Arabic character transformation as described above. To enable support for bidirectional text and Arabic character transformation, add the following line to your application:

```
GUI_UC_EnableBIDI(1);
```

If enabled, emWin follows the rules of the bidirectional algorithm, described by the Unicode consortium, to get the right visual order before drawing text.

32.3.6 Example

The `Sample` folder contains the example `FONT_Arabic`, which shows how to draw Arabic text. It contains an emWin font with Arabic characters and some small Arabic text examples.

32.3.7 Font files used with Arabic text

Font files used to render Arabic text need to include at least all characters defined in the ’Arabic’ range 0x600-0x6FF and the notation forms and ligatures listed in the tables of this chapter.

32.4 Thai language support

Nice to meet you.

ยินดีที่ได้รู้จัก

The Thai alphabet uses 44 consonants and 15 basic vowel characters. These are horizontally placed, left to right, with no intervening space, to form syllables, words, and sentences. Vowels are written above, below, before, or after the consonant they modify, although the consonant always sounds first when the syllable is spoken. The vowel characters (and a few consonants) can be combined in various ways to produce numerous compound vowels (diphthongs and triphthongs).

32.4.1 Requirements

As explained above the Thai language makes an extensive usage of compound characters. To be able to draw compound characters in emWin, a new font type is needed, which contains all required character information like the image size, image position and cursor incrementation value. From version 4.00 emWin supports a new font type with this information. This also means that older font types can not be used to draw Thai text.
Note that the standard fonts of emWin does not contain font files with Thai characters. To create a Thai font file, the Font Converter of version 3.04 or newer is required.

Memory

The Thai language support needs no additional ROM or RAM.

32.4.2 How to enable Thai support

Thai support does not need to be enabled by a configuration switch. The only thing required to draw Thai text is a font file of type 'Extended' created with the Font Converter from version 3.04 or newer.

32.4.3 Example

The `Sample` folder contains the example `FONT_ThaiText.c`, which shows how to draw Thai text. It contains an emWin font with Thai characters and some small Thai text examples.

32.4.4 Font files used with Thai text

Font files used to render Thai text need to include at least all characters defined in the 'Thai' range 0xE00-0xE7F.

32.5 Shift JIS support

Shift JIS (Japanese Industry Standard) is a character encoding method for the Japanese language. It is the most common Japanese encoding method. Shift JIS encoding makes generous use of 8-bit characters, and the value of the first byte is used to distinguish single- and multiple-byte characters.
The Shift JIS support of emWin is only needed if text with Shift JIS encoding needs to be rendered.
You need no special function calls to draw a Shift JIS string. The main requirement is a font file which contains the Shift JIS characters.

32.5.1 Creating Shift JIS fonts

The Font Converter can generate a Shift JIS font for emWin from any Windows font. When using a Shift JIS font, the functions used to display Shift JIS characters are linked automatically with the library.
Detailed information on how to create Shift-JIS fonts and implement them in a project can be found in the chapter “Font Converter” on page 253.

32.6 Limitations

Currently emWin is not able to make text transitions required for drawing Devanagari and similar scripts. It does not contain an engine for complete complex script support. Because of this only RTF and Arabic transitions are supported.

Chapter 33

Display drivers

A display driver supports a particular family of display controllers and all displays which are connected to one or more of these controllers. The drivers can be configured by modifying their configuration files whereas the driver itself does not need to be modified. The configuration files contain all required information for the driver including how the hardware is accessed and how the controller(s) are connected to the display.
This chapter provides an overview of the display drivers available for emWin. It explains the following in terms of each driver:

- Which display controllers can be accessed, as well as supported color depths and types of interfaces.
- RAM requirements.
- Driver specific functions.
- How to access the hardware.
- Special configuration switches.
- Special requirements for particular display controllers.

33.1 Available display drivers

Since emWin V5 the driver interface has changed. Old display drivers, developed for emWin V4 or earlier, are not longer supported.
The display driver interface was changed in order to be able to configure drivers at run-time. This was required because emWin is often used as a precompiled library which should not have to be changed when using a different display.

Warning: Creating a precompiled library including the source files of a compile-time configurable driver precludes configurability using the library.

To be able to support as many display controllers as possible in a short period, we migrated some of the older drivers to the new interface. Not all migrated display drivers are run-time configurable. The listings below show the sets of available run-time and compile-time configurable display drivers.

33.1.1 Driver file naming convention

All files belonging to the same display driver begin with the name of the driver. So all files called `<DriverName>*.*` describe the whole driver.

Example

The following files describe the `GUIDRV_IST3088` display driver:

- `GUIDRV_IST3088.c`
- `GUIDRV_IST3088.h`
- `GUIDRV_IST3088_4.c`
- `GUIDRV_IST3088_Private.h`
- `GUIDRV_IST3088_X_4.c`

33.1.2 Run-time configurable drivers

The following table lists the currently available run-time configurable drivers developed for the current interface of emWin:

Driver	Supported display controller / Purpose of driver	Supported bits/pixel
GUIDRV_BitPlains	This driver can be used for solutions without display controller. It manages separate 'bitplains' for each color bit. Initially it has been developed to support a solution for an R32C/111 which drives a TFT display without display controller. It can be used for each solution which requires the color bits in separate plains.	1 - 8
GUIDRV_DCache	Cache driver for managing a double cache. It manages the cache data separately from the driver and converts the data line by line immediately before a drawing operation is required. This driver makes it possible to use for example a 16bpp display driver in 1bpp mode with a cache which only requires 1 bit per pixel.	1 (could be enhanced on demand)
GUIDRV_Dist	This driver supports displays with multiple controllers	Depends on the actual display drivers.
GUIDRV_FlexColor	Epson S1D19122 FocalTech FT1509 Himax HX8353, HX8325A, HX8357, HX8340, HX8347, HX8352A, HX8352B, HX8301 Hitachi HD66772 Ilitek ILI9320, ILI9325, ILI9328, ILI9335, ILI9338, ILI9340, ILI9341, ILI9342, ILI9163, ILI9481, ILI9486, ILI9488, ILI9220, ILI9221 LG Electronics LGDP4531, LGDP4551, LGDP4525 Novatek NT39122 OriseTech SPFD5408, SPFD54124C, SPFD5414D Raio RA8870, RA8875 Renesas R61505, R61516, R61526, R61580 Samsung S6E63D6, S6D0117 Sitronix ST7628, ST7637, ST7687, ST7735, ST7712 Solomon SSD1284, SSD1289, SSD1298, SSD1355, SSD2119, SSD1963, SSD1961, SSD1351 Syncoam SEPS525	16, 18
GUIDRV_IST3088	Integrated Solutions Technology IST3088, IST3257	4
GUIDRV_Lin	This driver supports every display controller with linear addressable video memory with a direct (full bus) interface. This means that the video RAM is directly addressable by the address lines of the CPU. The driver contains no controller specific code. So it can also be used for solutions without display controller which require a driver which only manages the video RAM.	1, 2, 4, 8, 16, 24, 32
GUIDRV_S1D13748	Epson S1D13748	16
GUIDRV_S1D13781	Epson S1D13781	8
GUIDRV_S1D15G00	Epson S1D15G00	12
GUIDRV_SLin	Epson S1D13700, S1D13305 (indirect interface only!) RAIO 8835 Solomon SSD1848 Toshiba T6963 UltraChip UC1617	1, 2

Table 33.1: Run-time configurable drivers

Driver	Supported display controller / Purpose of driver	Supported bits/pixel
GUIDRV_SPage	Avant Electronics SBN0064G Epson S1D15605, S1D15606, S1D15607, S1D15608, S1D15705, S1D15710, S1D15714, S1D15E05, S1D15E06, S1D15719, S1D15721 Hitachi HD61202 Integrated Solutions Technology IST3020 New Japan Radio Company NJU6676 Novatek NT7502, NT7534, NT7538, NT75451 Samsung S6B0108 (KS0108), S6B0713, S6B0719, S6B0724, S6B1713 Sino Wealth SH1101A Sitronix ST7522, ST75256, ST7565, ST7567, ST7591 Solomon SSD1303, SSD1305, SSD1805, SSD1815, SSD1821 Sunplus SPLC501C UltraChip UC1601, UC1606, UC1608, UC1611, UC1638, UC1701	1, 2, 4
GUIDRV_SSD1926	Solomon SSD1926	8
GUIDRV_UC1698G	UltraChip UC1698G	5

Table 33.1: Run-time configurable drivers

33.1.3 Compile-time configurable drivers

The following table lists the currently available drivers which has already been migrated to the current version of emWin:

Driver	Supported display controller / Purpose of driver	Supported bits/pixel
GUIDRV_CompactColor_16	Ampire FSA506 Epson S1D13742, S1D13743, S1D19122 FocalTech FT1509 Himax HX8301, HX8312A, HX8325A, HX8340, HX8347, HX8352, HX8352B, HX8353 Hitachi HD66766, HD66772, HD66789 Ilitek ILI9161, ILI9220, ILI9221, ILI9320, ILI9325, ILI9326, ILI9328, ILI9342, ILI9481 LG Electronics LGDP4531, LGDP4551 MagnaChip D54E4PA7551 Novatek NT39122, NT7573 OriseTech SPFD5408, SPFD54124C, SPFD5414D, SPFD5420A Renesas R61505, R61509, R61516, R61526, R61580, R63401 Samsung S6D0110A, S6D0117, S6D0128, S6D0129, S6D04H0 Sharp LCY-A06003, LR38825 Sitronix ST7628, ST7637, ST7712, ST7715, ST7735, ST7787, ST7789 Solomon SSD1284, SSD1289, SSD1298, SSD1355, SSD1961, SSD1963, SSD2119 Toshiba JBT6K71	16
GUIDRV_Fujitsu_16	Fujitsu MB87J2020 (Jasmine) Fujitsu MB87J2120 (Lavender)	1, 2, 4, 8, 16
GUIDRV_Page1bpp	Epson S1D10605, S1D15605, S1D15705, S1D15710, S1D15714, S1D15721, S1D15E05, S1D15E06, SED1520, SED1560, SED1565, SED1566, SED1567, SED1568, SED1569, SED1575 Hitachi HD61202 IST IST3020 New Japan Radio Company NJU6676, NJU6679 Novatek NT7502, NT7534, NT7538, NT75451 Philips PCF8810, PCF8811, PCF8535, PCD8544 Samsung KS0108B, KS0713, KS0724, S6B0108B, S6B0713, S6B0719, S6B0724, S6B1713 Sino Wealth SH1101A Sitronix ST7522, ST7565, ST7567 Solomon SSD1303, SSD1805, SSD1815, SSD1821 ST Microelectronics ST7548, STE2001, STE2002 Sunplus SPLC501C UltraChip UC1601, UC1606, UC1608, UC1701	1
GUIDRV_07X1	Novatek NT7506, NT7508 Samsung KS0711, KS0741, S6B0711, S6B0741 Sitronix ST7541, ST7571 Solomon SSD1854 ST Microelectronics STE2010 Tomato TL0350A	2
GUIDRV_1611	Epson S1D15719, S1D15E05, S1D15E06 UltraChip UC1610 UltraChip UC1611, UC1611s	2 2 4
GUIDRV_6331	Samsung S6B33B0X, S6B33B1X, S6B33B2X	16
GUIDRV_7528	Sitronix ST7528	4
GUIDRV_7529	Sitronix ST7529	1, 4, 5

Table 33.2: Compile-time configurable drivers

33.1.4 Special purpose drivers

The basic package contains a driver which does not support a specific display controller. It can be used as template for a new driver or for measurement purpose:

Driver	LCD Controller	Supported bits/pixel
GUIDRV_Template	Driver template. Can be used as a starting point for writing a new driver. Part of the basic package	-

Table 33.3: Special purpose drivers

33.2 CPU / Display controller interface

Different display controllers can have different CPU interfaces. Basically there are two different interfaces:

- Direct interface
- Indirect interface

Whereas the direct interface accesses the video memory directly by the address bus of the CPU, the indirect interface requires a more complex communication with the display controller to get access to the video memory. This can be done by different kinds of connections:

- Parallel access
- 4 pin SPI interface
- 3 pin SPI interface
- I2C bus interface

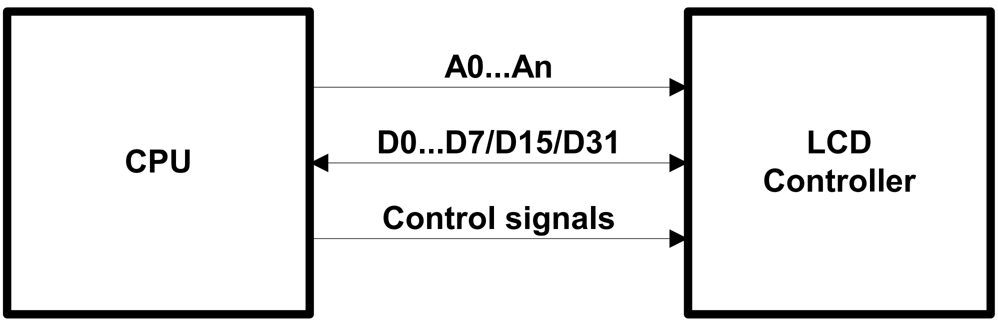
The following explains these interfaces and how to configure them. Note that not all configuration macros are always required. For details about which macros are required, refer to “Detailed display driver descriptions” on page 1075.

33.2.1 Direct interface

Some display controllers (especially those for displays with higher resolution) require a full address bus, which means they are connected to at least 14 address bits. In a direct interface configuration, video memory is directly accessible by the CPU; the address bus is connected to the display controller.
The only knowledge required when configuring a direct interface is information about the address range (which will generate a CHIP-SELECT signal for the LCD controller) and whether 8-, 16- or 32-bit accesses should be used (bus-width to the display controller). In other words, you need to know the following:

- Base address for video memory access
- Base address for register access
- Distance between adjacent video memory locations (usually 1/2/4-byte)
- Distance between adjacent register locations (usually 1/2/4-byte)
- Type of access (8/16/32-bit) for video memory
- Type of access (8/16/32-bit) for registers

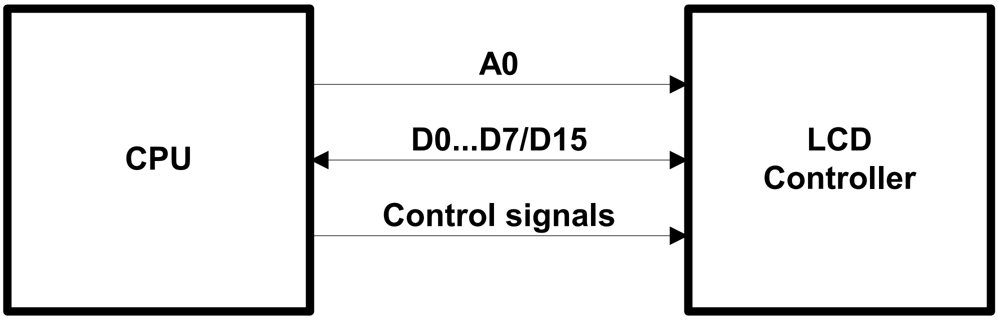
Typical block diagram



33.2.2 Indirect interface - Parallel bus

Most controllers for smaller displays use an indirect interface to connect to the CPU. With an indirect interface, only one address bit (usually A0) is connected to the LCD controller. Some of these controllers are very slow, so that the hardware designer may decide to connect it to input/output (I/O) pins instead of the address bus.

Typical block diagram



8 (16) data bits, one address bit and 2 or 3 control lines are used to connect the CPU and one LCD controller. Four macros inform the LCD driver how to access each controller used. If the LCD controller(s) is connected directly to the address bus of the CPU, configuration is simple and usually consists of no more than one line per macro. If the LCD controller(s) is connected to I/O pins, the bus interface must be simulated, which takes about 5-10 lines of program per macro (or a function call to a routine which simulates the bus interface). The signal **A0** is also called **C/D** (Command/ Data), **D/I** (Data/Instruction) or **RS** (Register select), depending on the display controller.

33.2.2.1 Example routines for connection to I/O pins

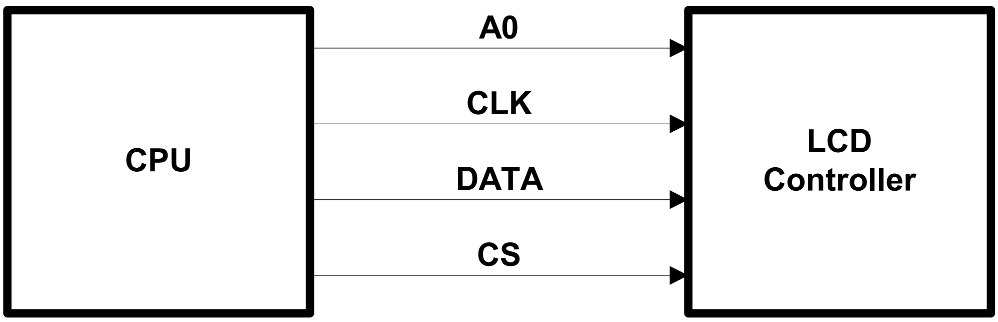
Examples can be found in the folder `Sample\LCD_X_Port`:

- `LCD_X_6800.c`, port routines for the 6800 parallel interface.
- `LCD_X_8080.c`, port routines for the 8080 parallel interface.

33.2.3 Indirect interface - 4 pin SPI

Using a 4 pin SPI interface is very similar to a parallel interface. To connect a LCD display using 4 pin SPI interface the lines A0, CLK, DATA, and CS must be connected to the CPU.

Typical block diagram



33.2.3.1 Example routines for connection to I/O pins

An example can be found in the folder `Sample\LCD_X_Port`:

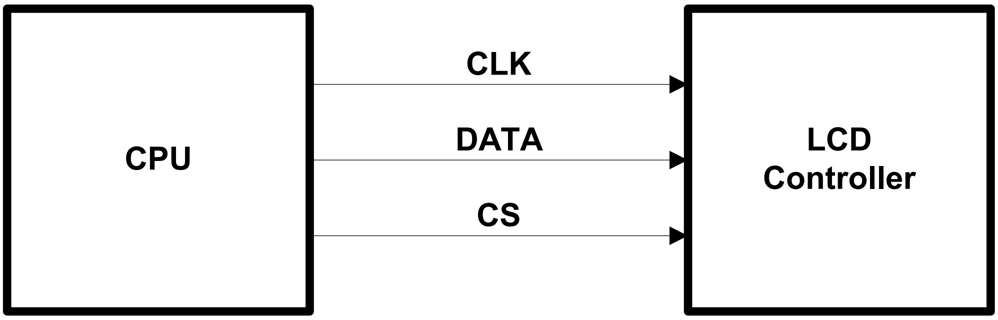
- `LCD_X_SERIAL.c`, port routines for a serial interface

This sample uses port pins for the communication. This works very slow but can be used with each CPU. This should be optimized by the customer by using the hardware support of the CPU for this kind of communication.

33.2.4 Indirect interface - 3 pin SPI

To connect a LCD display using 4 pin SPI interface the lines CLK, DATA, and CS must be connected to the CPU.

Typical block diagram



33.2.4.1 Example routines for connection to I/O pins

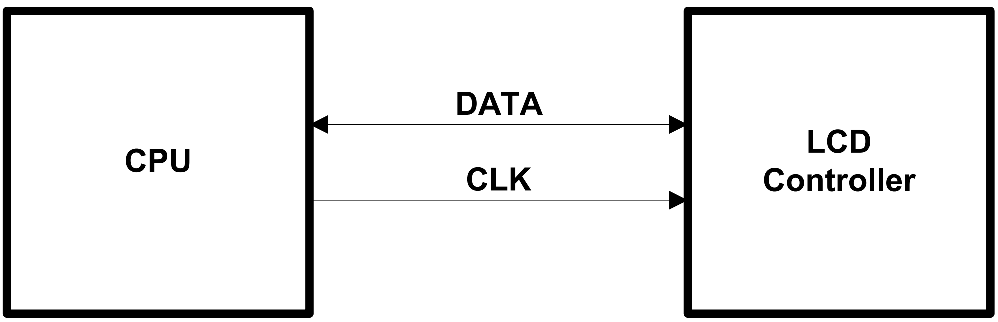
This interface does not have a separate line for distinguish between data and commands to be transmitted to the display controller. There is no standardized method to manage this. Some controllers use an additional bit for distinguish between data and command, other controllers work different.
Examples can be found in the folder `Sample\LCD_X_Port`:

- `LCD_X_Serial_3Pin.c`, port routines for a 3 pin serial interface
- `LCD_X_Serial_3Wire.c`, port routines for a 3 pin serial interface

33.2.5 Indirect interface - I2C bus

This kind of interface use only 2 lines and a standardized protocol for the communication with the display controller.

Typical block diagram



33.2.5.1 Example routines for connection to I/O pins

An example can be found in the folder `Sample\LCD_X_Port`:

- `LCD_X_I2CBUS.c`, port routines for a I2C bus interface

Similar to the serial communication examples this example uses port lines for the communication which works not very fast. If the CPU support this kind of communication these routines should be optimized by using the hardware functions.

33.3 Hardware interface configuration

The following explains how to configure the hardware communication between display driver and display controller.

33.3.1 Direct interface

Drivers which make use of a direct interface are usually configured by specifying the address of the video memory. In order to do so the function LCD_SetVRAMAddrEx() can be called. Details can be found in the section “Display driver API” on page 1158.

33.3.2 Indirect interface

There are 2 kinds of display drivers:

- Run-time configurable drivers
- Compile-time configurable drivers

Configuring these kinds of drivers works differently:

- Run-time configuration means the driver can be compiled without being configured. The configuration is done at run-time. This type of driver can still be configured at run-time when placed in a library.
- A compile-time configurable driver requires the configuration in a configuration header file, which is included at compile-time of the driver.

33.3.2.1 Run-time configuration

Run-time configurable drivers do not need to be configured at compile time. So this drivers can be used in a precompiled library.
Each driver has its own function(s) for setting up the hardware interface. This is done by passing a pointer to a `GUI_PORT_API` structure containing function pointers to the hardware routines to be used:

Elements of structure GUI_PORT_API

8 bit interface

Element	Data type	Description
pfWrite8_A0	void (*)(U8 Data)	Pointer to a function which writes one byte to the controller with C/D line low.
pfWrite8_A1	void (*)(U8 Data)	Pointer to a function which writes one byte to the controller with C/D line high.
pfWriteM8_A0	void (*)(U8 * pData, int NumItems)	Pointer to a function which writes multiple bytes to the controller with C/D line low.
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)	Pointer to a function which writes multiple bytes to the controller with C/D line high.
pfRead8_A0	U8 (*)(void)	Pointer to a function which reads one byte from the controller with C/D line low.
pfRead8_A1	U8 (*)(void)	Pointer to a function which reads one byte from the controller with C/D line high.
pfReadM8_A0	void (*)(U8 * pData, int NumItems)	Pointer to a function which reads multiple bytes from the controller with C/D line low.
pfReadM8_A1	void (*)(U8 * pData, int NumItems)	Pointer to a function which reads multiple bytes from the controller with C/D line high.

Table 33.4: GUI_PORT_API - 8 bit interface element list

16 bit interface

Element	Data type	Description
pfWrite16_A0	void (*)(U16 Data)	Pointer to a function which writes one 16 bit value to the controller with C/D line low.
pfWrite16_A1	void (*)(U16 Data)	Pointer to a function which writes one 16 bit value to the controller with C/D line high.
pfWriteM16_A0	void (*)(U16 * pData, int NumItems)	Pointer to a function which writes multiple 16 bit values to the controller with C/D line low.
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)	Pointer to a function which writes multiple 16 bit values to the controller with C/D line high.
pfRead16_A0	U16 (*)(void)	Pointer to a function which reads one 16 bit value from the controller with C/D line low.
pfRead16_A1	U16 (*)(void)	Pointer to a function which reads one 16 bit value from the controller with C/D line high.
pfReadM16_A0	void (*)(U16 * pData, int NumItems)	Pointer to a function which reads multiple 16 bit values from the controller with C/D line low.
pfReadM16_A1	void (*)(U16 * pData, int NumItems)	Pointer to a function which reads multiple 16 bit values from the controller with C/D line high.
pfReadM32_A1	void (*)(U32 * pData, int NumItems)	Pointer to a function which reads multiple 32 bit values from the controller with C/D line high.

Table 33.5: GUI_PORT_API - 16 bit interface element list

32 bit interface

Element	Data type	Description
pfWrite32_A0	void (*)(U32 Data)	Pointer to a function which writes one 32 bit value to the controller with C/D line low.
pfWrite32_A1	void (*)(U32 Data)	Pointer to a function which writes one 32 bit value to the controller with C/D line high.
pfWriteM32_A0	void (*)(U32 * pData, int NumItems)	Pointer to a function which writes multiple 32 bit values to the controller with C/D line low.
pfWriteM32_A1	void (*)(U32 * pData, int NumItems)	Pointer to a function which writes multiple 32 bit values to the controller with C/D line high.
pfRead32_A0	U32 (*)(void)	Pointer to a function which reads one 32 bit value from the controller with C/D line low.
pfRead32_A1	U32 (*)(void)	Pointer to a function which reads one 32 bit value from the controller with C/D line high.
pfReadM32_A0	void (*)(U32 * pData, int NumItems)	Pointer to a function which reads multiple 32 bit values from the controller with C/D line low.
pfReadM32_A1	void (*)(U32 * pData, int NumItems)	Pointer to a function which reads multiple 32 bit values from the controller with C/D line high.

Table 33.6: GUI_PORT_API - 32 bit interface element list

SPI interface

Element	Data type	Description
pfSetCS	void (*)(U8 NotActive)	Pointer to a function which is able to toggle the CS signal of the controller.

Table 33.7: GUI_PORT_API - SPI interface element list

This structure contains function pointers for 8-, 16- and 32 bit access. Not all function pointers are used by each driver. The required functions are listed in the description of the according display driver.

Example

The following shows a configuration example for the driver GUIDRV_SLin. It creates and configures the driver, initializes the required function pointers of the GUI_PORT_API structure and passes them to the driver:

```
GUI_DEVICE * pDevice;
CONFIG_SLIN Config = {0};
GUI_PORT_API PortAPI = {0};

//
// Set display driver and color conversion
//
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SLIN_2, GUICC_2, 0, 0);
//
// Common display driver configuration
//
LCD_SetSizeEx (0, XSIZE, YSIZE);
LCD_SetVSizeEx(0, XSIZE, YSIZE);
//
// Driver specific configuration
//
Config.UseCache  = 1;
GUIDRV_SLin_Config(pDevice, &Config);
//
// Select display controller
//
GUIDRV_SLin_SetS1D13700(pDevice);
//
// Setup hardware access routines
//
PortAPI.pfWrite16_A0  = _Write0;
PortAPI.pfWrite16_A1  = _Write1;
PortAPI.pfWriteM16_A0 = _WriteM0;
PortAPI.pfRead16_A1   = _Read1;
GUIDRV_SLin_SetBus8(pDevice, &PortAPI);
```

Details can be found in the descriptions of the run-time configurable drivers.

33.3.2.2 Compile-time configuration

A compile-time configurable driver requires its configuration in a header file. This configuration file is included when compiling the display driver. The compile-time configurable drivers use distinct macros for accessing the hardware. It depends on the interface details which macros are used. The following shows which macros are used by which kind of interface.

Macros used by an indirect interface

The following table shows the used hardware access macros:

Type	Macro	Description
F	LCD_READ_A0	Reads a byte from LCD controller with A0 - line low.
F	LCD_READ_A1	Reads a byte from LCD controller with A0 - line high.
F	LCD_WRITE_A0	Writes a byte to the display controller with A0 - line low.
F	LCD_WRITE_A1	Writes a byte to the display controller with A0 - line high.
F	LCD_WRITEM_A1	Writes several bytes to the LCD controller with A0 - line high.

Table 33.8: Hardware access macros - Indirect interface

Macros used by a 4 pin SPI interface

The following table shows the used hardware access macros:

Type	Macro	Description
F	LCD_WRITE_A0	Writes a byte to the display controller with A0 (C/D) - line low.
F	LCD_WRITE_A1	Writes a byte to the display controller with A0 (C/D) - line high.
F	LCD_WRITEM_A1	Writes several bytes to the LCD controller with A0 (C/D) - line high.

Table 33.9: Hardware access macros - 4 pin SPI interface

Macros used by a 3 pin SPI interface

The following table shows the used hardware access macros:

Type	Macro	Description
F	`LCD_WRITE`	Writes a byte to the display controller.
F	`LCD_WRITEM`	Writes several bytes to the LCD controller.

Table 33.10: Hardware access macros - 3 pin SPI interface

Macros used by a I2C bus interface

The following table shows the used hardware access macros:

Type	Macro	Description
F	`LCD_READ_A0`	Reads a status byte from LCD controller.
F	`LCD_READ_A1`	Reads a data byte from LCD controller.
F	`LCD_WRITE_A0`	Writes a instruction byte to the display controller.
F	`LCD_WRITE_A1`	Writes a data byte to the display controller.
F	`LCD_WRITEM_A1`	Writes several data bytes to the LCD controller.

Table 33.11: Hardware access macros - I2C bus interface

LCD_READ_A0

Description

Reads a byte from LCD controller with A0 (C/D) - line low.

Type

Function replacement

Prototype

```
#define LCD_READ_A0(Result)
```

Parameter	Description
`Result`	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

Table 33.12: LCD_READ_A0

LCD_READ_A1

Description

Reads a byte from LCD controller with A0 (C/D) - line high.

Type

Function replacement

Prototype

```
#define LCD_READ_A1(Result)
```

Parameter	Description
`Result`	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

Table 33.13: LCD_READ_A1

LCD_WRITE_A0

Description

Writes a byte to the display controller with A0 (C/D) - line low.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A0(Byte)
```

Parameter	Description
Byte	Byte to write.

Table 33.14: LCD_WRITE_A0

LCD_WRITE_A1

Description

Writes a byte to the display controller with A0 (C/D) - line high.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A1(Byte)
```

Parameter	Description
Byte	Byte to write.

Table 33.15: LCD_WRITE_A1

LCD_WRITEM_A1

Description

Writes several bytes to the LCD controller with A0 (C/D) - line high.

Type

Function replacement

Prototype

```
#define LCD_WRITEM_A1(paBytes, NumberOfBytes)
```

Parameter	Description
paBytes	Placeholder for the pointer to the first data byte.
NumberOfBytes	Number of data bytes to be written.

Table 33.16: LCD_WRITEM_A1

LCD_WRITE

Description

Writes a byte to the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITE(Byte)
```

Parameter	Description
Byte	Byte to write.

Table 33.17: LCD_WRITE

LCD_WRITEM

Description

Writes several bytes to the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITEM(paBytes, NumberOfBytes)
```

Parameter	Description
paBytes	Placeholder for the pointer to the first data byte.
NumberOfBytes	Number of data bytes to be written.

Table 33.18: LCD_WRITEM

33.4 Non readable displays

Some display controllers with an indirect interface do not support reading back display data. Especially displays which are connected via SPI interface often have this limitation. In this case we recommend using a display data cache. For details how to enable a display data cache, refer to “Detailed display driver descriptions” on page 1075.
On systems with a very small RAM it is sometimes not possible to use a display data cache. If a display is not readable and a display data cache can not be used some features of emWin will not work. The list below shows these features:

- Cursors and Sprites
- XOR-operations, required for text cursors in EDIT and MULTIEDIT widgets
- Alpha blending
- Antialiasing

This is valid for all drivers where one data unit (8 or 16 bit) represents one pixel. Display drivers, where one data unit represents more than one pixel, can not be used if no display data cache is available and the display is not readable. An example is the GUIDRV_Page1bpp driver where one byte represents 8 pixels.

33.5 Display orientation

If the original display orientation does not match the requirements, there are different ways to change the display orientation:
- Driver based configuration of the desired orientation
- Using GUI_SetOrientation()

33.5.1 Driver based configuration of display orientation

If the display driver supports different orientations it is recommended to use the driver for setting up the right orientation. The way how to configure the display orientation then depends on the display driver to be used. Whereas the display orientation of the most common drivers is run-time configurable some drivers need to be configured at compile time.

33.5.1.1 Run-time configuration

The display orientation of the most common driver is determined by creating the display driver device in `LCD_X_Config()` using the proper macro. The according macros are listed within the description of “GUIDRV_Lin” on page 1098.

33.5.1.2 Compile-time configuration

The display orientation of some drivers with indirect interface like GUIDRV_CompactColor_16 needs to be configured at compile time in the configuration file of the driver.

Display orientations

There are 8 possible display orientations; the display can be turned 0°, 90°, 180° or 270° and can also be viewed from top or from bottom. The default orientation is 0° and top view. These 4*2 = 8 different display orientations can also be expressed as a combination of 3 binary switches: X-mirroring, Y-mirroring and X/Y swapping. For this purpose, the binary configuration macros listed below can be used with each driver in any combination. If your display is not oriented well, take a look at the config switches in the table below to make it work properly. The orientation is handled as follows: Mirroring in X and Y first, then swapping (if selected).

Display	Orientation macros in driver configuration file	Display	Orientation macros in driver configuration file
	No orientation macro required		#define LCD_MIRROR_Y 1
	#define LCD_MIRROR_X 1		#define LCD_MIRROR_X 1 #define LCD_MIRROR_Y 1
	#define LCD_SWAP_XY 1		#define LCD_SWAP_XY 1 #define LCD_MIRROR_Y 1
	#define LCD_SWAP_XY 1 #define LCD_MIRROR_X 1		#define LCD_SWAP_XY 1 #define LCD_MIRROR_X 1 #define LCD_MIRROR_Y 1

Table 33.19: Display orientations

Details on how to use multiple orientations simultaneously can be found in the section “Run-time screen rotation” on page 961.

33.5.2 Function based configuration of display orientation

Another possibility to set up the display orientation is to call GUI_SetOrientation(). Using this function is recommended if the display driver can not be used.

GUI_SetOrientation()

Description

This function changes the display orientation by using a rotation device.

Prototype

```
int GUI_SetOrientation(int Orientation);
```

Parameter	Description
Orientation	See the table below for an overview of valid values.

Table 33.20: GUI_SetOrientation() parameter list

Resulting display	Value to use for GUI_SetOrientation()	Resulting display	Value to use for GUI_SetOrientation()
	0		GUI_MIRROR_Y
	GUI_MIRROR_X		GUI_MIRROR_X | GUI_MIRROR_Y
	GUI_SWAP_XY		GUI_SWAP_XY | GUI_MIRROR_Y
	GUI_SWAP_XY | GUI_MIRROR_X		GUI_SWAP_XY | GUI_MIRROR_X | GUI_MIRROR_Y

Return value

0 on success, 1 on error.

Additional information

The rotation device covers the complete virtual screen within an internal screen buffer. Because of this the use of this function requires additional memory for this additional screen buffer. The number of required bytes can be calculated as follows:

```
Virtual xSize * Virtual ySize * BytesPerPixel
```

The number of bytes per pixel is for configurations from 1-8bpp 1, for systems with more than 8bpp up to 16bpp 2 and for systems with more than 16bpp 4.
Each drawing operation first updates this buffer. After this the affected pixels are passed to the display driver device.

GUI_SetOrientationEx()

Description

This function changes the orientation in the specified layer by using a rotation device.

Prototype

```
int GUI_SetOrientation(int Orientation, int LayerIndex);
```

Parameter	Description
Orientation	Refer to "GUI_SetOrientation()" on page 1071 for an overview of valid values.
LayerIndex	Index of the layer which Orientation has to be (re-)configured.

Table 33.21: GUI_SetOrientationEx() parameter list

Return value

0 on success, 1 on error.

Additional information

See "GUI_SetOrientation()" on page 1071.

33.6 Display driver callback function

A display driver requires a callback function. It is called by the driver for several tasks. One task is putting the display driver into operation which is also explained in the chapter 'Configuration'. It is also called for other tasks which require hardware related operations like switching the display on and off or setting a lookup table entry.

LCD_X_DisplayDriver()

Description

This is the callback function of the display driver. It is called by the display driver for several jobs. It passes a command and a pointer to a data structure to the callback routine. The command tells the callback function what should be done. If the command requires parameters they are passed through the data pointer `pData`. It points to a structure whose format depends on the command.

Prototype

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData);
```

Parameter	Description
`LayerIndex`	Zero based layer index.
`Cmd`	Command to be executed. Detailed descriptions below.
`pData`	Pointer to a data structure.

Table 33.22: LCD_X_DisplayDriver() parameter list

Return value

The routine should return -2 if an error occurs, -1 if the command is not handled by the function and 0 if the command has been successfully executed.

33.6.1 Commands passed to the callback function

The following explains the common commands passed to the callback function. For details about display driver specific commands, refer to "Detailed display driver descriptions" on page 1075. They are described under the topic 'Additional callback commands'.

LCD_X_INITCONTROLLER

As mentioned above the application should initialize the display controller and put it into operation if the callback routine receives this command. No parameters are passed on this command. Typically an initialization routine which initializes the registers of the display controller should be called in reaction of this command.

Parameters

None.

LCD_X_SETVRAMADDR

This command is passed by the driver to tell the callback routine the start address of the video RAM. The typical reaction should be writing the address to the frame buffer start address register.

Parameters

`pData` points to a data structure of type `LCD_X_SETVRAMADDR_INFO`:

Elements of structure LCD_X_SETVRAMADDR_INFO

Data type	Element	Description
void *	pVRAM	Points to the start address of the video RAM. This address is typically written to the video RAM base address register of the display controller.

Table 33.23: LCD_X_SETVRAMADDR_INFO element list

LCD_X_ON

This command switches the display on.

Parameters

none

LCD_X_OFF

This command switches the display off.

Parameters

none

LCD_X_SETLUTENTRY

A lookup table entry should be set. The typical reaction should be writing an entry into the lookup table of the display controller.

Parameters

pData points to a data structure of type LCD_X_SETLUTENTRY_INFO:

Elements of structure LCD_X_SETLUTENTRY_INFO

Data type	Element	Description
LCD_COLOR	Color	RGB value of the color to be written to the LUT. Note that the format required by the hardware could be different to the RGB format.
U8	Pos	Zero based index of the LUT entry to be set.

Table 33.24: LCD_X_SETLUTENTRY_INFO element list

LCD_X_SETORG

The function is used in relation with virtual screens. It is called if the origin of the display should be set. A typical reaction can be modifying the frame buffer start address.

Parameters

pData points to a data structure of type LCD_X_SETORG_INFO:

Elements of structure LCD_X_SETORG_INFO

Data type	Element	Description
int	xPos	New X-position of the physical display position within the virtual screen.
int	yPos	New Y-position of the physical display position within the virtual screen.

Table 33.25: LCD_X_SETORG_INFO element list

33.7 Detailed display driver descriptions

33.7.1 GUIDRV_BitPlains

This driver has been developed for systems without display controller. It manages each color bit in a separate plain. This means if the color depth is for example 4 bits per pixel the driver manages 4 bit plains each containing one bit.
Initially the driver has been made to drive monochrome and color TFTs with an R323C/111 CPU via SPI interface. But the driver can be used also for similar applications.
The driver does only manage the content of the bit plains. It does not contain any display controller specific code.

Supported hardware

Controllers

None.

Bits per pixel

The driver has been developed for a color depth of 1 to 8 bits per pixel.

Interface

It is required to write an application defined routine which uses the content of the bit plains to generate the color signals for the display. The driver comes with a sample for the R32C/111 CPU which refreshes the display via timer interrupt routine using the SPI interface.

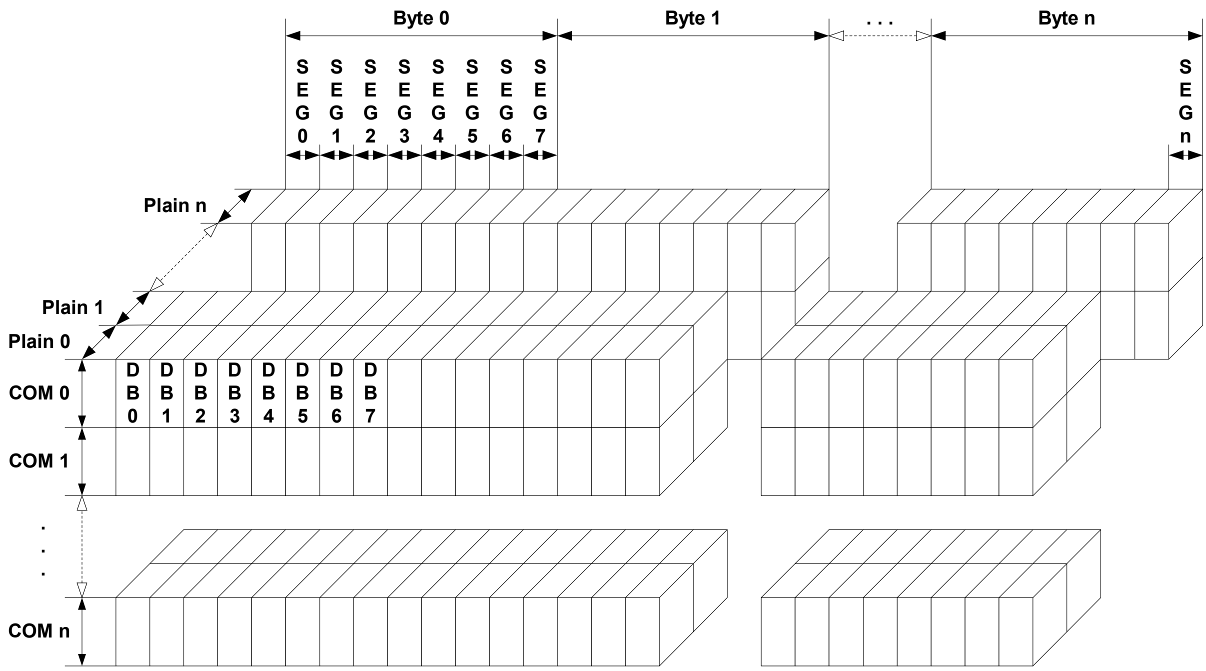
Driver selection

To use GUIDRV_BitPlains for the given display, the following command can be used e.g.:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_BITPLAINS, GUICC_M111, 0, 0);
```

More information about using the proper palette mode can be found in the chapter “Colors” on page 291.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display. The display memory is divided into separate plains for each bit of the colors. This means that bit 0 of each pixel is stored in plain 0, the bit 1 in plain 1 and so on. The advantage of this method is that each color bit of the display data can be accessed very quickly.

RAM requirements

The required size of the display memory area can be calculated as follows:

```
Size = BitsPerPixel * (LCD_XSIZE + 7) / 8 * LCD_YSIZE
```

Please note that the pointers to the bit plain areas need to be passed to the configuration routine of the driver. They are not allocated within the driver but from application side.

Hardware configuration

Normally, the hardware interface is an interrupt service routine (ISR) which updates the display. The driver comes with an example written in "C" code. This routine should serve as an example.

Additional run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_BitPlains_Config()	Passes a pointer to a CONFIG_BITPLAINS structure to the driver.
LCD_SetVRAMAddrEx()	Passes a pointer to a CONFIG_VRAM_BITPLAINS structure to the driver. See the description below. A description of the function can be found on page 1168.

Table 33.26: Run-time configuration API list

Elements of structure CONFIG_VRAM_BITPLAINS

Data type	Element	Description
U8 *	apVRAM	Array of pointers to the memory locations to be used by the driver for each bit plain. If the driver for example works in 2bpp mode only the first 2 pointers are used (One plain for each bit of the color information).

Table 33.27: CONFIG_VRAM_BITPLAINS element list

GUIDRV_BitPlains_Config()

Description

This function passes a pointer to a `CONFIG_BITPLAINS` structure to the driver.

Prototype

```
void GUIDRV_BitPlains_Config(GUI_DEVICE        * pDevice,
                                 CONFIG_BITPLAINS * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a `CONFIG_BITPLAINS` structure explained below.

Table 33.28: GUIDRV_BitPlains_Config() parameter list

Elements of structure CONFIG_BITPLAINS

Data type	Element	Description
int	Mirror	Config switch to mirror the bits of the display data.

Table 33.29: CONFIG_BITPLAINS element list

Configuration example

```
//
// Data arrays to be used by the display driver
//
static U8 _aPlain_0[BYTES_PER_LINE * YSIZE_PHYS];
static U8 _aPlain_1[BYTES_PER_LINE * YSIZE_PHYS];
static U8 _aPlain_2[BYTES_PER_LINE * YSIZE_PHYS];

//
// Structure to be passed to the driver
//
static struct {
  U8 * apVRAM[8];
} _VRAM_Desc = {
  _aPlain_0,
  _aPlain_1,
  _aPlain_2,
};

void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_BITPLAINS, COLOR_CONVERSION, 0, 0);
  //
  // Display driver configuration
  //
  if (LCD_GetSwapXY()) {
    LCD_SetSizeEx (0, YSIZE_PHYS, XSIZE_PHYS);
    LCD_SetVSizeEx(0, YSIZE_PHYS, XSIZE_PHYS);
  } else {
    LCD_SetSizeEx (0, XSIZE_PHYS, YSIZE_PHYS);
    LCD_SetVSizeEx(0, XSIZE_PHYS, YSIZE_PHYS);
  }
  //
  // Initialize VRAM access of the driver
  //
  LCD_SetVRAMAddrEx(0, (void *)&_VRAM_Desc);
}
```


33.7.2 GUIDRV_DCache

GUIDRV_DCache has been developed to minimize the communication between emWin and the display controller. It uses 2 caches to be able to check exactly which pixels have been changed between locking and unlocking the cache. When locking the cache the driver makes a copy of the current cache. When unlocking it, it checks exactly which pixels have been changed. Only the changed pixels will be send to the controller.
Using this double cache driver makes sense if the performance bottleneck is the communication between CPU and display controller.
The driver can not be used stand alone. It is required to use a ’real’ display driver for the drawing operations.
GUIDRV_DCache is part of the emWin basic package.

Supported hardware

The double cache driver is able to work with each runtime configurable display driver which works with 16bpp color format.

Driver selection

To be able to use this driver the following call has to be made:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_DCACHE, GUICC_1, 0, Layer);
```

RAM requirements

As the drivers name implies it uses 2 caches. Currently only a color depth of 1bpp is supported by the driver. The RAM usage can be calculated as follows:

```
Size = 2 * (LCD_XSIZE + 7) / 8 * LCD_YSIZE
```

Run-time configuration

First the ’real’ driver should be created and configured:

```
pDriver = GUI_DEVICE_Create(DISPLAY_DRIVER, GUICC_XXX, 0, Layer);
//
// Configuration of ’real’ driver
//
.
.
.
```

`GUICC_XXX` means any 16bpp color conversion scheme. After that the double cache driver can be created and configured:

```
//
// Create and configure (double) cache driver, ...
//
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_DCACHE, GUICC_1, 0, Layer);
//
// ... set size, ...
//
LCD_SetSizeEx (0, XSIZE_PHYS,  YSIZE_PHYS);
LCD_SetVSizeEx(0, VXSIZE_PHYS, VYSIZE_PHYS);
//
// ...set color depth, ...
//
GUIDRV_DCache_SetMode1bpp(pDevice);
```

Then the ’real’ driver should be added for doing the drawing operations:

```
//
// ... and add real driver.
//
GUIDRV_DCache_AddDriver(pDevice, pDriver);
```

Configuration API

Routine	Description
GUIDRV_DCache_AddDriver()	Adds the 'real' driver for the drawing operations.
GUIDRV_DCache_SetMode1bpp()	Sets the color depth to be used for the cache.

Table 33.30: Configuration API list

GUIDRV_DCache_AddDriver()

Description

Adds the 'real' driver to the DCache driver which is used for the drawing operations.

Prototype

```
void GUIDRV_DCache_AddDriver(GUI_DEVICE * pDevice, GUI_DEVICE * pDriver);
```

Parameter	Description
pDevice	Pointer to the DCache driver device.
pDriver	Pointer to the real driver device.

Table 33.31: GUIDRV_DCache_AddDriver() parameter list

Additional information

The used driver should work in 16bpp mode because the double cache driver currently only supports 16bpp output.

GUIDRV_DCache_SetMode1bpp()

Description

Sets the 1bpp mode for the DCache driver.

Prototype

```
void GUIDRV_DCache_SetMode1bpp(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the DCache driver device.

Table 33.32: GUIDRV_DCache_SetMode1bpp() parameter list

Additional information

Currently the DCache driver works only with a color depth of 1bpp.

33.7.3 GUIDRV_Dist

GUIDRV_Dist has been developed to support displays with multiple controllers. It is able to support multiple display areas each driven by a separate display controller. The distribution driver passes the drawing operations to the according display driver. This also works with overlapping operations. In these cases the operations are divided into sub operations for each affected controller. GUIDRV_Dist is part of the emWin basic package.

Supported hardware

The distribution driver is able to work with each runtime configurable display driver. Please note that it is required that each of the configured display drivers use the same color conversion as the distribution driver.

Driver selection

To be able to use this driver the following call has to be made:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_DIST, COLOR_CONVERSION, 0, Layer);
```

RAM requirements

None.

Run-time configuration

After the driver has been created the actual display drivers should be also created and added to the distribution device:

```
pDevice0 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
pDevice1 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
GUIDRV_Dist_AddDriver(pDevice, pDevice0, &Rect0);
GUIDRV_Dist_AddDriver(pDevice, pDevice1, &Rect1);
```

GUIDRV_Dist_AddDriver()

Description

Adds a display driver to the distribution driver.

Prototype

```
void GUIDRV_Dist_AddDriver(GUI_DEVICE * pDevice,
                           GUI_DEVICE * pDriver, GUI_RECT * pRect);
```

Parameter	Description
pDevice	Pointer to the already created distribution device.
pDriver	Pointer to the already created driver device to be added.
pRect	Pointer to the rectangle in which outputs have to affect the driver.

Table 33.33: GUIDRV_Dist_AddDriver() parameter list

Configuration example

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_DIST, COLOR_CONVERSION, 0, 0);
  //
  // Display size configuration
  //
  LCD_SetSizeEx (0, XSIZE_PHYS,  YSIZE_PHYS);
  LCD_SetVSizeEx(0, VXSIZE_PHYS, VYSIZE_PHYS);
  //
  // Create first display driver
  //
  pDevice0 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
```



```
  //
  // Configuration of first driver
  //
  ...
  //
  // Create second display driver
  //
  pDevice1 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
  //
  // Configuration of second driver
  //
  ...
  //
  // Add display drivers to distribution driver
  //
  Rect0.x0 = 0;
  Rect0.y0 = 160;
  Rect0.x1 = 223;
  Rect0.y1 = 319;
  GUIDRV_Dist_AddDriver(pDevice, pDevice0, &Rect0);
  Rect1.x0 = 0;
  Rect1.y0 = 0;
  Rect1.x1 = 223;
  Rect1.y1 = 159;
  GUIDRV_Dist_AddDriver(pDevice, pDevice1, &Rect1);
}
```


33.7.4 GUIDRV_FlexColor

Supported hardware

Controllers

The supported display controllers are listed in the description of the function “GUIDRV_FlexColor_SetFunc()” on page 1085.

Bits per pixel

Supported color depth is 16 bpp and 18 bpp.

Interfaces

The driver supports 8-bit, 9-bit, 16-bit and 18-bit indirect interface.

Driver selection

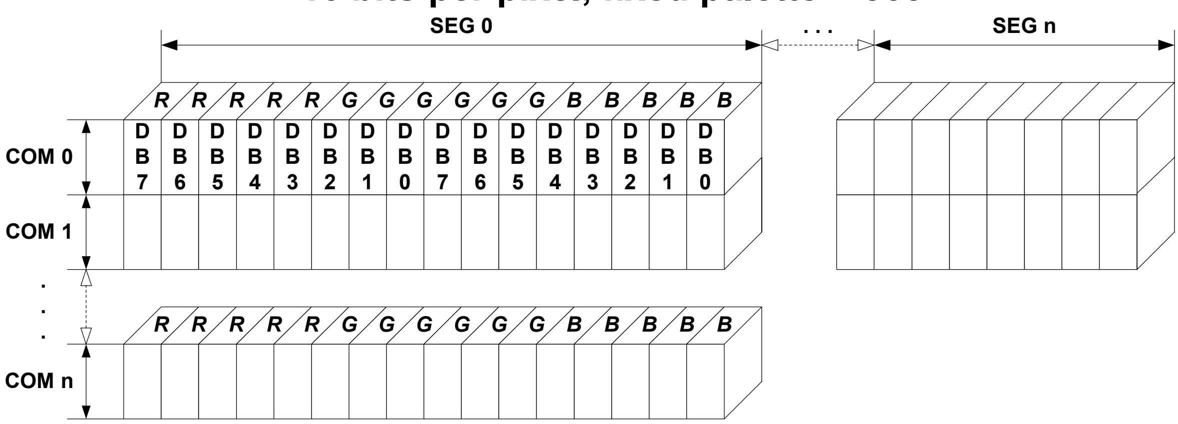
To be able to use this driver the following call has to be made:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_FLEXCOLOR,
                                   COLOR_CONVERSION, 0, Layer);
```

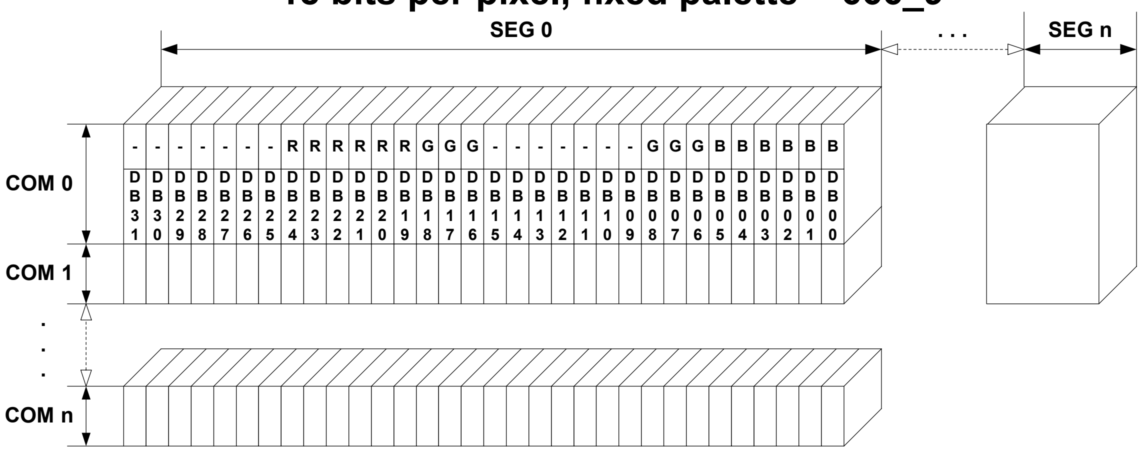
Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization

16 bits per pixel, fixed palette = 565



18 bits per pixel, fixed palette = 666_9



RAM requirements

This display driver requires app. 500 Bytes to work. It can also be used with and without a display data cache, containing a complete copy of the content of the display data RAM. The amount of memory used by the cache is:

```
LCD_XSIZE * LCD_YSIZE * BytesPerPixel
```

BytesPerPixel is 2 for 16bpp mode and 4 for 18bpp mode. Using a cache avoids reading operations from the display controller in case of XOR drawing operations and further it speeds up string output operations.

Run-time configuration API

The following table lists the available run-time configuration routines:

Routine	Description
LCD_SetDevFunc()	Can be used to set optional or custom defined routines.

Table 33.34: Run-time configuration API list

Commands supported by LCD_SetDevFunc()

Command	Description
LCD_DEVFUNC_READMPIXELS	Can be used to set a custom defined routine for reading multiple pixels from the display controller.
LCD_DEVFUNC_READPIXEL	Can be used to set a custom defined routine for reading a single pixel from the display controller.

Table 33.35: Commands supported by LCD_SetDevFunc()

Further information about the LCD layer routines can be found under “LCD layer routines” on page 1159.

Important note on reading back pixel data from the controller

Because of the plurality of the supported display controllers and their operation modes the driver has not been tested with each interface of each supported controller. The behavior of the controller when reading back pixel data often depends on custom configuration and hardware details. Because of that it could happen, that the driver has no appropriate reading function(s) available. In that case the above explained function `LCD_SetDevFunc()` can be used to set application defined functions for reading back pixel data.
The main problem for the driver here is not getting data from the driver. Getting the color bits in the right order is the problem here. The custom defined functions need to supply ’pixel index’ values. This index format needs to comply to the index format determined by the color conversion routines configured for the driver device.
Because the data supplied by the hardware interface functions of the driver in most cases does not have the right index format, the reading routines need to convert that raw data into the required pixel index format determined by the driver device configuration.
For details about the hardware interface functions please also refer to `GUIDRV_FlexColor_SetFunc()` and its parameter `pHW_API`.
For details about the pixel index format please also refer to `GUI_DEVICE_CreateAndLink()` and its parameter `pColorConvAPI` and the chapter ’Colors’.
A sample configuration which shows how custom reading functions can be achieved is available in the configuration file sample folder shipped with the driver.

Configuration API

Routine	Description
Common configuration routines	
GUIDRV_FlexColor_SetFunc()	Configures bus, cache, and hardware routines.
GUIDRV_FlexColor_Config()	Configures orientation and offset of the SEG- and COM-lines.
Detailed interface selection	
GUIDRV_FlexColor_SetInterface66712_B9()	Set up bus interface (TYPE_I, TYPE_II).
GUIDRV_FlexColor_SetInterface66712_B18()	Set up bus interface (TYPE_I, TYPE_II).
GUIDRV_FlexColor_SetInterface66715_B9()	Set up bus interface (TYPE_I, TYPE_II).
GUIDRV_FlexColor_SetInterface66715_B18()	Set up bus interface (TYPE_I, TYPE_II).
Configuration of read back function	
GUIDRV_FlexColor_SetReadFunc66709_B16()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66712_B9()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66712_B16()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66715_B9()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66715_B16()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66720_B16()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66772_B8()	Read back function settings.
GUIDRV_FlexColor_SetReadFunc66772_B16()	Read back function settings.

Table 33.36: Configuration API list

The above set of configuration functions set up the detailed behavior of the driver. In short they do the following:

GUIDRV_FlexColor_SetFunc()

- Configures the LCD-controller to be used, color depth and cache settings.

GUIDRV_FlexColor_Config()

- Configures display orientation, dummy reads and first SEG- and COM-lines.

GUIDRV_FlexColor_SetInterface()

- Configures the bus interface to be used.

GUIDRV_FlexColor_SetReadFunc()

- Configures the behavior when reading back pixel data.

Calling sequence

The following shows a recommended sequence of configuration function calls:

```
GUI_DEVICE_CreateAndLink()
GUIDRV_FlexColor_Config()
LCD_SetSizeEx()
LCD_SetVSizeEx()
GUIDRV_FlexColor_SetInterface()
GUIDRV_FlexColor_SetReadFunc()
GUIDRV_FlexColor_SetFunc()
```


GUIDRV_FlexColor_SetFunc()

Description

Configures bus width, cache usage and hardware routines.

Prototype

```
void GUIDRV_FlexColor_SetFunc(GUI_DEVICE   * pDevice,
                              GUI_PORT_API * pHW_API,
                              void (* pfFunc)(GUI_DEVICE * pDevice),
                              void (* pfMode)(GUI_DEVICE * pDevice));
```

Parameter	Description
pDevice	Pointer to the driver device structure.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.
pfFunc	Controller selection macro. See table below.
pfMode	See table below.

Table 33.37: GUIDRV_FlexColor_SetFunc() parameter list

Permitted values for parameter pfFunc Supported display controller	
GUIDRV_FLEXCOLOR_F66702	Set up the driver to use one of the following controllers: - Solomon SSD1284, SSD1289, SSD1298
GUIDRV_FLEXCOLOR_F66708	Set up the driver to use one of the following controllers: - FocalTech FT1509 - Ilitek ILI9320, ILI9325, ILI9328, ILI9335 - LG Electronics LGDP4531, LGDP4551 - OriseTech SPFD5408 - Renesas R61505, R61580
GUIDRV_FLEXCOLOR_F66709	Set up the driver to use one of the following controllers: - Epson S1D19122 - Himax HX8353, HX8325A, HX8357, HX8369 - Ilitek ILI9338, ILI9340, ILI9341, ILI9342, ILI9163, ILI9481, ILI9486, ILI9488 - Novatek NT39122 - Orisetech SPFD54124C, SPFD5414D - Renesas R61516, R61526 - Sitronix ST7628, ST7637, ST7687, ST7715, ST7735, ST7789 - Solomon SSD1355
GUIDRV_FLEXCOLOR_F66712	Set up the driver to use one of the following controllers: - Himax HX8340, HX8347, HX8352
GUIDRV_FLEXCOLOR_F66714	Set up the driver to use the following controller: - Solomon SSD2119
GUIDRV_FLEXCOLOR_F66715	Set up the driver to use one of the following controllers: - Himax HX8352B
GUIDRV_FLEXCOLOR_F66718	Set up the driver to use the following controller: - Syncoam SEPS525
GUIDRV_FLEXCOLOR_F66719	Set up the driver to use the following controller: - Samsung S6E63D6
GUIDRV_FLEXCOLOR_F66720	Set up the driver to use one of the following controllers: - Solomon SSD1961, SSD1963
GUIDRV_FLEXCOLOR_F66721	Set up the driver to use one of the following controller: - RAIO RA8870, RA8875
GUIDRV_FLEXCOLOR_F66722	Set up the driver to use one of the following controller: - Solomon SSD1351
GUIDRV_FLEXCOLOR_F66772	Set up the driver to use one of the following controllers: - Himax HX8301 - Ilitek ILI9220, ILI9221 - LG Electronics LGDP4525 - Samsung S6D0117 - Sitronix ST7712

The display controllers listed in the table above are the currently known controllers compatible to the driver. Please note that the used numbers of the selection macros are compatible to some of the LCD_CONTROLLER macro of the driver GUIDRV_CompactColor_16. This makes it easy to migrate from the compile time configurable GUIDRV_CompactColor_16 to the runtime configurable GUIDRV_FlexColor.

Permitted values for parameter pfMode	
GUIDRV_FLEXCOLOR_M16C0B8	16bpp, no cache, 8 bit bus
GUIDRV_FLEXCOLOR_M16C1B8	16bpp, cache, 8 bit bus
GUIDRV_FLEXCOLOR_M16C0B16	16bpp, no cache, 16 bit bus
GUIDRV_FLEXCOLOR_M16C1B16	16bpp, cache, 16 bit bus
GUIDRV_FLEXCOLOR_M18C0B9	18bpp, no cache, 9 bit bus
GUIDRV_FLEXCOLOR_M18C1B9	18bpp, cache, 9 bit bus
GUIDRV_FLEXCOLOR_M18C0B18	18bpp, no cache, 18 bit bus
GUIDRV_FLEXCOLOR_M18C1B18	18bpp, cache, 18 bit bus

Each controller selection supports different operation modes. The table below shows the supported modes for each controller:

Selection macro	M16C0B8	M16C1B8	M16C0B16	M16C1B16	M18C0B9	M18C1B9	M18C0B18	M18C1B18
GUIDRV_FLEXCOLOR_F66702	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66708	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66709	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66712	X	X	X	X	X	X	X	X
GUIDRV_FLEXCOLOR_F66714	X	X	X	X	X	X	-	-
GUIDRV_FLEXCOLOR_F66715	X	X	X	X	X	X	X	X
GUIDRV_FLEXCOLOR_F66718	X	X	X	X	X	X	-	-
GUIDRV_FLEXCOLOR_F66719	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66720	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66721	X	X	X	X	-	-	-	-
GUIDRV_FLEXCOLOR_F66722	X	X	-	-	-	-	-	-
GUIDRV_FLEXCOLOR_F66772	X	X	X	X	-	-	-	-

Table 33.38: Supported operation modes

'-' means not supported
'X' means supported

Required GUI_PORT_API routines

The required GUI_PORT_API routines depend on the used interface. If a cache is used the routines for reading data are unnecessary for each interface:

8 bit interface

Element	Data type
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfRead8_A1	U8 (*)(void)
pfReadM8_A1	void (*)(U8 * pData, int NumItems)

Table 33.39: GUI_PORT_API - 8 bit interface

16 bit interface

Element	Data type
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)
pfRead16_A1	U16 (*)(void)
pfReadM16_A1	void (*)(U16 * pData, int NumItems)

Table 33.40: GUI_PORT_API - 16 bit interface

18 bit interface

Element	Data type
pfWrite32_A0	void (*)(U32 Data)
pfWrite32_A1	void (*)(U32 Data)
pfWriteM32_A1	void (*)(U32 * pData, int NumItems)
pfReadM8_A1	U32 (*)(void)
pfReadM32_A1	void (*)(U32 * pData, int NumItems)

Table 33.41: GUI_PORT_API - 18 bit interface

9 bit interface

The following describes the behavior of the 9 bit bus variant of the driver. When working with a 9 bit interface the display controller uses the lines D17-D10 or lines D7-D0 (8 bit) for accessing the command register and D17-D9 or D8-D0 (9 bit) for passing data. This means the lines D17-D9 or D8-D0 are connected to the interface lines of the CPU.
The driver passes 16 bit values to the hardware routines. In dependence of the selected driver interface (TYPE_I or TYPE_II) the bits 7-0 (TYPE_I) or the bits 8-1 (TYPE_II) already contain the right values to be passed to the controller. No further shift operation is required in the hardware routines.
To be able to process pixel data as fast as possible, the driver passes two 16 bit data values per pixel (0000000R RRRRRGGG and 0000000G GGBBBBBB) to the hardware routines. Only the first 9 bits contain pixel data. So nothing need to be shifted in the hardware routines.

In case of using the 9 bit interface the driver requires 16 bit hardware routines for communicating with the controller.

Element	Data type	Description
pfWrite16_A0	`void (*)(U16 Data)`	Routine used to set up the index register. Dependent on used bus interface DB8-DB1 or DB7-DB0 are used.
pfWrite16_A1	`void (*)(U16 Data)`	Routine used to pass register parameters. Dependent on used bus interface DB8-DB1 or DB7-DB0 are used.
pfWriteM16_A1	`void (*)(U16 * pData, int NumItems)`	Data to be written (DB0-DB9)
pfReadM16_A1	`void (*)(U16 * pData, int NumItems)`	Data read (DB0-DB9)

Table 33.42: GUI_PORT_API - 9 bit interface

GUIDRV_FLEXCOLOR_F66721

In addition to the hardware interface functions described above, the driver requires the following functions in case it is configured for a 66721-type controller. These controllers require reading the controller status. According to the used interface these functions are:

8 bit interface

Element	Data type
pfRead8_A0	`U8 (*)(void);`

Table 33.43: GUIDRV_FLEXCOLOR_F66721 - 8 bit interface

16 bit interface

Element	Data type
pfRead16_A0	`U16 (*)(void);`

Table 33.44: GUIDRV_FLEXCOLOR_F66721 - 16 bit interface

Further these controllers do not support setting the orientation by the driver using `GUIDRV_FlexColor_Config()`. This needs to be done in the initialization by setting the Display Configuration Register 0x20 (DPCR) accordingly.

GUIDRV_FlexColor_Config()

Description

Configures orientation and offset of the SEG- and COM-lines.

Prototype

```
void GUIDRV_FlexColor_Config(GUI_DEVICE       * pDevice,
                             CONFIG_FLEXCOLOR * pConfig);
```

Parameter	Description
pDevice	Pointer to the device to configure.
pConfig	Pointer to a CONFIG_FLEXCOLOR structure. See element list below.

Table 33.45: GUIDRV_FlexColor_Config() parameter list

Elements of structure CONFIG_FLEXCOLOR

Data type	Element	Description
int	FirstSEG	First segment line.
int	FirstCOM	First common line.
int	Orientation	One or more "OR" combined values of the table below.
U16	RegEntryMode	Normally the display controller uses 3 bits of one register to define the required display orientation. Normally these are the bits ID0, ID1 and AM. To be able to control the content of the other bits the RegEntryMode element can be used. The driver combines this value with the required orientation bits during the initialization process.
int	NumDummyReads	Defines the number of reading operations which have to be done until valid data can be retrieved. Please note that only values != 0 are accepted. If the controller does not need one or more dummy reads, -1 should be used here.

Table 33.46: CONFIG_FLEXCOLOR element list

Permitted values for parameter Orientation	
GUI_MIRROR_X	Mirroring the X-axis
GUI_MIRROR_Y	Mirroring the Y-axis
GUI_SWAP_XY	Swapping X- and Y-axis

GUIDRV_FlexColor_SetInterface66712_B9()

GUIDRV_FlexColor_SetInterface66715_B9()

Description

Sets the type of interface to be used.

Prototype

```
void GUIDRV_FlexColor_SetInterface66712_B9(GUI_DEVICE * pDevice, int Type);
void GUIDRV_FlexColor_SetInterface66715_B9(GUI_DEVICE * pDevice, int Type);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Table 33.47: GUIDRV_FlexColor_SetInterface66712_B9() / GUIDRV_FlexColor_SetInterface66715_B9() parameter list

Permitted values for parameter Type	
GUIDRV_FLEXCOLOR_IF_TYPE_I	Uses lines DB7-DB0 for register access and lines DB8-DB0 for data access. (default)
GUIDRV_FLEXCOLOR_IF_TYPE_II	Uses lines DB8 to DB1 for register access and lines DB8-DB0 for data access.

Additional information

The difference between the interfaces affects the register access to the controller. Normally there are 2 kinds of possible interfaces available when working with the 18 bit bus interface. TYPE_I uses the lines D7 to D0 for register access whereas TYPE_II uses the lines D8 to D1.

GUIDRV_FlexColor_SetInterface66712_B18()

GUIDRV_FlexColor_SetInterface66715_B18()

Description

Sets the type of interface to be used.

Prototype

```
void GUIDRV_FlexColor_SetInterface66712_B18(GUI_DEVICE * pDevice, int Type);
void GUIDRV_FlexColor_SetInterface66715_B18(GUI_DEVICE * pDevice, int Type);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Table 33.48: GUIDRV_FlexColor_SetInterface66712_B18() / GUIDRV_FlexColor_SetInterface66715_B18() parameter list

Permitted values for parameter Type	
GUIDRV_FLEXCOLOR_IF_TYPE_I	Uses lines DB7 to DB0for register access and lines DB17-DB0 for data access. (default)
GUIDRV_FLEXCOLOR_IF_TYPE_II	Uses lines DB8 to DB1 for register access and lines DB17-DB0 for data access.

Additional information

The difference between the interfaces affects the register access to the controller. Normally there are 2 kinds of possible interfaces available when working with the 18 bit bus interface. `TYPE_I` uses the lines D7 to D0 for register access whereas `TYPE_II` uses the lines D8 to D1.

GUIDRV_FlexColor_SetReadFunc66709_B16()

Description

Sets the function(s) to be used for reading back pixel data.

Prototype

```
void GUIDRV_FlexColor_SetReadFunc66709_B16(GUI_DEVICE * pDevice, int Func);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Func	Type of the interface to be used. See possible types below.

Table 33.49: GUIDRV_FlexColor_SetReadFunc66709_B16() parameter list

Permitted values for parameter Func	
GUIDRV_FLEXCOLOR_READ_FUNC_I	3 cycles and data conversion required. (default)
GUIDRV_FLEXCOLOR_READ_FUNC_II	2 cycles and no conversion required.
GUIDRV_FLEXCOLOR_READ_FUNC_III	3 cycles and data conversion required.

Additional information

The difference between the interfaces affects only reading back pixels. The right interface depends on the used controller.

GUIDRV_FLEXCOLOR_READ_FUNC_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-
3rd	G5	G4	G3	G2	G1	G0	-	-	R4	R3	R2	R1	R0	-	-	-

Table 33.50: 66709 - GUIDRV_FLEXCOLOR_READ_FUNC_I

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

Table 33.51: 66709 - GUIDRV_FLEXCOLOR_READ_FUNC_II

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_III

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	-	-	-	G5	G4	G3	G2	G1	G0	-	-
3rd	R4	R3	R2	R1	R0	-	-	-	-	-	-	-	-	-	-	-

Table 33.52: 66709 - GUIDRV_FLEXCOLOR_READ_FUNC_III

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetReadFunc66712_B9()

GUIDRV_FlexColor_SetReadFunc66715_B9()

Description

Sets the function(s) to be used for reading back pixel data.

Prototype

```
void GUIDRV_FlexColor_SetReadFunc66712_B16(GUI_DEVICE * pDevice, int Func);
void GUIDRV_FlexColor_SetReadFunc66715_B16(GUI_DEVICE * pDevice, int Func);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Table 33.53: GUIDRV_FlexColor_SetReadFunc66712_B9() / GUIDRV_FlexColor_SetReadFunc66715_B9() parameter list

Permitted values for parameter Func	
GUIDRV_FLEXCOLOR_READ_FUNC_I	3 cycles and data conversion required. (default)
GUIDRV_FLEXCOLOR_READ_FUNC_II	3 cycles and data conversion required.

Additional information

The right function to be used depends on the behavior of the used controller.

GUIDRV_FLEXCOLOR_READ_FUNC_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	B5	B4	B3	B2	B1	B0	G5	G4	G3
3rd	-	-	-	-	-	-	-	G2	G1	G0	R5	R4	R3	R2	R1	R0

Table 33.54: 66712 / 66715 - B9 - GUIDRV_FLEXCOLOR_READ_FUNC_I

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B5	B4	B3	B2	B1	B0	-	-	G5	G4	G3	G2	G1	G0	-	-
3rd	R5	R4	R3	R2	R1	R0	-	-	-	-	-	-	-	-	-	-

Table 33.55: 66712 / 66715 - B9 - GUIDRV_FLEXCOLOR_READ_FUNC_II

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetReadFunc66712_B16()

GUIDRV_FlexColor_SetReadFunc66715_B16()

Description

Sets the function(s) to be used for reading back pixel data.

Prototype

```
void GUIDRV_FlexColor_SetReadFunc66712_B16(GUI_DEVICE * pDevice, int Func);
```

Parameter	Description
`pDevice`	Pointer to the device to configure.
`Type`	Type of the interface to be used. See possible types below.

Table 33.56: GUIDRV_FlexColor_SetReadFunc66712_B16() / GUIDRV_FlexColor_SetReadFunc66715_B16() parameter list

Permitted values for parameter `Func`	
`GUIDRV_FLEXCOLOR_READ_FUNC_I`	4 cycles and data conversion required. (default)
`GUIDRV_FLEXCOLOR_READ_FUNC_II`	4 cycles and data conversion required.
`GUIDRV_FLEXCOLOR_READ_FUNC_III`	3 cycles and data conversion required.

Additional information

The right function to be used depends on the behavior of the used controller.

GUIDRV_FLEXCOLOR_READ_FUNC_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-
3rd	-	-	-	-	-	-	-	-	G5	G4	G3	G2	G1	G0	-	-
4th	-	-	-	-	-	-	-	-	R4	R3	R2	R1	R0	-	-	-

Table 33.57: 66712 / 66715 - B16 - GUIDRV_FLEXCOLOR_READ_FUNC_I

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	-	G5	G4	G3	G2	G1	G0	-	-
3rd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-
4th	-	-	-	-	-	-	-	-	R4	R3	R2	R1	R0	-	-	-

Table 33.58: 66712 / 66715 - B16 - GUIDRV_FLEXCOLOR_READ_FUNC_II

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_III

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	-	-	-	G5	G4	G3	G2	G1	G0	-	-
3rd	R4	R3	R2	R1	R0	-	-	-	-	-	-	-	-	-	-	-

Table 33.59: 66712 / 66715 - B16 - GUIDRV_FLEXCOLOR_READ_FUNC_III

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetReadFunc66720_B16()

Description

Sets the function(s) to be used for reading back pixel data.

Prototype

```
void GUIDRV_FlexColor_SetReadFunc66720_B16(GUI_DEVICE * pDevice, int Func);
```

Parameter	Description
`pDevice`	Pointer to the device to configure.
`Type`	Type of the interface to be used. See possible types below.

Table 33.60: GUIDRV_FlexColor_SetReadFunc66720_B16() parameter list

Permitted values for parameter `Func`	
`GUIDRV_FLEXCOLOR_READ_FUNC_I`	3 cycles and data conversion required. (default)
`GUIDRV_FLEXCOLOR_READ_FUNC_II`	2 cycles and no conversion required.

Additional information

The right function to be used depends on the behavior of the used controller. Whereas `..._FUNC_I` extracts the index value by assembling it from the second and third word received from the controller, `..._FUNC_II` uses the second word as it is. Please note that the right interface depends on the behavior of the used controller.

GUIDRV_FLEXCOLOR_READ_FUNC_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	-	-	-	-	-	-	-	-	B4	B3	B2	B1	B0	-	-	-
3rd	G5	G4	G3	G2	G1	G0	-	-	R4	R3	R2	R1	R0	-	-	-

Table 33.61: 66720 - GUIDRV_FLEXCOLOR_READ_FUNC_I

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

Table 33.62: 66720 - GUIDRV_FLEXCOLOR_READ_FUNC_II

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetReadFunc66772_B8()

Description

Sets the function(s) to be used for reading back pixel data.

Prototype

```
void GUIDRV_FlexColor_SetReadFunc66772_B8(GUI_DEVICE * pDevice, int Func);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Table 33.63: GUIDRV_FlexColor_SetReadFunc66772_B8()

Permitted values for parameter Func	
GUIDRV_FLEXCOLOR_READ_FUNC_I	3 cycles and no conversion required. (default)
GUIDRV_FLEXCOLOR_READ_FUNC_II	3 cycles and conversion required.

Additional information

The right function to be used depends on the behavior of the used controller. Whereas ..._FUNC_I extracts the index value by assembling it from the second and third word received from the controller, ..._FUNC_II uses the second word as it is. Please note that the right interface depends on the behavior of the used controller.

GUIDRV_FLEXCOLOR_READ_FUNC_I

Cycle	D7	D6	D5	D4	D3	D2	D1	D0
1st								
2nd	B4	B3	B2	B1	B0	G5	G4	G3
3rd	G2	G1	G0	R4	R3	R2	R1	R0

Table 33.64: 66772 - B8 - GUIDRV_FLEXCOLOR_READ_FUNC_I

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_II

Cycle	D7	D6	D5	D4	D3	D2	D1	D0
1st								
2nd	R4	R3	R2	R1	R0	G5	G4	G3
3rd	G2	G1	G0	B4	B3	B2	B1	B0

Table 33.65: 66772 - B8 - GUIDRV_FLEXCOLOR_READ_FUNC_II

In dependence of controller settings red and blue could be swapped.

GUIDRV_FlexColor_SetReadFunc66772_B16()

Description

Sets the function(s) to be used for reading back pixel data.

Prototype

```
void GUIDRV_FlexColor_SetReadFunc66772_B16(GUI_DEVICE * pDevice, int Func);
```

Parameter	Description
pDevice	Pointer to the device to configure.
Type	Type of the interface to be used. See possible types below.

Table 33.66: GUIDRV_FlexColor_SetReadFunc66772_B16()

Permitted values for parameter Func	
GUIDRV_FLEXCOLOR_READ_FUNC_I	2 cycles and no conversion required. (default)
GUIDRV_FLEXCOLOR_READ_FUNC_II	2 cycles and conversion required.

Additional information

The right function to be used depends on the behavior of the used controller. Whereas ..._FUNC_I extracts the index value by assembling it from the second and third word received from the controller, ..._FUNC_II uses the second word as it is. Please note that the right interface depends on the behavior of the used controller.

GUIDRV_FLEXCOLOR_READ_FUNC_I

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	B4	B3	B2	B1	B0	G5	G4	G3	G2	G1	G0	R4	R3	R2	R1	R0

Table 33.67: 66772 - B16 - GUIDRV_FLEXCOLOR_READ_FUNC_I

In dependence of controller settings red and blue could be swapped.

GUIDRV_FLEXCOLOR_READ_FUNC_II

Cycle	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1st	Dummy read															
2nd	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

Table 33.68: 66772 - B16 - GUIDRV_FLEXCOLOR_READ_FUNC_II

In dependence of controller settings red and blue could be swapped.

33.7.5 GUIDRV_IST3088

Supported hardware

Controllers

This driver works with the following display controllers:

- Integrated Solutions Technology IST3088, IST3257

Bits per pixel

The supported color depth is 4 bpp.

Interfaces

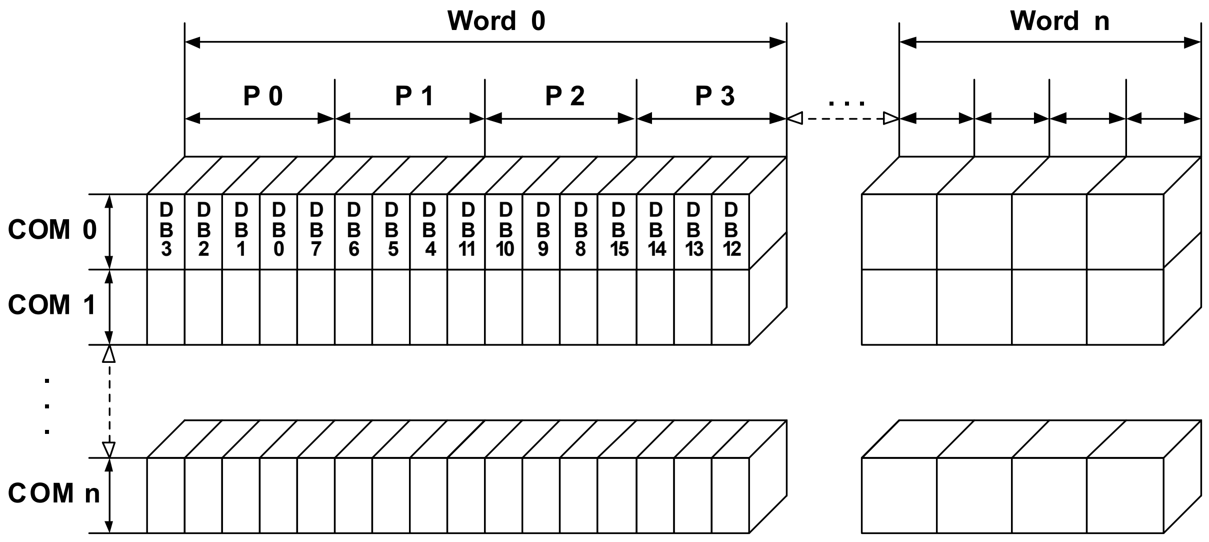
The driver supports the 16-bit indirect interface.

Driver selection

To use GUIDRV_IST3088 for the given display, the following command should be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_IST3088_4, GUICC_4, 0, 0);
```

Display data RAM organization



The delineation above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

This display driver can be used with and without a display data cache, containing a complete copy of the content of the display data RAM. The amount of memory (in bytes) used by the cache is:

```
LCD_XSIZE * LCD_YSIZE / 2.
```


Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_IST3088_SetBus16()	Tells the driver to use the 16 bit indirect interface and passes pointer to a GUI_PORT_API structure to the driver.

Table 33.69: Run-time configuration

GUIDRV_IST3088_SetBus16()

Description

Tells the driver to use the 16 bit indirect interface and passes a pointer to a GUI_PORT_API structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_IST3088_SetBus16(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.

Table 33.70: GUIDRV_IST3088_SetBus16() parameter list

Required GUI_PORT_API routines

Element	Data type
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)

Table 33.71: Required GUI_PORT_API routines

Special requirements

The driver needs to work in the fixed palette mode GUICC_4. The driver does not work with other palettes or fixed palette modes. You should use GUICC_4 as color conversion.

33.7.6 GUIDRV_Lin

This driver supports all display controllers with linear video memory accessible via direct interface. It can be used with and without a display controller. The driver does only manage the contents of the video memory. It does not send any commands to the display controller or assumes any specific registers. So it is independent of the register interface of the display controller and can be used for managing each linear mapped video memory.

Supported hardware

Controllers

The driver supports all systems with linear mapped video memory.

Bits per pixel

Supported color depths are 1, 2, 4, 8, 16, 24 and 32 bits per pixel.

Interfaces

The driver supports a full bus interface from the CPU to the video memory. The video memory needs to be accessible 8, 16 or 32 bit wise.

Color depth and display orientation

The driver consists of several files. They are named `_[O]_BPP.c`. where the optional `'O'` stands for the desired display orientation and `'BPP'` for the color depth. The following table shows the driver files and the configuration macros which should be used to create and link the driver during the initialization:

Identifier	Color depth and orientation
GUIDRV_LIN_1	1bpp, default orientation
GUIDRV_LIN_2	2bpp, default orientation
GUIDRV_LIN_4	4bpp, default orientation
GUIDRV_LIN_8	8bpp, default orientation
GUIDRV_LIN_OX_8	8bpp, X axis mirrored
GUIDRV_LIN_OXY_8	8bpp, X and Y axis mirrored
GUIDRV_LIN_16	16bpp, default orientation
GUIDRV_LIN_OX_16	16bpp, X axis mirrored
GUIDRV_LIN_OXY_16	16bpp, X and Y axis mirrored
GUIDRV_LIN_OY_16	16bpp, Y axis mirrored
GUIDRV_LIN_OS_16	16bpp, X and Y swapped
GUIDRV_LIN_OSX_16	16bpp, X axis mirrored, X and Y swapped
GUIDRV_LIN_OSY_16	16bpp, Y axis mirrored, X and Y swapped
GUIDRV_LIN_24	24bpp, default orientation
GUIDRV_LIN_OX_24	24bpp, X axis mirrored
GUIDRV_LIN_OXY_24	24bpp, X and Y axis mirrored
GUIDRV_LIN_OY_24	24bpp, Y axis mirrored
GUIDRV_LIN_OS_24	24bpp, X and Y swapped
GUIDRV_LIN_OSX_24	24bpp, X axis mirrored, X and Y swapped
GUIDRV_LIN_OSY_24	24bpp, Y axis mirrored, X and Y swapped
GUIDRV_LIN_32	32bpp, default orientation
GUIDRV_LIN_OX_32	32bpp, X axis mirrored
GUIDRV_LIN_OXY_32	32bpp, X and Y axis mirrored
GUIDRV_LIN_OY_32	32bpp, Y axis mirrored
GUIDRV_LIN_OS_32	32bpp, X and Y swapped
GUIDRV_LIN_OSX_32	32bpp, X axis mirrored, X and Y swapped
GUIDRV_LIN_OSY_32	32bpp, Y axis mirrored, X and Y swapped

Table 33.72: Color depth and display orientation

The table above shows identifiers which can be used to select the driver. Each combination of orientation and color depth is possible. Please note that currently not all combinations are shipped with the driver. If the required combination is not available, please send a request to obtain the required combination.

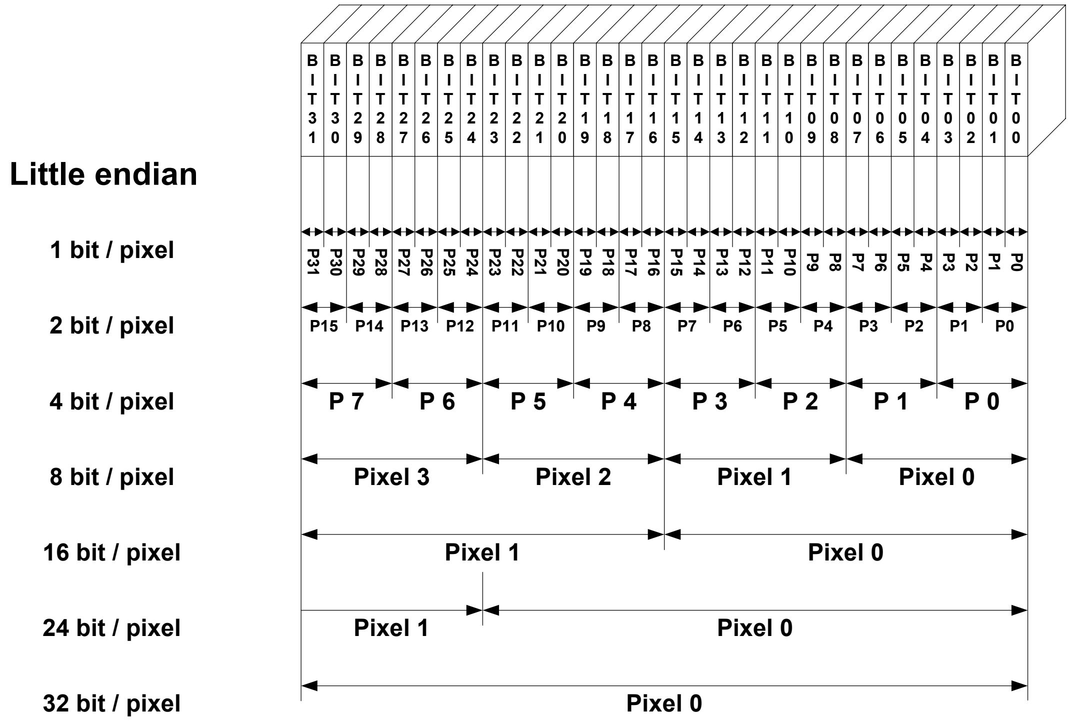
Driver selection

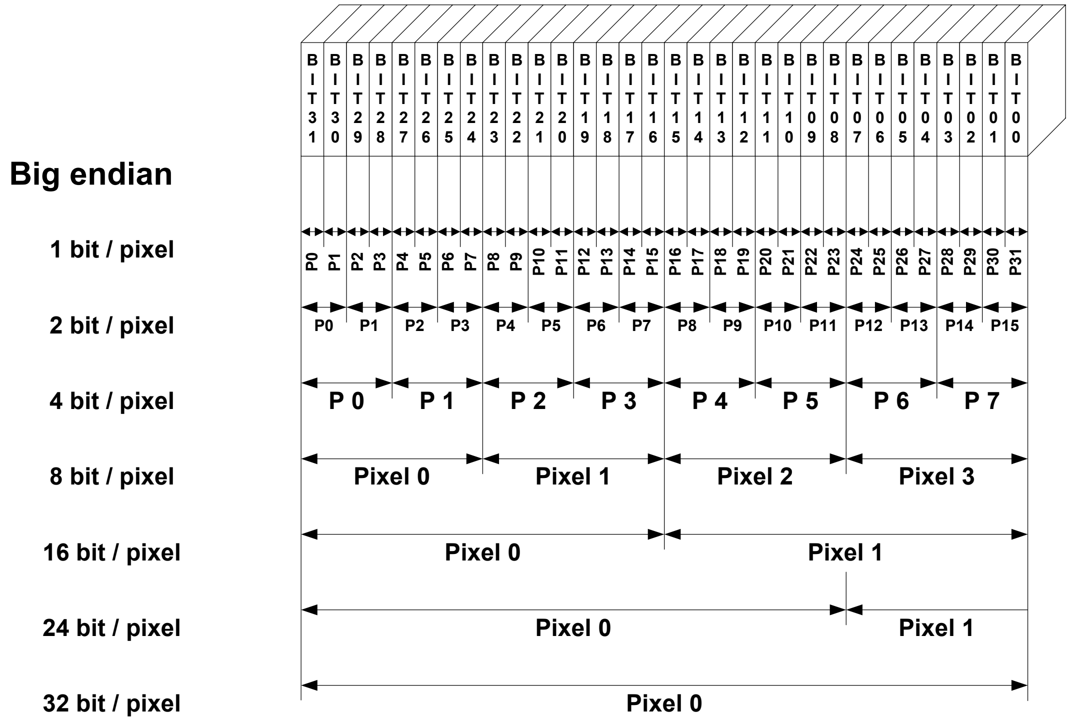
To use for the given display, the following command can be used e.g.:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_LIN_OX_16, GUICC_565, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization





The picture above shows the relation between the display memory and the pixels of the LCD in terms of the color depth and the endian mode.

Little endian video mode

Least significant bits are used and output first. The least significant bits are for the first (left-most) pixel.

Big endian video mode

Most significant bits are used and output first. The most significant bits are for the first (left-most) pixel.

RAM requirements

None.

Compile-time configuration

The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_ENDIAN_BIG	Should be set to 1 for big endian mode, 0 (default) for little endian mode.

Table 33.73: Compile-time configuration macros

Run-time configuration

The following table lists the available run-time configuration routines:

Routine	Description
LCD_SetDevFunc()	Can be used to set optional or custom defined routines.
LCD_SetSizeEx()	Changes the size of the visible area.
LCD_SetVRAMAddrEx()	Changes the video RAM start address.
LCD_SetVSizeEx()	Changes the size of the virtual display area.

Table 33.74: Run-time configuration API list

Commands supported by LCD_SetDevFunc()

The following table shows the supported values of the function:

Value	Description
LCD_DEVFUNC_COPYBUFFER	Can be used to set a custom defined routine for copying buffers. Makes only sense in combination with multiple buffers.
LCD_DEVFUNC_COPYRECT	Can be used to set a custom defined routine for copying rectangular areas of the display.
LCD_DEVFUNC_DRAWBMP_1BPP	Can be used to set a custom routine for drawing 1bpp bitmaps.
LCD_DEVFUNC_DRAWBMP_8BPP	Can be used to set a custom routine for drawing 8bpp bitmaps.
LCD_DEVFUNC_FILLRECT	Can be used to set a custom defined routine for filling rectangles. Makes sense if for example a BitBLT engine should be used for filling operations.

Table 33.75: Commands supported by LCD_SetDevFunc()

Further information about the LCD layer routines can be found under “LCD layer routines” on page 1159.

Configuration example

The following shows how to create a display driver device with this driver and how to configure it:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,     // Display driver
                           GUICC_8666,       // Color conversion
                           0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx    (0, 320,  240);           // Physical display size in pixels
  LCD_SetVSizeEx   (0, 320,  480);           // Virtual  display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0x20000000); // Video RAM start address
}
```


Using the Lin driver in systems with cache memory

The rules to follow are quite simple:

Rule 1

All caches (if applicable, as in your case) should be fully enabled. This means I- and D- caches in systems with separate caches.

Rule 2

All code and data should be placed in cacheable areas to achieve maximum performance. If other parts of the application require some or all data to be placed in non-cacheable areas, this is not a problem but may degrade performance.

Rule 3

The cache settings for the frame buffer memory (which is really a shared memory area, accessed by both the CPU and the LCD-controller DMA) should make sure, that write operations are 'write-through' operations. The physical memory should be always up to date, so that the DMA-access of the LCD-controller always get the current content of the frame buffer. In case of a 'write-back' cache a write operation only changes the content of the cache, which is written to the physical memory not before the cache location is superseded.
In many systems with MMU, this can be achieved by mapping the RAM twice into the virtual address space: At its normal address, the RAM is cacheable and bufferable, at the second address, it is cacheable but not bufferable. The address of the VRAM given to the driver should be the non bufferable address.
If the CPU does not support a 'write-through' cache the frame buffer memory needs to be uncached.

33.7.7 GUIDRV_S1D13748

Supported hardware

Controllers

This driver has been tested with the Epson S1D13748.

Bits per pixel

The supported color depth is 16 bpp.

Interfaces

The driver supports the 16-bit indirect interface.

Basic function

The driver currently supports indirect mode only. Only 2 registers, namely register 0 and 2 are used.

Hardware interface

AB[1] = GND
AB[2] = Used as Address pin
AB[3] = GND

AB[3:0]	Register
000	Index
001	Status
010	Data
011	Reserved
100	GPIO Status
101	GPIO Config
110	GPIO Input Enable
111	GPIO Pull-down Control

Table 33.76: Hardware interface

Reset

The RESET pin should be connected to the system reset. The RESET pin of the Microcontroller / CPU is usually called NRESET.

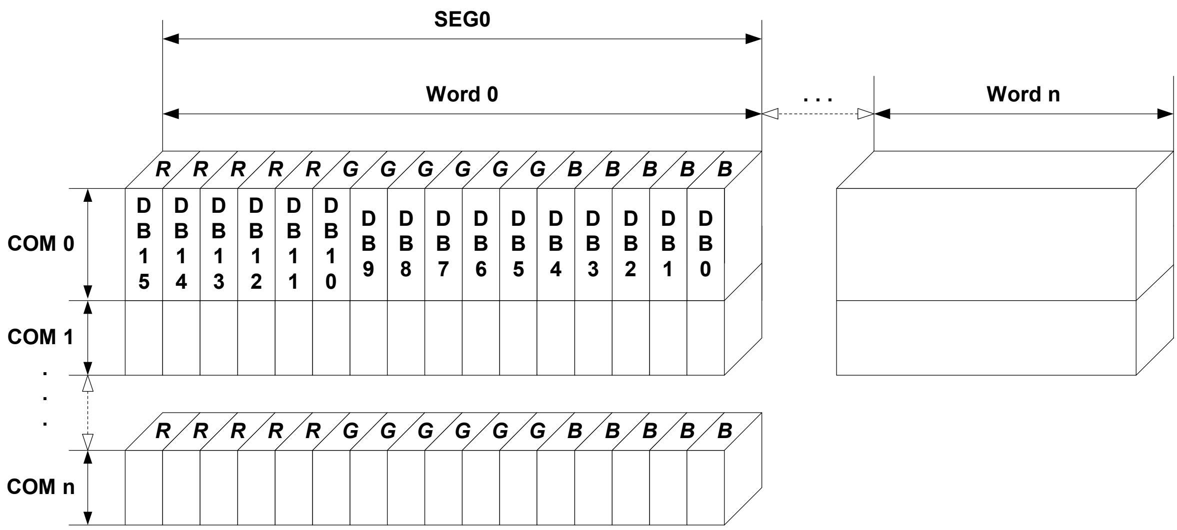
Driver selection

To use GUIDRV_S1D13748 for the given display, the following command should be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D13748, GUICC_M565, 0, 0);
```

Display data RAM organization

16 bits per pixel, fixed palette = 565



The delineation above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

Approximately 500 bytes.

Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_S1D13748_Config()	Passes a pointer to a CONFIG_S1D13748 structure to the driver.
GUIDRV_S1D13748_SetBus_16()	Configures the driver to use the 16 bit indirect interface by passing a pointer to a GUI_PORT_API structure.

Table 33.77: Run-time configuration

GUIDRV_S1D13748_Config()

Description

Configures the driver to work according to the passed `CONFIG_S1D13748` structure.

Prototype

```
void GUIDRV_S1D13748_Config(GUI_DEVICE       * pDevice,
                            CONFIG_S1D13748 * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a `CONFIG_S1D13748` structure described below.

Table 33.78: GUIDRV_S1D13748_Config() parameter list

Elements of structure CONFIG_S1D13748

Data type	Element	Description
U32	BufferOffset	This offset added to the VideoRAM start address, results in the start address used for the selected PIP layer.
int	UseLayer	PIP layer to be used.

Table 33.79: CONFIG_S1D13748 element list

GUIDRV_S1D13748_SetBus_16()

Description

Tells the driver to use the 16 bit indirect interface and passes a pointer to a GUI_PORT_API structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_S1D13748_SetBus_16(GUI_DEVICE   * pDevice,
                               GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.

Table 33.80: GUIDRV_S1D13748_SetBus_16() parameter list

Required GUI_PORT_API routines

Data type	Element	Description
void (*)(U16 Data)	pfWrite16_A0	Pointer to a function which writes one word to the controller with C/D line low.
void (*)(U16 Data)	pfWrite16_A1	Pointer to a function which writes one word to the controller with C/D line high.
void (*)(U16 * pData, int NumItems)	pfWriteM16_A1	Pointer to a function which writes multiple words to the controller with C/D line high.
U16 (*)(void)	pfRead16_A1	Pointer to a function which reads one word from the controller with C/D line high.
void (*)(U16 * pData, int NumItems)	pfReadM16_A1	Pointer to a function which reads multiple words from the controller with C/D line high.

Table 33.81: Required GUI_PORT_API routines

Special requirements

The driver needs to work with the fixed palette mode GUICC_M565. The driver does not work with other palettes or fixed palette modes.

33.7.8 GUIDRV_S1D13781

Supported hardware

Controllers

This driver has been tested with the Epson S1D13781.

Bits per pixel

Currently the supported color depth is 8 bpp. This could be enhanced on demand.

Interfaces

Currently the driver supports only the 8-bit indirect serial host interface. Could be enhanced on demand.

Display orientation

The driver can be used with different orientations. The following table shows the configuration macros which can be used to create and link the driver during the initialization:

Identifier	Color depth and orientation
GUIDRV_S1D13781_8C0	8bpp, default orientation
GUIDRV_S1D13781_OXY_8C0	8bpp, X and Y axis mirrored
GUIDRV_S1D13781_OSY_8C0	8bpp, X axis mirrored, X and Y swapped
GUIDRV_S1D13781_OSX_8C0	8bpp, Y axis mirrored, X and Y swapped

Table 33.82: Display orientation

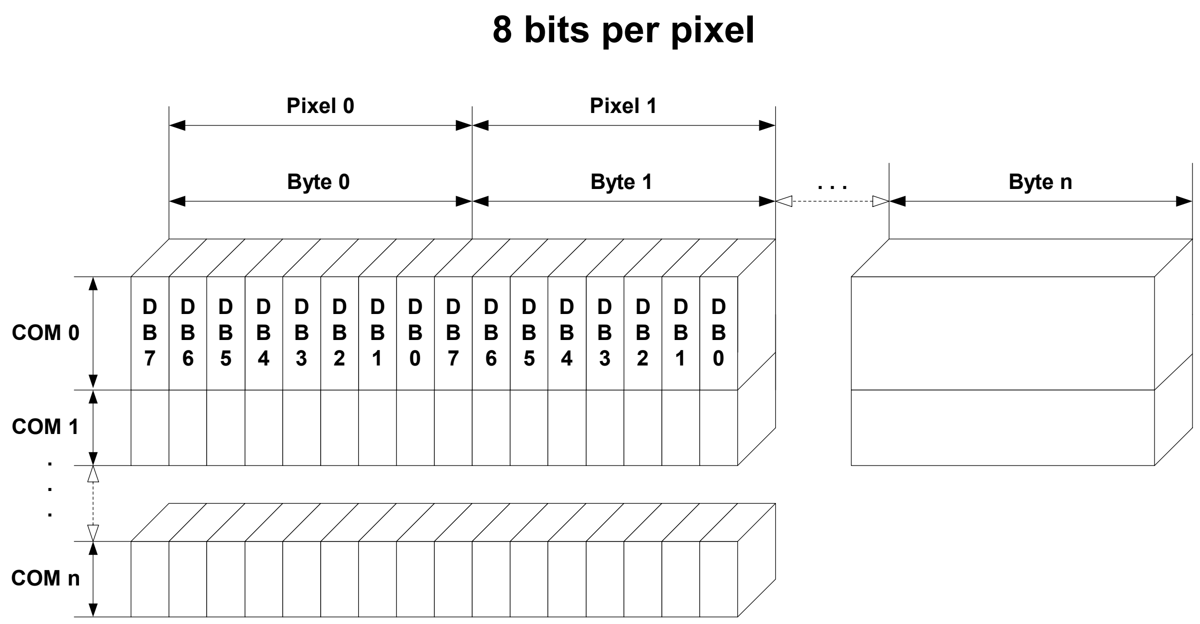
The table above shows identifiers which can be used to select the driver.

Driver selection

To use GUIDRV_S1D13781 for the given display, the following command should be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D13781, GUICC_8666, 0, 0);
```

Display data RAM organization



The delineation above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

Approximately 1KByte.

Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_S1D13781_Config()	Passes a pointer to a CONFIG_S1D13781 structure to the driver.
GUIDRV_S1D13781_SetBusSPI()	Configures the driver to use the 8 bit indirect serial host interface by passing a pointer to a GUI_PORT_API structure.

Table 33.83: Run-time configuration

GUIDRV_S1D13781_Config()

Description

Configures the driver to work according to the passed CONFIG_S1D13781 structure.

Prototype

```
void GUIDRV_S1D13781_Config(GUI_DEVICE      * pDevice,
                            CONFIG_S1D13781 * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a CONFIG_S1D13781 structure described below.

Table 33.84: GUIDRV_S1D13781_Config() parameter list

Elements of structure CONFIG_S1D13781

Data type	Element	Description
U32	BufferOffset	This offset added to the VideoRAM start address, results in the start address used for the selected PIP layer.
int	WriteBufferSize	Number of bytes used for the write buffer. The buffer should be large enough to be able to store at least one line of data + 5 bytes. Because the layer size can be changed dynamically, it is required to set up the buffer size during the configuration. The default value of the buffer size is 500 bytes.
int	UseLayer	Should be 1 if PIP layer should be used.
int	WaitUntilVNDP	Used for Multiple Buffering configurations only. If set to 1 the driver waits until the next vertical non display period has been reached. This can be used to reduce flickering effects with fast animations.

Table 33.85: CONFIG_S1D13781 element list

GUIDRV_S1D13781_SetBusSPI()

Description

Tells the driver to use the 8 bit indirect serial host interface and passes a pointer to a GUI_PORT_API structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_S1D13781_SetBusSPI(GUI_DEVICE * pDevice,
                               GUI_PORT_API * pHW_API);
```


Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.

Table 33.86: GUIDRV_S1D13781_SetBusSPI() parameter list

Required GUI_PORT_API routines

Data type	Element	Description
void (*)(U8 Data)	pfWrite8_A0	Pointer to a function which writes one byte to the controller with C/D line low.
void (*)(U8 Data)	pfWrite8_A1	Pointer to a function which writes one byte to the controller with C/D line high.
void (*)(U8 * pData, int NumItems)	pfWriteM8_A1	Pointer to a function which writes multiple bytes to the controller with C/D line high.
U8 (*)(void)	pfRead8_A1	Pointer to a function which reads one byte from the controller with C/D line high.
void (*)(U8 * pData, int NumItems)	pfReadM8_A1	Pointer to a function which reads multiple bytes from the controller with C/D line high.
void (*)(U8 NotActive)	pfSetCS	Routine which is able to toggle the CS signal of the controller: NotActive = 1 means CS = high NotActive = 0 means CS = low

Table 33.87: Required GUI_PORT_API routines

Optional functions

The following table shows the optional LCD-functions which are available with this driver:

Routine	Description
GUI_GetLayerPosEx()	Returns the position of the given layer.
GUI_SetLayerPosEx()	Sets the position of the given layer.
GUI_SetLayerSizeEx()	Sets the size of the given layer.
GUI_SetLayerVisEx()	Sets the visibility of the given layer.
LCD_SetAlphaEx()	Sets the alpha value for the given layer.
LCD_SetChromaEx()	Sets the key color to be used. Only the first color passed by the function is used.
LCD_SetChromaModeEx()	Toggles usage of transparent key color. 1 enables transparent key color, 0 disables it.

Table 33.88: Optional functions

More details about the optional functions can be found in "MultiLayer API" on page 965.

Additional information

The display driver automatically initializes the following registers:

Register	Description
0x60824	xSize of main layer.
0x60828	ySize of main layer.
0x60840	Main layer settings.

Table 33.89: Register initialization

This means the above registers do not need to be initialized by the applications initialization code for the display controller.

33.7.9 GUIDRV_S1D15G00

Supported hardware

Controllers

The driver supports the Epson S1D15G00 controller.

Bits per pixel

Supported color depth is 12bpp.

Interfaces

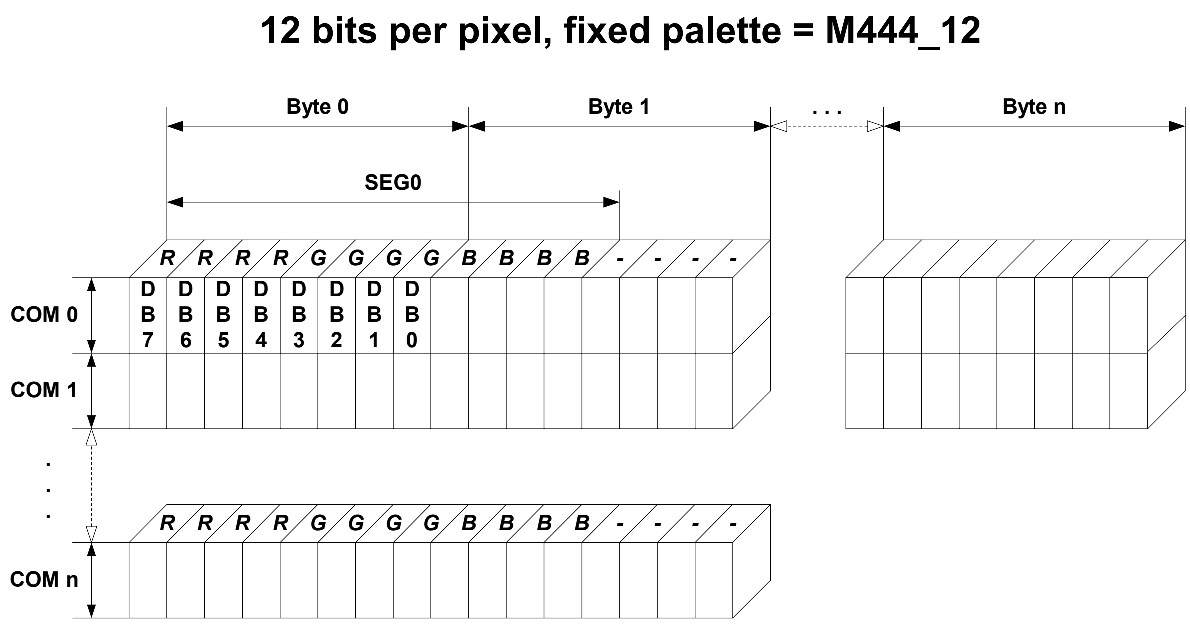
The driver supports the 8 bit indirect interface.

Driver selection

To use GUIDRV_S1D15G00 for the given display, the following command should be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D15G00, GUICC_M444_12, 0, 0);
```

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

This display driver can be used with and without a display data cache, containing a complete copy of the contents of the LCD data RAM. The amount of memory used by the cache is:

```
LCD_XSIZE x LCD_YSIZE x 2 bytes
```

Using a cache is recommended only if a lot of drawing operations uses the XOR drawing mode. A cache would avoid reading the display data in this case. Normally the use of a cache is not recommended.

Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_S1D15G00_Config()	Passes a pointer to a CONFIG_S1D15G00 structure to the driver.
GUIDRV_S1D15G00_SetBus8()	Tells the driver to use the 8 bit indirect interface and passes pointer to a GUI_PORT_API structure to the driver.

Table 33.90: Run-time configuration

GUIDRV_S1D15G00_Config()

Description

Passes a pointer to a `CONFIG_S1D15G00` structure to the driver.

Prototype

```
void GUIDRV_S1D15G00_Config(GUI_DEVICE      * pDevice,
                            CONFIG_S1D15G00 * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a `CONFIG_S1D15G00` structure described below.

Table 33.91: GUIDRV_S1D15G00_Config() parameter list

Elements of structure CONFIG_S1D15G00

Data type	Element	Description
int	FirstSEG	First segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	FirstCOM	First common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	UseCache	Enables or disables use of a data cache. Should be set to 1 for enabling and to 0 for disabling.

Table 33.92: CONFIG_S1D15G00 element list

GUIDRV_S1D15G00_SetBus8()

Description

Tells the driver to use the 8 bit indirect interface and passes a pointer to a `GUI_PORT_API` structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_S1D15G00_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a `GUI_PORT_API` structure. See required routines below.

Table 33.93: GUIDRV_S1D15G00_SetBus8() parameter list

Required GUI_PORT_API routines

Data type	Element	Description
void (*)(U8 Data)	pfWrite8_A0	Pointer to a function which writes one byte to the controller with C/D line low.
void (*)(U8 Data)	pfWrite8_A1	Pointer to a function which writes one byte to the controller with C/D line high.
void (*)(U8 * pData, int NumItems)	pfWriteM8_A1	Pointer to a function which writes multiple bytes to the controller with C/D line high.
U8 (*)(void)	pfRead8_A1	Pointer to a function which reads one byte from the controller with C/D line high.

Table 33.94: Required GUI_PORT_API routines

Configuration Example

```
#define XSIZE 130
#define YSIZE 130

GUI_PORT_API _PortAPI;

void LCD_X_Config(void) {
  GUI_DEVICE * pDevice;
  CONFIG_S1D15G00 Config = {0};

  //
  // Set display driver and color conversion for 1st layer
  //
  pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D15G00, GUICC_M444_12, 0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx (0, XSIZE,  YSIZE);
  LCD_SetVSizeEx(0, XSIZE,  YSIZE);
  //
  // Driver specific configuration
  //
  Config.FirstCOM  = 2;
  GUIDRV_S1D15G00_Config(pDevice, &Config);
  //
  // Setup hardware access routines
  //
  _PortAPI.pfWrite8_A0  = _Write_A0;
  _PortAPI.pfWrite8_A1  = _Write_A1;
  _PortAPI.pfWriteM8_A1 = _WriteM_A1;
  GUIDRV_S1D15G00_SetBus8(pDevice, &_PortAPI);
}
```


33.7.10 GUIDRV_SLin

Supported hardware

Controllers

The driver works with the following display controllers:
- Epson S1D13700, S1D13305 (indirect interface only!)
- RAIO 8835
- Solomon SSD1325, SSD1848
- Ultrachip UC1617
- Toshiba T6963

Bits per pixel

Supported color depth is 1, 2 and 4 bits per pixel. Please consider that the supported controllers normally do not support all possible color depths to be used with the driver.

Interfaces

The driver supports the 8 bit indirect interface.

Color depth and display orientation

The driver can be used with different orientations and color depths. The following table shows the configuration macros which can be used to create and link the driver during the initialization:

Identifier	Color depth and orientation
GUIDRV_SLIN_1	1bpp, default orientation
GUIDRV_SLIN_OY_1	1bpp, Y axis mirrored
GUIDRV_SLIN_OX_1	1bpp, X axis mirrored
GUIDRV_SLIN_OXY_1	1bpp, X and Y axis mirrored
GUIDRV_SLIN_OS_1	1bpp, X and Y swapped
GUIDRV_SLIN_OSY_1	1bpp, X and Y swapped, Y axis mirrored
GUIDRV_SLIN_OSX_1	1bpp, X and Y swapped, X axis mirrored
GUIDRV_SLIN_OSXY_1	1bpp, X and Y swapped, X and Y axis mirrored
GUIDRV_SLIN_2	2bpp, default orientation
GUIDRV_SLIN_OY_2	2bpp, Y axis mirrored
GUIDRV_SLIN_OX_2	2bpp, X axis mirrored
GUIDRV_SLIN_OXY_2	2bpp, X axis mirrored, Y axis mirrored
GUIDRV_SLIN_OS_2	2bpp, X and Y swapped
GUIDRV_SLIN_OSY_2	2bpp, X and Y swapped, Y axis mirrored
GUIDRV_SLIN_OSX_2	2bpp, X and Y swapped, X axis mirrored
GUIDRV_SLIN_OSXY_2	2bpp, X and Y swapped, Y and X axis mirrored
GUIDRV_SLIN_4	4bpp, default orientation
GUIDRV_SLIN_OY_4	4bpp, Y axis mirrored
GUIDRV_SLIN_OX_4	4bpp, X axis mirrored
GUIDRV_SLIN_OXY_4	4bpp, X axis mirrored, Y axis mirrored
GUIDRV_SLIN_OS_4	4bpp, X and Y swapped
GUIDRV_SLIN_OSY_4	4bpp, X and Y swapped, Y axis mirrored
GUIDRV_SLIN_OSX_4	4bpp, X and Y swapped, X axis mirrored
GUIDRV_SLIN_OSXY_4	4bpp, X and Y swapped, Y and X axis mirrored

Table 33.95: Color depth and display orientation

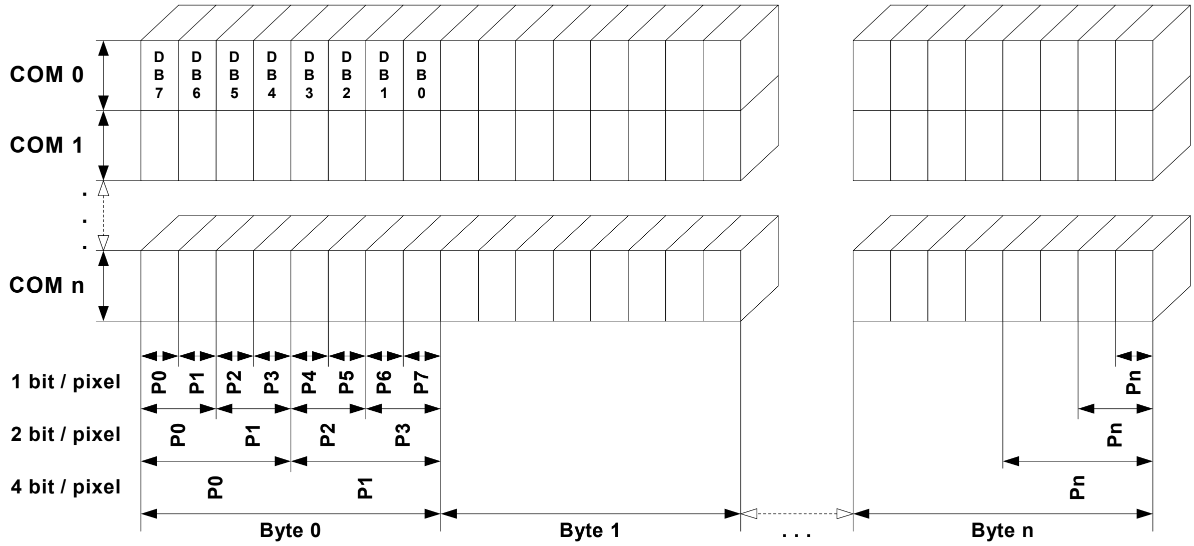
Driver selection

To use GUIDRV_SLin for the given display, the following command can be used e.g.:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SLIN_OX_1, GUICC_1, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display.

RAM requirements

This display driver may be used with or without a display data cache, containing a complete copy of the frame buffer. If no cache is used, the driver only requires app. 256 bytes of RAM.
It is recommended to use this driver with a data cache for faster LCD-access. The additional amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) = BitsPerPixel * (LCD_XSIZE + 7) / 8 * LCD_YSIZE
```

Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_SLin_Config()	Passes a pointer to a CONFIG_SLIN structure to the driver.
GUIDRV_SLin_SetBus8()	Tells the driver to use the 8 bit indirect interface and passes pointer to a GUI_PORT_API structure to the driver.
GUIDRV_SLin_SetS1D13700()	Set up the driver to use one of the following controllers: - Epson S1D13700, S1D13305 - RAIO 8835
GUIDRV_SLin_SetSSD1325()	Set up the driver to use one of the following controllers: - Solomon SSD1325
GUIDRV_SLin_SetSSD1848()	Set up the driver to use one of the following controllers: - Solomon SSD1848
GUIDRV_SLin_SetT6963()	Set up the driver to use one of the following controllers: - Toshiba T6963
GUIDRV_SLin_SetUC1617()	Set up the driver to use one of the following controllers: - Ultrachip UC1617

Table 33.96: Run-time configuration

GUIDRV_SLin_Config()

Description

Passes a pointer to a CONFIG_SLIN structure to the driver.

Prototype

```
void GUIDRV_SLin_Config(GUI_DEVICE * pDevice, CONFIG_SLIN * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a CONFIG_SLIN structure described below.

Table 33.97: GUIDRV_SLin_Config() parameter list

Elements of structure CONFIG_SLIN

Data type	Element	Description
int	FirstSEG	First segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	FirstCOM	First common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	UseCache	Enables or disables use of a data cache. Should be set to 1 for enabling and to 0 for disabling.
int	UseDualScan	Used only for the T6963. This element should be set to 1 in case a dual screen LCD is used.
int	UseMirror	Used only for the SSD1848. Should be normally 1.

Table 33.98: CONFIG_SLIN element list

GUIDRV_SLin_SetBus8()

Description

Tells the driver to use the 16 bit indirect interface and passes a pointer to a GUI_PORT_API structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_Slin_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.

Table 33.99: GUIDRV_SLin_SetBus8() parameter list

Required GUI_PORT_API routines

Element	Data type
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A0	void (*)(U8 * pData, int NumItems)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfRead8_A1	U8 (*)(void)

Table 33.100: Required GUI_PORT_API routines

GUIDRV_SLin_SetS1D13700()

Description

Tells the driver that an Epson S1D13700 or S1D13305 controller should be used. Works also for RAIO 8835.

Prototype

```
void GUIDRV_SLin_SetS1D13700(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.101: GUIDRV_SLin_SetS1D13700() parameter list

GUIDRV_SLin_SetSSD1325()

Description

Tells the driver that a Solomon SSD1325 controller should be used.

Prototype

```
void GUIDRV_SLin_SetSSD1325(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.102: GUIDRV_SLin_SetSSD1325() parameter list

GUIDRV_SLin_SetSSD1848()

Description

Tells the driver that a Solomon SSD1848 controller should be used.

Prototype

```
void GUIDRV_SLin_SetSSD1848(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.103: GUIDRV_SLin_SetSSD1848() parameter list

GUIDRV_SLin_SetT6963()

Description

Tells the driver that a Toshiba T6963 controller should be used.

Prototype

```
void GUIDRV_SLin_SetT6963(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.104: GUIDRV_SLin_SetT6963() parameter list

GUIDRV_SLin_SetUC1617()

Description

Tells the driver that an Ultrachip UC1617 controller should be used.

Prototype

```
void GUIDRV_SLin_SetUC1617(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.105: GUIDRV_SLin_SetUC1617() parameter list

Configuration Example

```
#define XSIZE 320
```



```
#define YSIZE 240

void LCD_X_Config(void) {
  GUI_DEVICE * pDevice;
  CONFIG_SLIN  Config  = {0};
  GUI_PORT_API PortAPI = {0};

  //
  // Set display driver and color conversion
  //
  pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SLIN_2, GUICC_2, 0, 0);
  //
  // Common display driver configuration
  //
  LCD_SetSizeEx (0, XSIZE, YSIZE);
  LCD_SetVSizeEx(0, XSIZE, YSIZE);
  //
  // Driver specific configuration
  //
  Config.UseCache  = 1;
  GUIDRV_SLin_Config(pDevice, &Config);
  //
  // Select display controller
  //
  GUIDRV_SLin_SetS1D13700(pDevice);
  //
  // Setup hardware access routines
  //
  PortAPI.pfWrite16_A0  = _Write0;
  PortAPI.pfWrite16_A1  = _Write1;
  PortAPI.pfWriteM16_A0 = _WriteM0;
  PortAPI.pfRead16_A1   = _Read1;
  GUIDRV_SLin_SetBus8(pDevice, &PortAPI);
}
```


33.7.11 GUIDRV_SPage

Supported hardware

Controllers

The driver works with the following display controllers:

- Avant Electronics SBN0064G
- Epson S1D15E05, S1D15E06, S1D15605, S1D15606, S1D15607, S1D15608, S1D15705, S1D15710, S1D15714, S1D15719, S1D15721
- Integrated Solutions Technology IST3020
- New Japan Radio Company NJU6676
- Novatek NT7502, NT7534, NT7538, NT75451
- Samsung S6B0713, S6B0719, S6B0724, S6B1713
- Sino Wealth SH1101A
- Sitronix ST7522, ST75256, ST7565, ST7567, ST7591
- Solomon SSD1303, SSD1305, SSD1306, SSD1805, SSD1815
- Sunplus SPLC501C
- UltraChip UC1601, UC1606, UC1608, UC1611, UC1638, UC1701

Bits per pixel

The driver currently supports 1, 2 and 4 bpp resolutions.

Interfaces

The driver supports the indirect interface (8 bit) of the display controller. Parallel, 4-pin SPI or I2C bus can be used.

Color depth and display orientation

The driver can be used with different orientations and color depths. Each configuration can be used with or without a display driver cache. The following table shows the configuration macros which can be used to create and link the driver during the initialization:

Identifier	Color depth	Cache	Orientation
GUIDRV_SPAGE_1C0	1bpp	No	default
GUIDRV_SPAGE_OY_1C0	1bpp	No	Y axis mirrored
GUIDRV_SPAGE_OX_1C0	1bpp	No	X axis mirrored
GUIDRV_SPAGE_OXY_1C0	1bpp	No	X and Y axis mirrored
GUIDRV_SPAGE_OS_1C0	1bpp	No	X and Y swapped
GUIDRV_SPAGE_OSY_1C0	1bpp	No	X and Y swapped, Y axis mirrored
GUIDRV_SPAGE_OSX_1C0	1bpp	No	X and Y swapped, X axis mirrored
GUIDRV_SPAGE_OSXY_1C0	1bpp	No	X and Y swapped, X and Y axis mirrored
GUIDRV_SPAGE_1C1	1bpp	Yes	default
GUIDRV_SPAGE_OY_1C1	1bpp	Yes	Y axis mirrored
GUIDRV_SPAGE_OX_1C1	1bpp	Yes	X axis mirrored
GUIDRV_SPAGE_OXY_1C1	1bpp	Yes	X and Y axis mirrored
GUIDRV_SPAGE_OS_1C1	1bpp	Yes	X and Y swapped
GUIDRV_SPAGE_OSY_1C1	1bpp	Yes	X and Y swapped, Y axis mirrored
GUIDRV_SPAGE_OSX_1C1	1bpp	Yes	X and Y swapped, X axis mirrored
GUIDRV_SPAGE_OSXY_1C1	1bpp	Yes	X and Y swapped, X and Y axis mirrored
GUIDRV_SPAGE_2C0	2bpp	No	default
GUIDRV_SPAGE_OY_2C0	2bpp	No	Y axis mirrored
GUIDRV_SPAGE_OX_2C0	2bpp	No	X axis mirrored
GUIDRV_SPAGE_OXY_2C0	2bpp	No	X and Y axis mirrored
GUIDRV_SPAGE_OS_2C0	2bpp	No	X and Y swapped
GUIDRV_SPAGE_OSY_2C0	2bpp	No	X and Y swapped, Y axis mirrored
GUIDRV_SPAGE_OSX_2C0	2bpp	No	X and Y swapped, X axis mirrored
GUIDRV_SPAGE_OSXY_2C0	2bpp	No	X and Y swapped, X and Y axis mirrored
GUIDRV_SPAGE_2C1	2bpp	Yes	default

Table 33.106: Color depth and display orientation

Identifier	Color depth	Cache	Orientation
GUIDRV_SPAGE_OY_2C1	2bpp	Yes	Y axis mirrored
GUIDRV_SPAGE_OX_2C1	2bpp	Yes	X axis mirrored
GUIDRV_SPAGE_OXY_2C1	2bpp	Yes	X and Y axis mirrored
GUIDRV_SPAGE_OS_2C1	2bpp	Yes	X and Y swapped
GUIDRV_SPAGE_OSY_2C1	2bpp	Yes	X and Y swapped, Y axis mirrored
GUIDRV_SPAGE_OSX_2C1	2bpp	Yes	X and Y swapped, X axis mirrored
GUIDRV_SPAGE_OSXY_2C1	2bpp	Yes	X and Y swapped, X and Y axis mirrored
GUIDRV_SPAGE_4C0	4bpp	No	default
GUIDRV_SPAGE_OY_4C0	4bpp	No	Y axis mirrored
GUIDRV_SPAGE_OX_4C0	4bpp	No	X axis mirrored
GUIDRV_SPAGE_OXY_4C0	4bpp	No	X and Y axis mirrored
GUIDRV_SPAGE_OS_4C0	4bpp	No	X and Y swapped
GUIDRV_SPAGE_OSY_4C0	4bpp	No	X and Y swapped, Y axis mirrored
GUIDRV_SPAGE_OSX_4C0	4bpp	No	X and Y swapped, X axis mirrored
GUIDRV_SPAGE_OSXY_4C0	4bpp	No	X and Y swapped, X and Y axis mirrored
GUIDRV_SPAGE_4C1	4bpp	Yes	default
GUIDRV_SPAGE_OY_4C1	4bpp	Yes	Y axis mirrored
GUIDRV_SPAGE_OX_4C1	4bpp	Yes	X axis mirrored
GUIDRV_SPAGE_OXY_4C1	4bpp	Yes	X and Y axis mirrored
GUIDRV_SPAGE_OS_4C1	4bpp	Yes	X and Y swapped
GUIDRV_SPAGE_OSY_4C1	4bpp	Yes	X and Y swapped, Y axis mirrored
GUIDRV_SPAGE_OSX_4C1	4bpp	Yes	X and Y swapped, X axis mirrored
GUIDRV_SPAGE_OSXY_4C1	4bpp	Yes	X and Y swapped, X and Y axis mirrored

Table 33.106: Color depth and display orientation

Important note for mirroring

As far as we know nearly all supported controllers of this driver support hardware mirroring for X- and Y-axis. If one or both of axis need to be mirrored it is highly recommended to use the hardware commands for mirroring within the initialization sequence of the controller, because software mirroring could cause a negative effect on the performance.

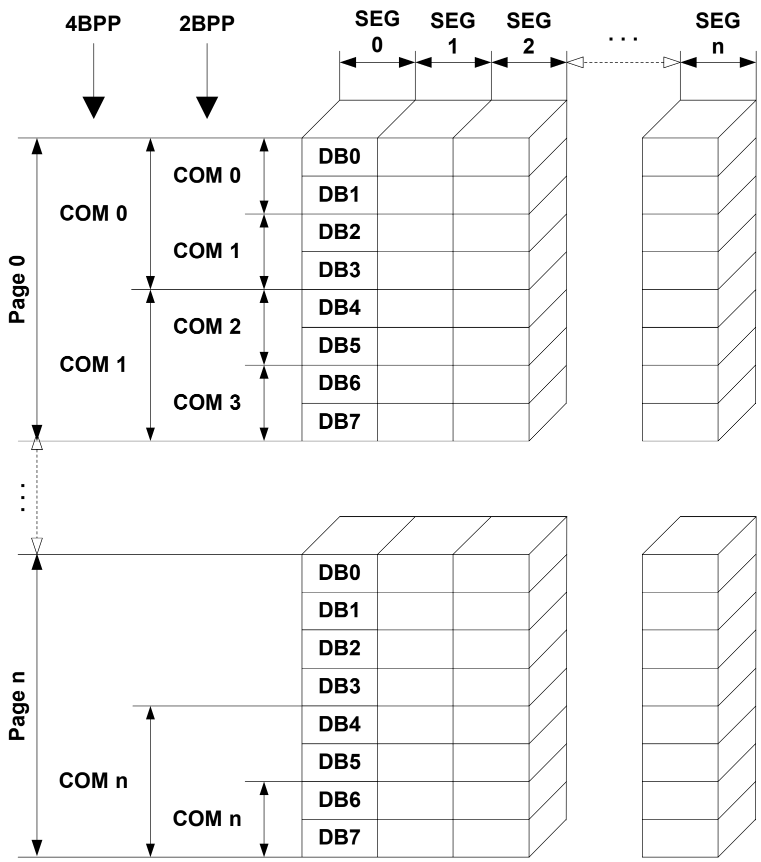
Driver selection

To use GUIDRV_SPage for the given display, the following call may be used in the function LCD_X_Config:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SPAGE_4C0, GUICC_4, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display.

RAM requirements

This display driver can be used with or without a display data cache. The data cache contains a complete copy of the LCD data RAM. If no cache is used, there are no additional RAM requirements.
It is highly recommended to use this driver with a data cache for faster LCD-access. Not using a cache degrades the performance of this driver seriously. The amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) =
(LCD_YSIZE + (8 / LCD_BITSPERPIXEL - 1)) / 8 * LCD_BITSPERPIXEL * LCD_XSIZE
```

Run-time configuration

The table below shows the available run-time configuration routines for this driver:

Routine	Description
GUIDRV_SPage_Config()	Passes a pointer to a CONFIG_SPAGE structure.
GUIDRV_SPage_SetBus8()	Configures the driver to use the 8 bit indirect interface and passes a pointer to a GUI_PORT_API structure to the driver.
GUIDRV_SPage_Set1502()	Set up the driver to use one of the following controllers: - Avant Electronics SBN0064G - Hitachi HD61202 - S6B0108 (KS0108)

Table 33.107: Run-time configuration

Routine	Description
GUIDRV_SPage_Set1510()	Set up the driver to use one of the following controllers: - Epson S1D15605, S1D15606, S1D15607, S1D15608, S1D15705, S1D15710, S1D15714 - Integrated Solutions Technology IST3020 - New Japan Radio Company NJU6676 - Novatek NT7502, NT7534, NT7538, NT75451 - OriseTech SPLC502B - Samsung S6B0713, S6B0719, S6B0724, S6B1713 - Sino Wealth SH1101A - Sitronix ST7522, ST7565, ST7567 - Solomon SSD1303, SSD1305, SSD1805, SSD1815, SSD1821 - Sunplus SPLC501C - UltraChip UC1601, UC1606, UC1608, UC1701
GUIDRV_SPage_Set1512()	Set up the driver to use one of the following controllers: - Epson S1D15E05, S1D15E06, S1D15719, S1D15721
GUIDRV_SPage_SetST75256()	Set up the driver to use the following controller: - Sitronix ST75256
GUIDRV_SPage_SetST7591()	Set up the driver to use the following controller: - Sitronix ST7591
GUIDRV_SPage_SetUC1611()	Set up the driver to use the following controller: - UltraChip UC1611
GUIDRV_SPage_SetUC1638()	Set up the driver to use the following controller: - UltraChip UC1638

Table 33.107: Run-time configuration

GUIDRV_SPage_Config()

Description

Passes a pointer to a `CONFIG_SPAGE` structure to the driver.

Prototype

```
void GUIDRV_SPage_Config(GUI_DEVICE * pDevice, CONFIG_SPAGE * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a `CONFIG_SPAGE` structure described below.

Table 33.108: GUIDRV_SPage_Config() parameter list

Elements of structure CONFIG_SPAGE

Data type	Element	Description
int	FirstSEG	First segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	FirstCOM	First common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.

Table 33.109: CONFIG_SPAGE element list

GUIDRV_SPage_SetBus8()

Description

Tells the driver to use the 8 bit indirect interface and passes a pointer to a `GUI_PORT_API` structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_SPage_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.

Table 33.110: GUIDRV_SPage_SetBus8() parameter list

Required GUI_PORT_API routines

Element	Data type
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfRead8_A1	U8 (*)(void)

Table 33.111: Required GUI_PORT_API routines

GUIDRV_SPage_Set1502()

Description

Configures the driver to use one of the following controllers:
- Avant Electronics SBN0064G
- Hitachi HD61202
- Samsung S6B0108, KS0108

Prototype

```
void GUIDRV_SPage_Set1502(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.112: GUIDRV_SPage_Set1502() parameter list

GUIDRV_SPage_Set1510()

Description

Configures the driver to use one of the following controllers:
- Epson S1D15605, S1D15606, S1D15607, S1D15608, S1D15705, S1D15710, S1D15714
- Integrated Solutions Technology IST3020
- New Japan Radio Company NJU6676
- Novatek NT7502, NT7534, NT7538, NT75451
- Samsung S6B0713, S6B0719, S6B0724, S6B1713
- Sino Wealth SH1101A
- Sitronix ST7522, ST7565, ST7567
- Solomon SSD1303, SSD1305, SSD1306, SSD1805, SSD1815, SSD1821
- Sunplus SPLC501C
- UltraChip UC1601, UC1606, UC1608, UC1701

Prototype

```
void GUIDRV_SPage_Set1510(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.113: GUIDRV_SPage_Set1510() parameter list

GUIDRV_SPage_Set1512()

Description

Configures the driver to use one of the following controllers:

- Epson S1D15E05, S1D15E06, S1D15719, S1D15721

Prototype

```
void GUIDRV_SPage_Set1512(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.114: GUIDRV_SPage_Set1512() parameter list

GUIDRV_SPage_SetST75256()

Description

Configures the driver to use the Sitronix ST75256 controller. This currently supports 2bpp only.

Prototype

```
void GUIDRV_SPage_SetST75256(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.115: GUIDRV_SPage_SetST75256() parameter list

GUIDRV_SPage_SetST7591()

Description

Configures the driver to use the Sitronix ST7591 controller.

Prototype

```
void GUIDRV_SPage_SetST7591(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.116: GUIDRV_SPage_SetST7591() parameter list

GUIDRV_SPage_SetUC1611()

Description

Configures the driver use to the UltraChip UC1611 controller.

Prototype

```
void GUIDRV_SPage_SetUC1611(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.117: GUIDRV_SPage_SetUC1611() parameter list

GUIDRV_SPage_SetUC1638()

Description

Configures the driver use to the UltraChip UC1638 controller.

Prototype

```
void GUIDRV_SPage_SetUC1638(GUI_DEVICE * pDevice);
```

Parameter	Description
pDevice	Pointer to the driver device.

Table 33.118: GUIDRV_SPage_SetUC1638() parameter list

Configuration Example

```
void LCD_X_Config(void) {
  GUI_PORT_API   PortAPI = {0};
  CONFIG_SPAGE   Config  = {0};
  GUI_DEVICE   * pDevice;

  //
  // Set display driver and color conversion for 1st layer
  //
  pDevice = GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Display size configuration
  //
  if (LCD_GetSwapXY()) {
    LCD_SetSizeEx (0, YSIZE_PHYS,   XSIZE_PHYS);
    LCD_SetVSizeEx(0, VYSIZE_PHYS,  VXSIZE_PHYS);
  } else {
    LCD_SetSizeEx (0, XSIZE_PHYS,   YSIZE_PHYS);
    LCD_SetVSizeEx(0, VXSIZE_PHYS,  VYSIZE_PHYS);
  }
  //
  // Driver configuration
  //
  Config.FirstSEG = 0;//256 - 224;
  GUIDRV_SPage_Config(pDevice, &Config);
  //
  // Configure hardware routines
  //
  PortAPI.pfWrite8_A0  = _Write8_A0;
  PortAPI.pfWrite8_A1  = _Write8_A1;
  PortAPI.pfWriteM8_A1 = _WriteM8_A1;
  PortAPI.pfReadM8_A1  = LCD_X_8080_8_ReadM01;
  GUIDRV_SPage_SetBus8(pDevice, &PortAPI);
  //
  // Controller configuration
  //
  GUIDRV_SPage_SetUC1611(pDevice);
}
```


33.7.12 GUIDRV_SSD1926

Supported hardware

Controllers

This driver works with the Solomon SSD1926 display controller.

Bits per pixel

Currently supported color depth is 8. The display controller supports up to 32 bits per pixel. The driver can be extended on demand if support for an other color depth is required.

Interfaces

The driver supports the 16 bit indirect interface.

Color depth and display orientation

This driver can be used with different orientations. The following table shows the configuration macros which can be used to create and link the driver during the initialization:

Identifier	Color depth and orientation
GUIDRV_SSD1926_8	8bpp, default orientation
GUIDRV_SSD1926_OY_8	8bpp, Y axis mirrored
GUIDRV_SSD1926_OX_8	8bpp, X axis mirrored
GUIDRV_SSD1926_OXY_8	8bpp, X and Y axis mirrored
GUIDRV_SSD1926_OS_8	8bpp, X and Y swapped
GUIDRV_SSD1926_OSY_8	8bpp, X and Y swapped, Y axis mirrored
GUIDRV_SSD1926_OSX_8	8bpp, X and Y swapped, X axis mirrored
GUIDRV_SSD1926_OSXY_8	8bpp, X and Y swapped, X and Y axis mirrored

Table 33.119: Color depth and display orientation

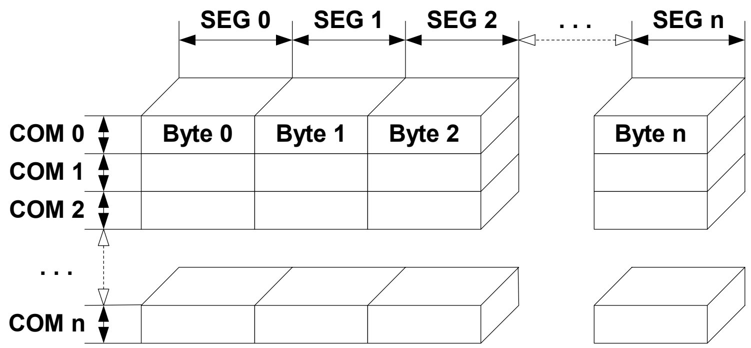
Driver selection

To use GUIDRV_SSD1926 for the given display, the following command can be used e.g.:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SSD1926, GUICC_323, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display.

RAM requirements

This display driver may be used with or without a display data cache, containing a complete copy of the LCD data RAM. If no cache is used, there are no additional RAM requirements.
It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) = LCD_XSIZE * LCD_YSIZE
```

Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_SSD1926_Config()	Passes a pointer to a CONFIG_SSD1926 structure to the driver.
GUIDRV_SSD1926_SetBus16()	Tells the driver to use the 16 bit indirect interface and passes pointer to a GUI_PORT_API structure to the driver.

Table 33.120: Run-time configuration

GUIDRV_SSD1926_Config()

Description

Passes a pointer to a `CONFIG_SSD1926` structure to the driver.

Prototype

```
void GUIDRV_SSD1926_Config(GUI_DEVICE * pDevice, CONFIG_SSD1926 * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a CONFIG_SSD1926 structure described below.

Table 33.121: GUIDRV_SSD1926_Config()

Elements of structure CONFIG_SSD1926

Data type	Element	Description
int	FirstSEG	First segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	FirstCOM	First common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	UseCache	Enables or disables use of a data cache. Should be set to 1 for enabling and to 0 for disabling.

Table 33.122: CONFIG_SSD1926 element list

GUIDRV_SSD1926_SetBus16()

Description

Tells the driver to use the 16 bit indirect interface and passes a pointer to a `GUI_PORT_API` structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_SSD1926_SetBus16(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. See required routines below.

Table 33.123: GUIDRV_SSD1926_SetBus16() parameter list

Required GUI_PORT_API routines

Data type	Element	Description
void (*)(U16 Data)	pfWrite16_A0	Pointer to a function which writes one word to the controller with C/D line low.
void (*)(U16 Data)	pfWrite16_A1	Pointer to a function which writes one word to the controller with C/D line high.
void (*)(U16 * pData, int NumItems)	pfWriteM16_A0	Pointer to a function which writes multiple words to the controller with C/D line low.
void (*)(U16 * pData, int NumItems)	pfWriteM16_A1	Pointer to a function which writes multiple words to the controller with C/D line high.
U16 (*)(void)	pfRead16_A1	Pointer to a function which reads one word from the controller with C/D line high.

Table 33.124: Required GUI_PORT_API routines

Configuration Example

```
#define XSIZE 320L
#define YSIZE 240L

GUI_PORT_API _PortAPI;

void LCD_X_Config(void) {
  GUI_DEVICE * pDevice_0;
  CONFIG_SSD1926 Config_0 = {0};

  //
  // Set display driver and color conversion
  //
  pDevice_0 = GUI_DEVICE_CreateAndLink(GUIDRV_SSD1926_8, GUICC_8666, 0, 0);
  //
  // Common display driver configuration
  //
  LCD_SetSizeEx (0, XSIZE, YSIZE);
  LCD_SetVSizeEx(0, XSIZE, YSIZE);
  //
  // Set driver specific configuration items
  //
  Config_0.UseCache  = 1;
  //
  // Set hardware access routines
  //
  _PortAPI.pfWrite16_A0  = LCD_X_8080_16_Write00_16;
  _PortAPI.pfWrite16_A1  = LCD_X_8080_16_Write01_16;
  _PortAPI.pfWriteM16_A0 = LCD_X_8080_16_WriteM00_16;
  _PortAPI.pfWriteM16_A1 = LCD_X_8080_16_WriteM01_16;
  _PortAPI.pfRead16_A1   = LCD_X_8080_16_Read01_16;
  GUIDRV_SSD1926_SetBus16(pDevice, &_PortAPI);
  //
  // Pass configuration structure to driver
  //
  GUIDRV_SSD1926_Config(pDevice, &Config_0);
}
```


33.7.13 GUIDRV_UC1698G

Supported Hardware

Controllers

This driver has been tested with the UltraChip UC1698G.

Bits per pixel

5 bpp grayscales.

Interfaces

The driver supports the 8- and 16-bit indirect interface.

Color depth and display orientation

The driver consists of several files. They are named `_[O]_[BPP]C[CACHE].c`. The [O] is optional and stands for the desired display orientation. [BPP] means the color depth to use and [CACHE] is defined with 1 to use a cache and 0 to work without cache. The following table shows the driver files and the configuration macros which should be used to create and link the driver during the initialization:

Identifier	Color depth and orientation
GUIDRV_UC1698G_5C0	5bpp, no cache, default orientation.
GUIDRV_UC1698G_OY_5C0	5bpp, no cache, Y axis mirrored.
GUIDRV_UC1698G_OX_5C0	5bpp, no cache, X axis mirrored.
GUIDRV_UC1698G_OXY_5C0	5bpp, no cache, Y and X axis mirrored.
GUIDRV_UC1698G_OS_5C0	5bpp, no cache, X and Y axis swapped.
GUIDRV_UC1698G_OSY_5C0	5bpp, no cache, Y axis mirrored, X and Y axis swapped.
GUIDRV_UC1698G_OSX_5C0	5bpp, no cache, X axis mirrored, X and Y axis swapped.
GUIDRV_UC1698G_OSXY_5C0	5bpp, no cache, X and Y axis mirrored, X and Y axis swapped.
GUIDRV_UC1698G_5C1	5bpp, cache, default orientation.
GUIDRV_UC1698G_OY_5C1	5bpp, cache, Y axis mirrored.
GUIDRV_UC1698G_OX_5C1	5bpp, cache, X axis mirrored.
GUIDRV_UC1698G_OXY_5C1	5bpp, cache, Y and X axis mirrored.
GUIDRV_UC1698G_OS_5C1	5bpp, cache, X and Y axis swapped.
GUIDRV_UC1698G_OSY_5C1	5bpp, cache, Y axis mirrored, X and Y axis swapped.
GUIDRV_UC1698G_OSX_5C1	5bpp, cache, X axis mirrored, X and Y axis swapped.
GUIDRV_UC1698G_OSXY_5C1	5bpp, cache, X and Y axis mirrored, X and Y axis swapped.

Table 33.125: Color depth and display orientation

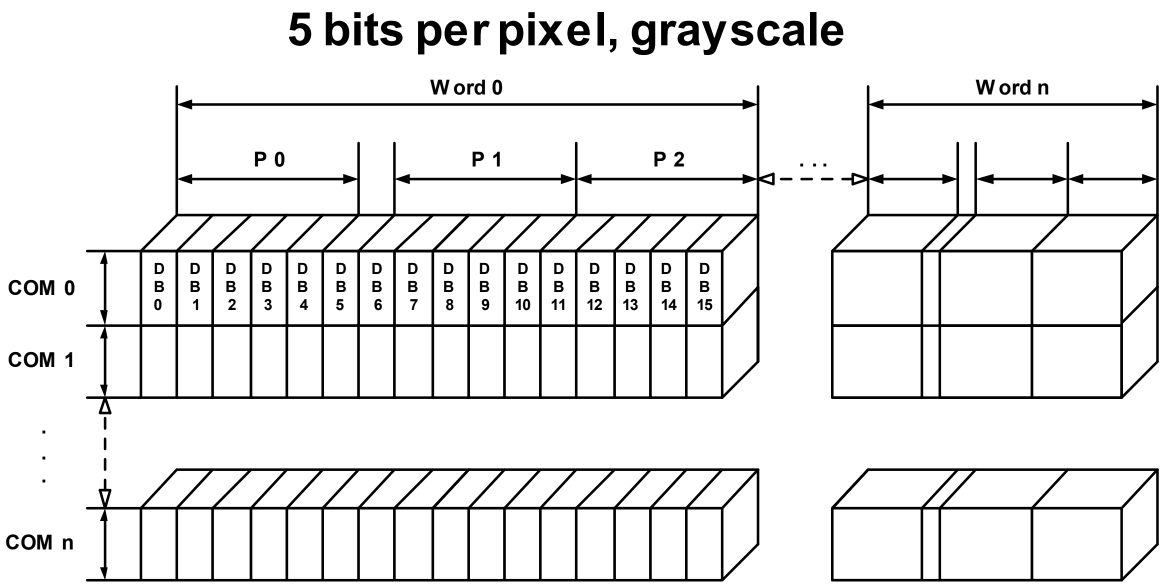
Driver selection

To use for the given display, the following command can be used e.g.:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_UC1698G_5C1, GUICC_5, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization



The picture above shows the relation between the display memory and the pixels of the LCD in terms of the color depth

RAM requirements

This display driver requires app. 500 Bytes to work. It can also be used with and without a display data cache, containing a complete copy of the content of the display data RAM. The amount of memory used by the cache is:

```
(LCD_XSIZE + 2) / 3 * LCD_YSIZE * 2
```

Using a cache avoids reading operations from the display controller in case of XOR drawing operations and further it speeds up string output operations.

Run-time configuration

The table below shows the available run-time configuration routines of this driver:

Routine	Description
GUIDRV_UC1698G_Config()	Passes a pointer to a CONFIG_UC1698G structure to the driver.
GUIDRV_UC1698G_SetBus8()	Tells the driver to use the 8 bit indirect interface and passes pointer to a GUI_PORT_API structure to the driver.
GUIDRV_UC1698G_SetBus16()	Tells the driver to use the 16 bit indirect interface and passes pointer to a GUI_PORT_API structure to the driver.

Table 33.126: Run-time configuration

GUIDRV_UC1698G_Config()

Description

Configures the driver to work according to the passed `CONFIG_UC1698G` structure.

Prototype

```
void GUIDRV_UC1698G_Config(GUI_DEVICE * pDevice, CONFIG_UC1698G * pConfig);
```

Parameter	Description
pDevice	Pointer to the driver device.
pConfig	Pointer to a CONFIG_UC1698G structure described below.

Table 33.127: GUIDRV_UC1698G_Config() parameter list

Elements of structure CONFIG_UC1698G

Data type	Element	Description
int	FirstSEG	First segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	FirstCOM	First common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation. The value is normally 0.
int	NumDummyReads	Number of dummy reads to do before the actual read operation.

Table 33.128: CONFIG_UC1698G element list

GUIDRV_UC1698G_SetBus8()

Description

Tells the driver to use the 8 bit indirect interface and passes a pointer to a GUI_PORT_API structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_UC1698G_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. The required routines are listed below.

Table 33.129: GUIDRV_UC1698G_SetBus8() parameter list

GUIDRV_UC1698G_SetBus16()

Description

Tells the driver to use the 16 bit indirect interface and passes a pointer to a GUI_PORT_API structure to the driver containing function pointers to the hardware routines to be used.

Prototype

```
void GUIDRV_UC1698G_SetBus16(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

Parameter	Description
pDevice	Pointer to the driver device.
pHW_API	Pointer to a GUI_PORT_API structure. The required routines are listed below.

Table 33.130: GUIDRV_UC1698G_SetBus16() parameter list

Required GUI_PORT_API routines

The required GUI_PORT_API routines depend on the used interface. If a cache is used the routines for reading data are unnecessary for each interface:

8 bit interface

Data type	Element	Description
void (*)(U8 Data);	pfWrite8_A0	Pointer to a function which writes one byte to the controller with C/D line low.
void (*)(U8 Data);	pfWrite8_A1	Pointer to a function which writes one byte to the controller with C/D line high.
void (*)(U8 * pData, int NumItems);	pfWriteM8_A0	Pointer to a function which writes multiple bytes to the controller with C/D line low.
void (*)(U8 * pData, int NumItems);	pfWriteM8_A1	Pointer to a function which writes multiple bytes to the controller with C/D line high.
U8 (*)(void);	pfRead8_A0	Pointer to a function which reads one byte from the controller with C/D line low.
U8 (*)(void);	pfRead8_A1	Pointer to a function which reads one byte from the controller with C/D line high.
void (*)(U8 * pData, int NumItems);	pfReadM8_A1	Pointer to a function which reads multiple bytes from the controller with C/D line high.

Table 33.131: GUI_PORT_API - 8 bit interface

16 bit interface

Data type	Element	Description
void (*)(U16 Data);	pfWrite16_A0	Pointer to a function which writes one word to the controller with C/D line low.
void (*)(U16 Data);	pfWrite16_A1	Pointer to a function which writes one word to the controller with C/D line high.
void (*)(U16 * pData, int NumItems);	pfWriteM16_A0	Pointer to a function which writes multiple words to the controller with C/D line low.
void (*)(U16 * pData, int NumItems);	pfWriteM16_A1	Pointer to a function which writes multiple words to the controller with C/D line high.
U16 (*)(void);	pfRead16_A0	Pointer to a function which reads one word from the controller with C/D line low.
U16 (*)(void);	pfRead16_A1	Pointer to a function which reads one word from the controller with C/D line high.
void (*)(U16 * pData, int NumItems);	pfReadM16_A1	Pointer to a function which reads multiple words from the controller with C/D line high.

Table 33.132: GUI_PORT_API - 16 bit interface

33.7.14 GUIDRV_CompactColor_16

Supported Hardware

Controllers

This driver works with the following display controllers:
- Ampire FSA506
- Epson S1D13742, S1D13743, S1D19122
- FocalTech FT1509
- Himax HX8301, HX8312A, HX8325A, HX8340, HX8347, HX8352, HX8352B, HX8353
- Hitachi HD66766, HD66772, HD66789
- Ilitek ILI9161, ILI9220, ILI9221, ILI9320, ILI9325, ILI9326, ILI9328, ILI9342, ILI9481
- LG Electronics LGDP4531, LGDP4551
- MagnaChip D54E4PA7551
- Novatek NT39122, NT7573
- OriseTech SPFD5408, SPFD54124C, SPFD5414D, SPFD5420A
- Renesas R61505, R61509, R61516, R61526, R61580, R63401
- Samsung S6D0110A, S6D0117, S6D0128, S6D0129, S6D04H0
- Sharp LCY-A06003, LR38825
- Sitronix ST7628, ST7637, ST7687, ST7712, ST7715, ST7735, ST7787, ST7789
- Solomon SSD1284, SSD1289, SSD1298, SSD1355, SSD1961, SSD1963, SSD2119
- Toshiba JBT6K71

Bits per pixel

Supported color depth is 16 bpp.

Interfaces

The driver supports the indirect interface (8- and 16-bit) and the 3 pin SPI interface. Default mode is 8-bit indirect.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

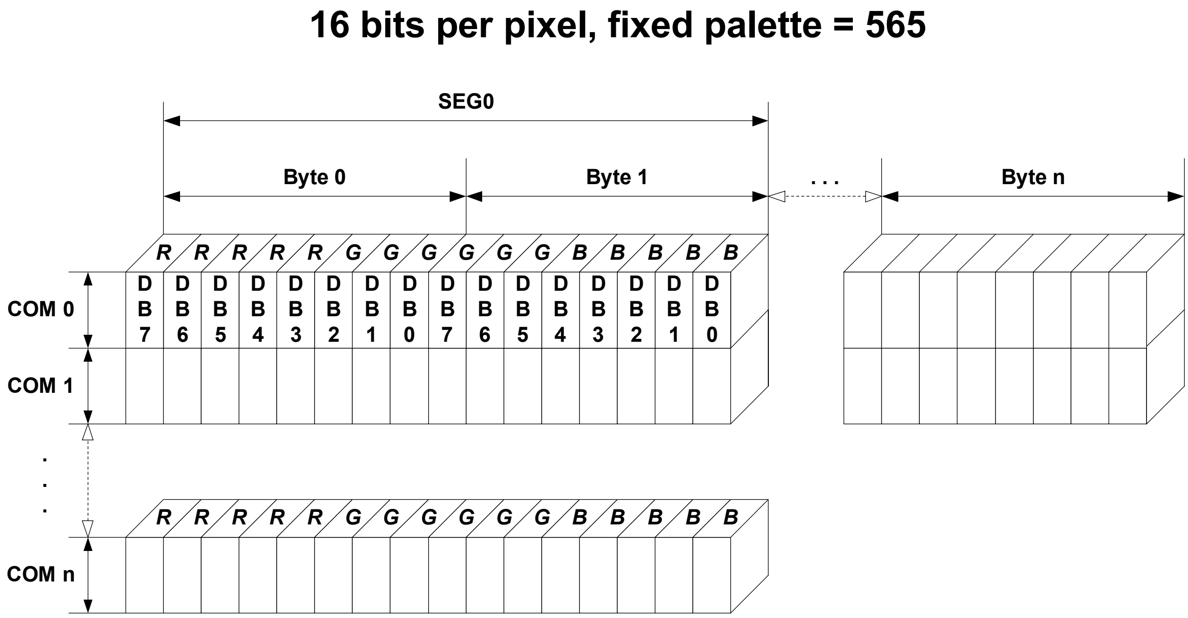
```
#define LCD_USE_COMPACT_COLOR_16
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_CompactColor_16.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_CompactColor_16 for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_COMPACT_COLOR_16,
                                   GUICC_565, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display.

RAM requirements

This display driver can be used with and without a display data cache, containing a complete copy of the contents of the display data RAM. The amount of memory used by the cache is: `LCD_XSIZE * LCD_YSIZE * 2 bytes`. Using a cache is only recommended if it is intended to use a lot of drawing operations using the XOR drawing mode. A cache would avoid reading the display data in this case. Normally the use of a cache is not recommended.

The driver uses a write buffer for drawing multiple pixels of the same color. If multiple pixels of the same color should be drawn, the driver first fills the buffer and then performs a single call of the LCD_WRITEM_A1 macro to transfer the data to the display controller at once. The default buffer size is 500 bytes.

Compile-time configuration

Controller selection

To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file LCDConf_CompactColor_16.h. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
66700	Sharp LR38825
66701	Ilitek ILI9326 OriseTech SPFD5420A Renesas R61509, R63401
66702	Solomon SSD1284, SSD1289, SSD1298
66703	Toshiba JBT6K71
66704	Sharp LCY-A06003
66705	Samsung S6D0129
66706	MagnaChip D54E4PA7551
66707	Himax HX8312

Table 33.133: Controller selection

Number	Supported Controller
66708	FocalTech FT1509 Ilitek ILI9320, ILI9325, ILI9328 LG Electronics LGDP4531, LGDP4551 OriseTech SPFD5408 Renesas R61505, R61580
66709	Epson S1D19122 Himax HX8353 Ilitek ILI9342, ILI9481 Novatek NT39122 Orisetech SPFD54124C, SPFD5414D Renesas R61516, R61526 Samsung S6D04H0 Sitronix ST7628, ST7637, ST7687, ST7715, ST7735 Solomon SSD1355, SSD1961, SSD1963
66710	Novatek NT7573
66711	Epson S1D13742, S1D13743
66712	Himax HX8347, HX8352
66713	Himax HX8340
66714	Solomon SSD2119
66715	Himax HX8352B
66716	Ampire FSA506
66717	Sitronix ST7787, ST7789
66766	Hitachi HD66766 Ilitec ILI9161 Samsung S6D0110A
66772	Himax HX8301 Hitachi HD66772 Ilitec ILI9220, ILI9221 Samsung S6D0117, S6D0128 Sitronix ST7712
66789	Hitachi HD66789

Table 33.133: Controller selection

Display configuration

The following table shows the available configuration macros:

Macro	Description
LCD_MIRROR_X	Activate to mirror X-axis.
LCD_MIRROR_Y	Activate to mirror Y-axis.
LCD_SWAP_XY	Activate to swap X- and Y-axis.

Table 33.134: Display configuration

For details, refer to “Display orientation” on page 1070.

Hardware access

The following table shows the available configuration macros which can be defined in this file for configuring the hardware access:

Macro	Description
LCD_NUM_DUMMY_READS	Number of required dummy reads if a read operation should be executed. The default value is 2. If using a serial interface the display controllers HD66766 and HD66772 need 5 dummy reads. Sharp LR38825 needs 3 dummy reads with a 8-bit bus.
LCD_REG01	This macro is only required if a Himax HX8312A is used. Unfortunately the register 0x01 (Control register 1) contains orientation specific settings as well as common settings. So this macro should contain the contents of this register.
LCD_SERIAL_ID	With a serial 3 wire interface this macro defines the ID signal of the device ID code. It should be 0 (default) or 1. Please note: This macro is only used with the 3 wire protocol for Hitachi HD66772, Samsung S6D0117, Himax HX8301 and Ilitek ILI9220.
LCD_USE_SERIAL_3PIN	This configuration macro has been implemented to support the 3 wire serial interface of the following controllers: Hitachi HD66772, Samsung S6D0117, Himax HX8301, Ilitek ILI9220. Should be set to 1 if the 3 wire serial interface is used. Default is 0. Please note: Do not use this macro with other display controllers!
LCD_USE_PARALLEL_16	Should be set to 1 if the 16 bit parallel interface is used. Default is 0.
LCD_WRITE_BUFFER_SIZE	Defines the size of the write buffer. Using a write buffer increases the performance of the driver. If multiple pixels should be written with the same color, the driver first fills the buffer and then writes the content of the buffer using LCD_WRITEM_A1 instead of multiple calls of LCD_WRITE_A1. The default buffer size is 500 bytes.
LCD_WRITE_A0	Write a byte to display controller with RS-line low.
LCD_WRITE_A1	Write a byte to display controller with RS-line high.
LCD_READM_A1	Read multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) from display controller with RS-line high.
LCD_WRITEM_A1	Write multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) to display controller with RS-line high.
LCD_WRITEM_A0	Write multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) to display controller with RS-line low.

Table 33.135: Hardware access

The 'Driver Output Mode' and 'Entry Mode' registers are initialized automatically.

Run-time configuration

The following table lists the available run-time configuration routines:

Routine	Description
LCD_SetSizeEx()	Changes the size of the visible area.

Table 33.136: Run-time configuration routines

Configuration example

The following shows how to select the driver and how it can be configured:

LCDConf.h

As explained above it should include the following for selecting the driver:

```
#define LCD_USE_COMPACT_COLOR_16
```

LCDConf_CompactColor_16.h

This file contains the display driver specific configuration and could look as the following:

```
//
// General configuration of LCD
//
#define LCD_CONTROLLER      66709 // Renesas R61516
#define LCD_BITSPERPIXEL       16
#define LCD_USE_PARALLEL_16     1
#define LCD_MIRROR_Y            1
//
// Indirect interface configuration
//
void LCD_X_Write01_16(unsigned short c);
void LCD_X_Write00_16(unsigned short c);
void LCD_X_WriteM01_16(unsigned short * pData, int NumWords);
void LCD_X_WriteM00_16(unsigned short * pData, int NumWords);
void LCD_X_ReadM01_16 (unsigned short * pData, int NumWords);
#define LCD_WRITE_A1(Word) LCD_X_Write01_16(Word)
#define LCD_WRITE_A0(Word) LCD_X_Write00_16(Word)
#define LCD_WRITEM_A1(Word, NumWords) LCD_X_WriteM01_16(Word, NumWords)
#define LCD_WRITEM_A0(Word, NumWords) LCD_X_WriteM00_16(Word, NumWords)
#define LCD_READM_A1(Word, NumWords)  LCD_X_ReadM01_16(Word, NumWords)
```

LCDConf.c

The following shows how to create a display driver device with this driver and how to configure it:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_COMPACT_COLOR_16, // Display driver
                           GUICC_M565,              // Color conversion
                           0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx(0, 240, 320);                      // Physical display size in pixels
}
```


33.7.15 GUIDRV_Fujitsu_16

This driver supports the Fujitsu Graphic display controllers. It has been tested with "Jasmine", but it should also work with "Lavender", since all relevant registers are compatible.

Supported hardware

Controllers

This driver works with the following display controllers:

- Fujitsu Jasmine
- Fujitsu Lavender

Bits per pixel

Supported color depths are 1, 2, 4, 8 and 16 bpp.

Interfaces

The driver has been tested with a 32 bit interface to the CPU. If a 16 bit interface is used, the 32-bit accesses can be replaced by 2 16-bit accesses.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_FUJITSU_16
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_Fujitsu_16.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_Fujitsu_16 for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_FUJITSU_16, GUICC_556, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Available configuration macros (compile time configuration)

Controller selection

To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file `LCDConf_Fujitsu_16.h`. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
8720	Fujitsu Jasmine
8721	Fujitsu Lavender

Table 33.137: Controller selection

Display data RAM organization

The display controller uses DRAM in an optimized, non-linear way (described in the Fujitsu documentation). Direct memory access is not used by the driver.

RAM requirements

About 16 bytes for some static variables.

Hardware configuration

This driver requires a direct interface for hardware access as described in the chapter "Configuration" on page 1203. The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_READ_REG	Read a register of the display controller. (as 32 bit value) (optional)
LCD_WRITE_REG	Write a register of the display controller. (as 32 bit value) (optional)

Table 33.138: Hardware configuration

The driver contains a default for hardware access macros, which configures 32 bit access on the Fujitsu demonstration platform (Using an MB91361 or MB91362 and a Jasmine chip at address 0x30000000); if the target hardware is compatible with these settings, then LCD_READ_REG(), LCD_WRITE_REG() do not need to be defined.

Color format (R/B swap)

It seems that on some target systems, Red and blue are swapped. This can be changed via software if the Config switch LCD_SWAP_RB is toggled in the configuration file.

Hardware initialization

The display controller requires a complicated initialization. Example code is available from Fujitsu in the GDC module. This code is not part of the driver, since it depends on the actual chip used, on the clock settings, the display and a lot of other things. We recommend using the original Fujitsu code, since the documentation of the chips is not sufficient to write this code. Before calling GUI_Init(), the GDC should be initialized using this code (typically called as GDC_Init(0xff)).

Example:

LCDConf.h for VGA display, 8bpp, Jasmine:

```
#define LCD_XSIZE          640 // X-resolution of LCD, Logical color
#define LCD_YSIZE          480 // Y-resolution of LCD, Logical color
#define LCD_BITSPERPIXEL     8
#define LCD_CONTROLLER    8720 // Jasmine
```

Additional configuration switches

The following table shows optional configuration macros available for this driver:

Macro	Description
LCD_ON	Function replacement macro which switches the display on.
LCD_OFF	Function replacement macro which switches the display off.

Table 33.139: Additional configuration switches

33.7.16 GUIDRV_Page1bpp

Supported hardware

Controllers

This driver works with the following display controllers:

- Epson S1D10605, S1D15605, S1D15705, S1D15710, S1D15714, S1D15721, S1D15E05, S1D15E06, SED1520, SED1560, SED1565, SED1566, SED1567, SED1568, SED1569, SED1575
- Hitachi HD61202
- Integrated Solutions Technology IST3020
- New Japan Radio Company NJU6676, NJU6679
- Novatek NT7502, NT7534, NT7538, NT75451
- Philips PCF8810, PCF8811, PCF8535, PCD8544
- Samsung KS0108B, KS0713, KS0724, S6B0108B, S6B0713, S6B0719, S6B0724, S6B1713
- Sino Wealth SH1101A
- Sitronix ST7522, ST7565, ST7567
- Solomon SSD1303, SSD1805, SSD1815, SSD1821
- ST Microelectronics ST7548, STE2001, STE2002
- Sunplus SPLC501C
- UltraChip UC1601, UC1606, UC1608, UC1701

It should be assumed that it will also work with every similar organized controller.

Bits per pixel

Supported color depth is 1bpp.

Interfaces

The driver supports the indirect interface (8 bit) of the display controller. Parallel, 4-pin SPI or I2C bus can be used.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_PAGE1BPP
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_Page1bpp.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_Page1bpp for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_PAGE1BPP, GUICC_1, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Compile-time configuration

Controller selection

To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file `LCDConf_Page1bpp.h`. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
1501	Samsung KS0713, KS0724, S6B0713, S6B0724 UltraChip UC1601, UC1606
1502	Samsung KS0108B S6B0108B
1503	Hitachi HD61202
1504	Philips PCF8810, PCF8811
1505	Philips PCF8535
1506	New Japan Radio Company NJU6679
1507	Philips PCD8544
1508	Epson S1D15710
1509	Solomon SSD1303 OLED controller
1510	Epson S1D15714 Integrated Solutions Technology IST3020 New Japan Radio Company NJU6676 Novatek NT7538, NT75451 Samsung S6B0719 Sino Wealth SH1101A Sitronix ST7522, ST7565, ST7567 Solomon SSD1805, SSD1821 UltraChip UC1608, UC1701
1511	Epson S1D15721
1512	Epson S1D15E05, S1D15E06
1513	ST Microelectronics ST7548, STE2001, STE2002
1520	Epson SED1520
1560	Epson SED1560
1565	Epson SED1565, S1D10605, S1D15605 Novatek NT7502, NT7534 Samsung S6B1713 Solomon SSD1815 Sunplus SPLC501C
1566	Epson SED1566
1567	Epson SED1567
1568	Epson SED1568
1569	Epson SED1569
1575	Epson SED1575, S1D15705

Table 33.140: Controller selection

RAM requirements

This display driver can be used with or without a display data cache in the most cases. If one display contains more than 1 display controller you can not disable the cache. The data cache contains a complete copy of the contents of the display data RAM. If a cache is not used, there are no additional RAM requirements.
It is recommended to use this driver with a data cache for faster display-access. The amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) = (LCD_YSIZE + 7) / 8 * LCD_XSIZE
```

Additional driver functions

LCD_ControlCache

The detailed description of this function can be found on page 1168.

Hardware configuration

This driver accesses the hardware via indirect interface as described in the chapter “Configuration” on page 1203. The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_READ_A0	Read a byte from the display controller with A-line low.
LCD_READ_A1	Read a byte from the display controller with A-line high.
LCD_WRITE_A0	Write a byte to the display controller with A-line low.
LCD_WRITE_A1	Write a byte to the display controller with A-line high.
LCD_WRITEM_A1	Write multiple bytes to the display controller with A-line high.

Table 33.141: Hardware configuration

Display orientation

Some of the supported display controllers supports hardware mirroring of x/y axis. It is recommended to use these functions instead of the display orientation macros of emWin.
If mirroring of the X axis is needed, the command 0xA1 (ADC select reverse) should be used in the initialization macro. This causes the display controller to reverse the assignment of column address to segment output. If the display size in X is smaller than the number of segment outputs of the display controller, the macro LCD_FIRSTSEG0 can be used to add an offset to the column address to make sure, the right RAM address of the display controller is accessed.
If mirroring of the Y axis is needed the command 0xC8 (SHL select revers) should be used in the initialization macro and the macro LCD_FIRSTCOM0 should be used to define the offset needed to access the right RAM address of the display controller.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Description
LCD_CACHE	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).
LCD_FIRSTCOM0	This macro can be used to define the first common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display doc.
LCD_FIRSTSEG0	This macro can be used to define the first segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display doc.
LCD_SUPPORT_CACHECONTROL	When set to 1, LCD_ControlCache() can be used.

Table 33.142: Additional configuration switches

33.7.17 GUIDRV_07X1

Supported hardware

Controllers

This driver works with the following display controllers:
- Novatek NT7506, NT7508
- Samsung KS0711, KS0741, S6B0711, S6B0741
- Sitronix ST7541, ST7571
- Solomon SSD1854
- ST Microelectronics STE2010
- Tomato TL0350A

Bits per pixel

Supported color depth is 2 bpp.

Interface

The controller supports either the 8-bit parallel interface as well as the 4-pin or 3-pin serial peripheral interface (SPI). The current version of the driver supports the 8-bit parallel or 4-pin SPI interface. 3 pin SPI is currently not supported.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_07X1
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_07X1.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the `GUIDRV_07X1` for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_07X1, GUICC_2, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

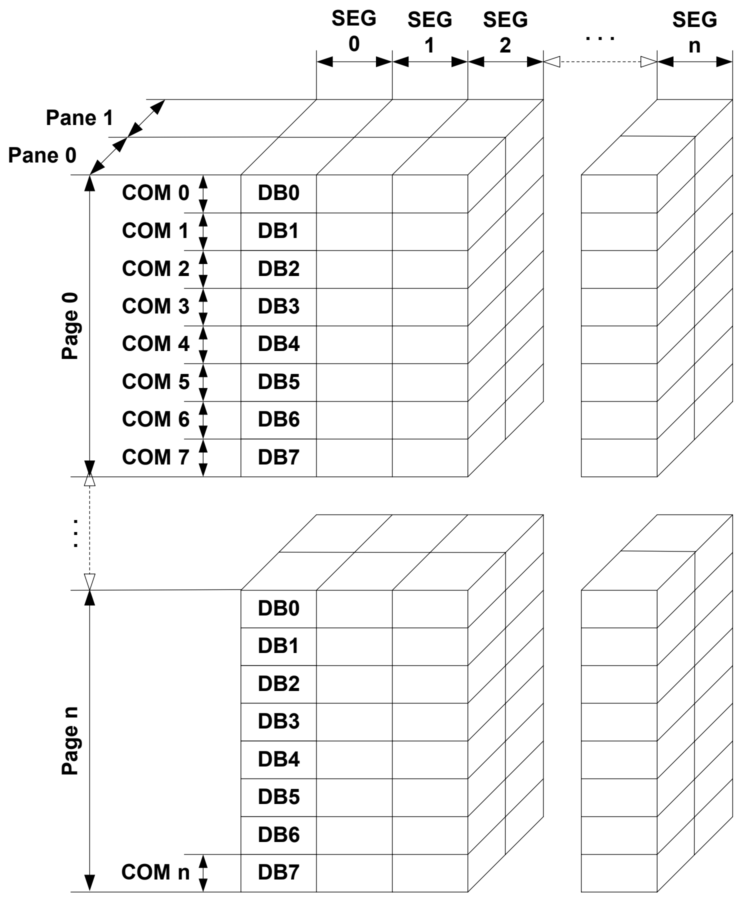
Controller selection

To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file `LCDConf_07X1.h`. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
701	Novatek NT7506 Solomon SSD1854
702	ST Microelectronics STE2010
711	Samsung KS0711, S6B0711
741	Novatek NT7508 Samsung KS0741, S6B0741 Sitronix ST7541, ST7571 Tomato TL0350A

Table 33.143: Controller selection of GUIDRV_07X1

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display. The display memory is divided into two panes for each pixel. The lower bit of each pixel is stored in pane 0 and the higher bit is stored in pane 1.

RAM requirements

This display driver may be used with or without a display data cache, containing a complete copy of the contents of the display data RAM. If a cache is not used, there are no
additional RAM requirements.
It is recommended to use this driver with a data cache for faster display-access. The amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) = (LCD_YSIZE + 7) / 8 * LCD_XSIZE * 2
```

Additional driver functions

LCD_ControlCache

The detailed function description can be found on page 1168.

Hardware configuration

This driver accesses the hardware using the indirect interface as described in the chapter “Configuration” on page 1203. The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_READ_A0	Read a byte from display controller with A-line low. (Used only if working without cache)
LCD_READ_A1	Read a byte from display controller with A-line high. (Used only if working without cache)
LCD_WRITE_A0	Write a byte to display controller with A-line low.
LCD_WRITE_A1	Write a byte to display controller with A-line high.
LCD_WRITEM_A1	Write multiple bytes to display controller with A-line high.

Table 33.144: Hardware configuration

Display orientation

The supported display controllers supports hardware mirroring of x/y axis. It is recommended to use these functions instead of the display orientation macros of emWin. If mirroring of the X axis is needed, the command 0xA1 (ADC select reverse) should be used in the initialization macro. This causes the display controller to reverse the assignment of column address to segment output. If the display size in X is smaller than the number of segment outputs of the display controller, the macro LCD_FIRSTSEG0 can be used to add an offset to the column address to make sure, the right RAM address of the LCD controller is accessed.
If mirroring of the Y axis is needed the command 0xC8 (SHL select revers) should be used in the initialization macro and the macro LCD_FIRSTCOM0 should be used to define the offset needed to access the right RAM address of the display controller.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Description
LCD_FIRSTCOM0	This macro can be used to define the first common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation.
LCD_FIRSTSEG0	This macro can be used to define the first segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation.

Table 33.145: Additional configuration switches

33.7.18 GUIDRV_1611

Supported hardware

Controllers

This driver works with the following display controllers:
- Epson S1D15E05, S1D15E06, S1D15719
- UltraChip UC1610, UC1611, UC1611s

Bits per pixel

Supported color depth is 2bpp (UC1610, S1D15E05, S1D15E06, S1D15719) and 4bpp (UC1611).

Interfaces

The driver supports the indirect interface (8 bit) of the display controller. Parallel, 4-pin SPI or I2C bus can be used.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_1611
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_1611.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_1611 for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_1611, GUICC_2, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

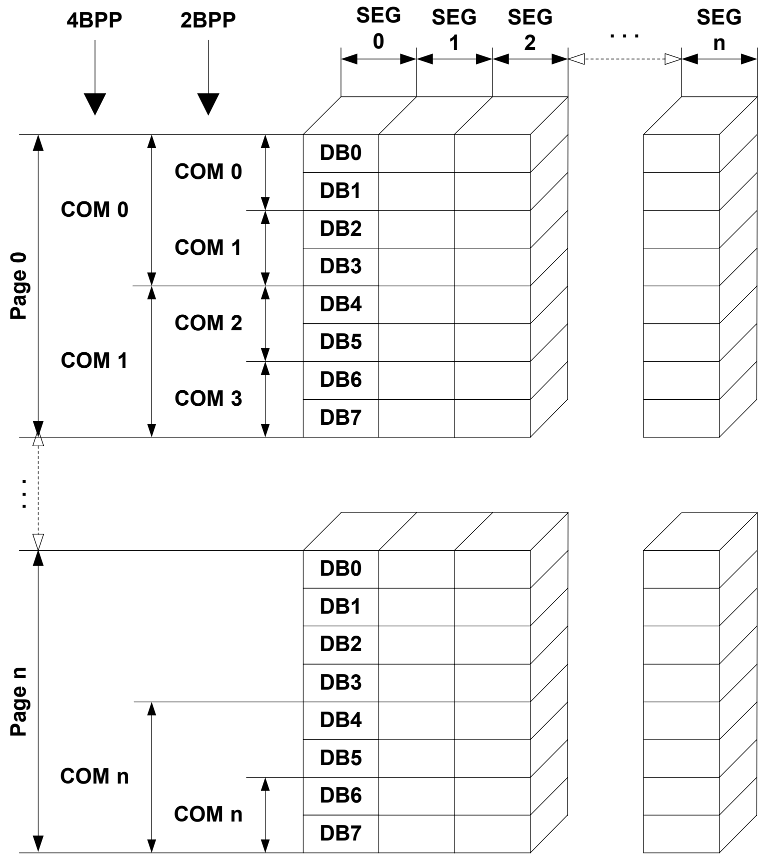
Controller selection

To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file `LCDConf_1611.h`. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
1701	Epson S1D15E05
1702	Epson S1D15719
1800	UltraChip UC1611
1801	UltraChip UC1610
1802	UltraChip UC1611s

Table 33.146: Controller selection of GUIDRV_1611

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

This display driver can be used with or without a display data cache. The data cache contains a complete copy of the LCD data RAM. If no cache is used, there are no additional RAM requirements.
It is highly recommended to use this driver with a data cache for faster LCD-access. Not using a cache degrades the performance of this driver seriously. The amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) =
(LCD_YSIZE + (8 / LCD_BITSPERPIXEL - 1)) / 8 * LCD_BITSPERPIXEL * LCD_XSIZE
```

Hardware configuration

This driver accesses the hardware with the indirect interface. The following table lists the macros which need to be defined for hardware access:

Macro	Description
LCD_READ_A0	Read a byte from LCD controller with A-line low.
LCD_READ_A1	Read a byte from LCD controller with A-line high.
LCD_WRITE_A0	Write a byte to LCD controller with A-line low.
LCD_WRITE_A1	Write a byte to LCD controller with A-line high.
LCD_WRITEM_A1	Write multiple bytes to LCD controller with A-line high.

Table 33.147: Hardware access macros

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Description
LCD_CACHE	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

Table 33.148: Additional configuration switches

33.7.19 GUIDRV_6331

Supported hardware

Controllers

This driver works with the following display controllers:

- Samsung S6B33B0X, S6B33B1X, S6B33B2X

Bits per pixel

Supported color depth is 16 bpp.

Interfaces

The driver supports the indirect interface (8 bit) of the display controller. Parallel or 4-pin SPI bus can be used.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file LCDConf.h:

```
#define LCD_USE_6331
```

After this define has been added the display driver assumes the driver specific configuration file LCDConf_6331.h in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_6331 for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_6331, GUICC_565, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

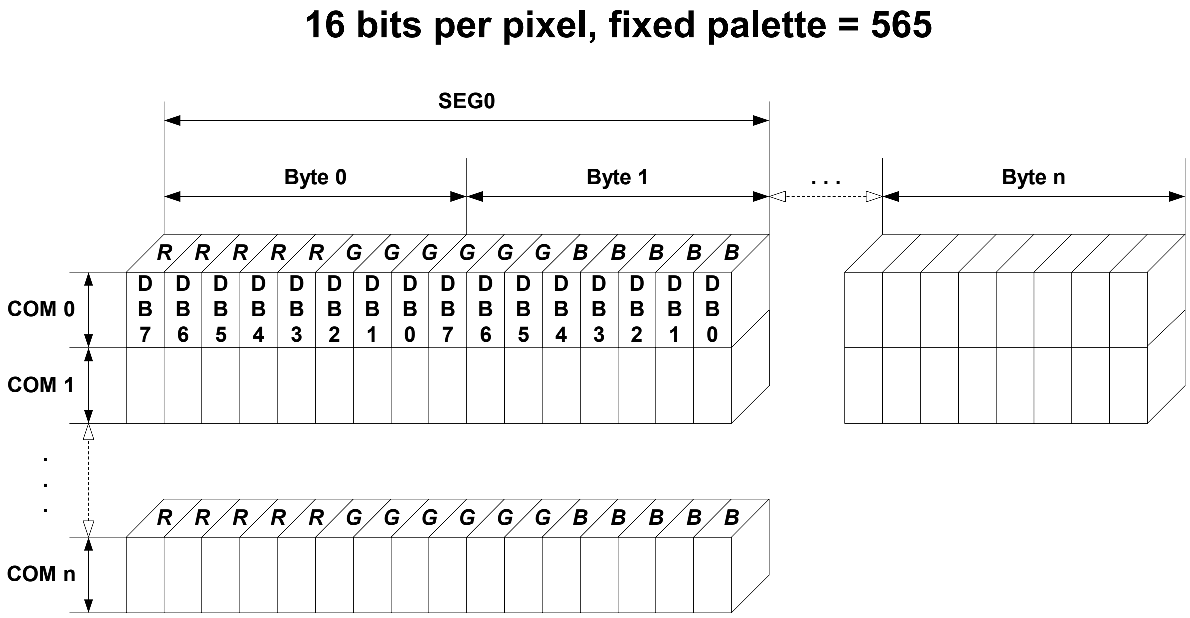
Controller selection

To select the desired controller the macro LCD_CONTROLLER should be used in the configuration file LCDConf_6331.h. The table below shows the values to be used to select the appropriate controller:

Number	Supported Controller
6331	Samsung S6B33B0X, S6B33B1X, S6B33B2X

Table 33.149: Controller selection of GUIDRV_6331

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

This display driver can be used with or without a display data cache, containing a complete copy of the LCD data RAM. The amount of memory used by the cache is:

`LCD_XSIZE x LCD_YSIZE x 2 bytes.`

Hardware configuration

This driver accesses the hardware with the indirect interface. The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_WRITE_A0	Write a byte to display controller with A-line low.
LCD_WRITE_A1	Write a byte to display controller with A-line high.
LCD_WRITEM_A1	Write multiple bytes to display controller with A-line high.
LCD_DRIVER_OUTPUT_MODE_DLN	'Display Line Number' (DLN) selection bits of the 'Driver Output Mode Set' instruction. Details can be found in the display controller documentation.
LCD_DRIVER_ENTRY_MODE_16B	Data bus width selection bit of the 'Entry Mode Set' instruction. Details can be found in the display controller documentation.

Table 33.150: Hardware access macros

The 'Driver Output Mode' and 'Entry Mode' are initializes automatically.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Description
LCD_CACHE	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

Table 33.151: Additional configuration switches

Special requirements

The driver needs to work with the fixed palette mode 565. The driver does not work with other palettes or fixed palette modes. Further the driver needs to swap the red and the blue part of the color index. You should use the following macro definitions in the configuration file LCDConf.h:

```
#define LCD_FIXEDPALETTE 565
#define LCD_SWAP_RB        1
```


33.7.20 GUIDRV_7528

Supported hardware

Controllers

This driver works with the Sitronix ST7528 display controller.

Bits per pixel

Supported color depth is 4 bpp.

Interfaces

The driver supports the 8 bit parallel (simple bus) 4-pin SPI interface.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_7528
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_7528.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_7528 for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_7528, GUICC_4, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

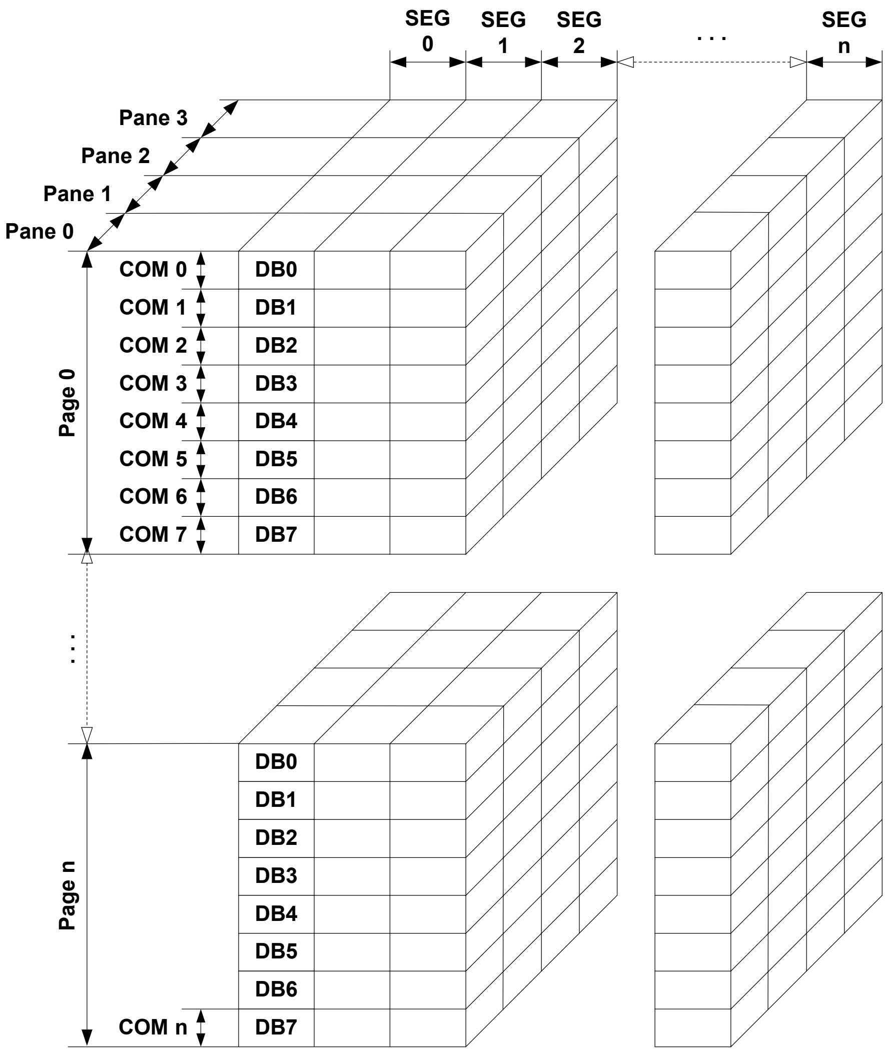
Controller selection

To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file `LCDConf_7528.h`. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
7528	Sitronix ST7528

Table 33.152: Controller selection of GUIDRV_7528

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD. The display memory is divided into four panes for each pixel. The least significant bit (LSB) of each pixel is stored in pane 0 and the MSB is stored in pane 3.

RAM requirements

This LCD driver may be used with or without a display data cache. If the cache is used it holds a complete copy of the contents of the LCD data RAM. If cache is not used, there are no additional RAM requirements.
It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

```
Size of RAM (in bytes) = (LCD_YSIZE + 7) / 8 * LCD_XSIZE * 4
```

A cache is required in SPI mode, because SPI does not allow reading of display contents.

Hardware configuration

This driver accesses the hardware with the indirect interface. The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_WRITE_A0	Write a byte to LCD controller with A-line low.
LCD_WRITE_A1	Write a byte to LCD controller with A-line high.
LCD_WRITEM_A1	Write multiple bytes to display controller with A-line high.
LCD_READ_A1	Read a single byte from display controller with A-line high. Required only if no display data cache is configured.
LCD_READM_A1	Read multiple bytes from display controller with A-line high. Required only if no display data cache is configured.

Table 33.153: Hardware access macros

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Description
LCD_FIRSTCOM0	This macro can be used to define the first common address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation.
LCD_FIRSTSEG0	This macro can be used to define the first segment address to be used in the data RAM of the display controller. The value can be determined experimentally or taken from the display documentation.
LCD_NUM_COM0	A Sitronix ST7528 controller can operate in 2 modes. Mode 0 with 132 segment and 128 common outputs and mode 1 with 160 segment and 100 common outputs. which mode is used depends on hardware, the mode can not be changed via command. Defines the number of available common outputs of the display controller. Possible values for Sitronix ST7528 are: 128 (default, mode 0) 100 (mode 1)
LCD_NUM_SEG0	Defines the number of available segment outputs of the display controller. Possible values for Sitronix ST7528 are: 132 (default, mode 0) 160 (mode 1)
LCD_CACHE	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

Table 33.154: Additional configuration switches

33.7.21 GUIDRV_7529

Supported hardware

Controllers

This driver works with the Sitronix ST7529 display controller.

Bits per pixel

Supported color depths are 5 bpp (default), 4 bpp and 1bpp.

Interfaces

The driver supports the indirect interface (8 and 16 bit) of the display controller. Parallel, 3-pin SPI or 4-pin SPI access can be used.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_7529
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_7529.h` in the configuration folder. All further compile time configuration macros should be defined in this file. To create a driver device using the GUIDRV_7529 for the given display, e.g. the following command can be used:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_7529, GUICC_5, 0, 0);
```

Detailed information about palette modes can be found in the chapter “Colors” on page 291.

Controller selection

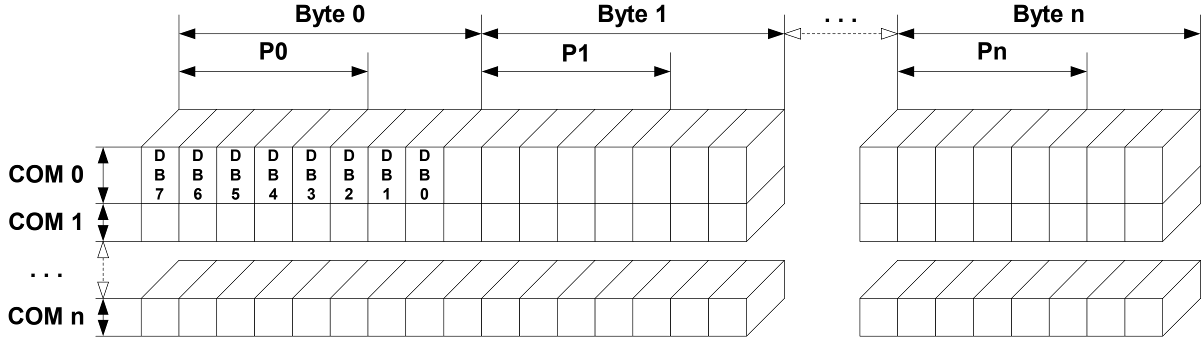
To select the desired controller the macro `LCD_CONTROLLER` should be used in the configuration file `LCDConf_7529.h`. The following table shows the values to be used to select the appropriate controller:

Number	Supported Controller
7529	Sitronix ST7529

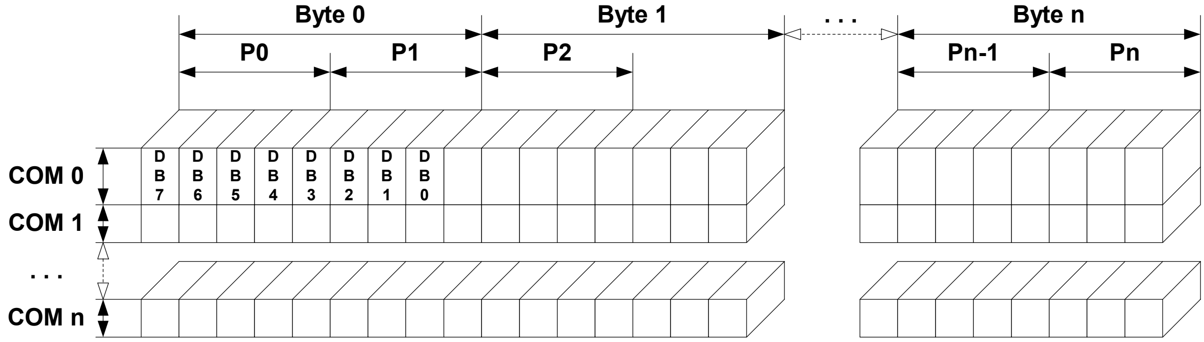
Table 33.155: Controller selection of GUIDRV_7529

Display data RAM organization

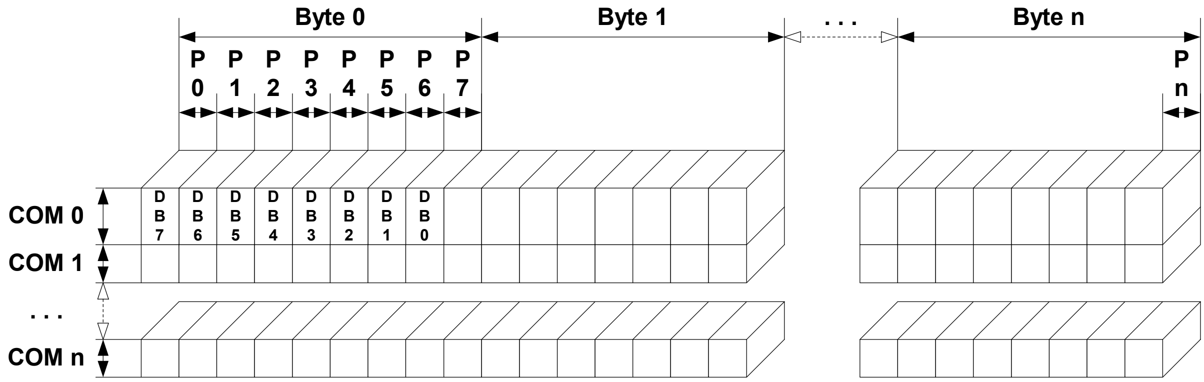
5 bits per pixel, fixed palette = 5 (default)



4 bits per pixel, fixed palette = 4



1 bit per pixel, fixed palette = 1



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

RAM requirements

This display driver can be used with or without a display data cache, containing a complete copy of the LCD data RAM. If no cache is used, there are no additional RAM requirements.
It is optional (but recommended) to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

```
5bpp mode:
Size of RAM (in bytes) =  (LCD_XSIZE + 2) / 3 * 3 * LCD_YSIZE
4bpp mode:
Size of RAM (in bytes) = ((LCD_XSIZE + 2) / 3 * 3 + 1) / 2 * LCD_YSIZE
1bpp mode:
Size of RAM (in bytes) = ((LCD_XSIZE + 2) / 3 * 3 + 7) / 8 * LCD_YSIZE
```


Hardware configuration

This driver accesses the hardware with the indirect interface. The following table lists the macros which must be defined for hardware access:

Macro	Description
LCD_WRITE_A0	Write a byte to LCD controller with A-line low.
LCD_WRITE_A1	Write a byte to LCD controller with A-line high.
LCD_WRITEM_A1	Write multiple bytes to display controller with A-line high.
LCD_READM_A1	Read multiple bytes from display controller with A-line high. Required only if no display data cache is configured.
LCD_FIRSTPIXEL0	If the display size in X is smaller than the number of segment outputs of the display controller, this macro can be used for defining the first visible pixel of the display. It should be used if the first segment lines of the display controller are not connected to the display.

Table 33.156: Hardware access macros

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Description
LCD_CACHE	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

Table 33.157: Additional configuration switches

33.7.22 GUIDRV_Template - Template for a new driver

This driver is part of the basic package and can be easily adapted to each display controller. It contains the complete functionality needed for a display driver.

Adapting the template driver

To adapt the driver to a currently not supported display controller you only have to adapt the routines `_SetPixelIndex()` and `_GetPixelIndex()`. The upper layers calling this routines already make sure that the given coordinates are in range, so that no check on the parameters needs to be performed.
If a display is not readable the function `_GetPixelIndex()` won't be able to read back the contents of the display data RAM. In this case a display data cache should be implemented in the driver, so that the contents of each pixel is known by the driver. If no data cache is available in this case some functions of emWin will not work right. These are all functions which need to invert pixels. Especially the XOR draw mode and the drawing of text cursors (which also uses the XOR draw mode) will not work right. A simple application which does not use the XOR draw mode will also work without adapting the function `_GetPixelIndex()`.
In a second step it should be optimized to improve drawing speed.

33.8 LCD layer and display driver API

emWin requires a driver for the hardware. This chapter explains what an LCD driver for emWin does and what routines it supplies to emWin (the application programming interface, or API).
Under most circumstances, you probably do not need to read this chapter, as most calls to the LCD layer of emWin will be done through the GUI layer. In fact, we recommend that you only call LCD functions if there is no GUI equivalent (for example, if you wish to modify the lookup table of the LCD controller directly). The reason for this is that LCD driver functions are not thread-safe, unlike their GUI equivalents. They should therefore not be called directly in multitask environments.

33.8.1 Display driver API

The table below lists the available emWin LCD-related routines in alphabetical order. Detailed descriptions of the routines can be found in the sections that follow.

LCD layer routines

Routine	Description
"Get" group	
LCD_GetBitsPerPixel()	Return the number of bits per pixel.
LCD_GetBitsPerPixelEx()	Returns the number of bits per pixel of given layer/display.
LCD_GetNumColors()	Return the number of available colors.
LCD_GetNumColorsEx()	Returns the number of available colors of given layer/display.
LCD_GetVXSize()	Return virtual X-size of LCD in pixels.
LCD_GetVXSizeEx()	Returns virtual X-size of given layer/display in pixels.
LCD_GetVYSize()	Return virtual Y-size of LCD in pixels.
LCD_GetVYSizeEx()	Returns virtual Y-size of given layer/display in pixels.
LCD_GetXMag()	Returns the magnification factor in x.
LCD_GetXMagEx()	Returns the magnification factor of given layer/display in x.
LCD_GetXSize()	Return physical X-size of LCD in pixels.
LCD_GetXSizeEx()	Returns physical X-size of given layer/display in pixels.
LCD_GetYMag()	Returns the magnification factor in y.
LCD_GetYMagEx()	Returns the magnification factor of given layer/display in y.
LCD_GetYSize()	Return physical Y-size of LCD in pixels.
LCD_GetYSizeEx()	Returns physical Y-size of given layer/display in pixels.
"Set" group	
LCD_SetAlphaEx()	Sets the layer alpha value.*1
LCD_SetAlphaModeEx()	Enables layer alpha mode. *1
LCD_SetChromaEx()	Sets colors to be used for chroma mode.*1
LCD_SetChromaModeEx()	Enables chroma mode.*1
LCD_SetVisEx()	Sets the visibility of a layer.*1
Configuration group	
LCD_SetDevFunc()	Sets optional or custom defined routines for the display driver.*1
LCD_SetMaxNumColors()	Sets the maximum number of colors used by the application.
LCD_SetSizeEx()	Sets the physical size in pixels of the given layer.
LCD_SetVRAMAddrEx()	Sets the address of the video RAM of the given layer.*1
LCD_SetVSizeEx()	Sets the size of the virtual display area of the given layer.*1
Cache group	
LCD_ControlCache()	Locks, unlocks and flushes the cache of the display controller if it is supported.

Table 33.158: LCD layer routines

Note:
1. Optional function, not supported by each driver.

33.8.2 LCD layer routines

33.8.2.1 "Get" group

LCD_GetBitsPerPixel()

Description

Returns the number of bits per pixel.

Prototype

```
int LCD_GetBitsPerPixel(void);
```

Return value

Number of bits per pixel.

LCD_GetBitsPerPixelEx()

Description

Returns the number of bits per pixel.

Prototype

```
int LCD_GetBitsPerPixelEx(int Index);
```

Parameter	Description
Index	Layer index.

Table 33.159: LCD_GetBitsPerPixelEx() parameter list

Return value

Number of bits per pixel.

LCD_GetNumColors()

Description

Returns the number of currently available colors on the LCD.

Prototype

```
int LCD_GetNumColors(void);
```

Return value

Number of available colors

LCD_GetNumColorsEx()

Description

Returns the number of currently available colors on the LCD.

Prototype

```
U32 LCD_GetNumColorsEx(int Index);
```

Parameter	Description
Index	Layer index.

Table 33.160: LCD_GetNumColorsEx() parameter list

Return value

Number of available colors.

LCD_GetVXSize()

LCD_GetVYSize()

Description

Returns the virtual X- or Y-size, respectively, of the LCD in pixels. In most cases, the virtual size is equal to the physical size.

Prototype

```
int LCD_GetVXSize(void)
int LCD_GetVYSize(void)
```

Return value

Virtual X/Y-size of the display.

LCD_GetVXSizeEx()

LCD_GetVYSizeEx()

Description

Returns the virtual X- or Y-size, respectively, of the LCD in pixels. In most cases, the virtual size is equal to the physical size.

Prototype

```
int LCD_GetVXSizeEx(int Index);
int LCD_GetVYSizeEx(int Index);
```

Parameter	Description
Index	Layer index.

Table 33.161: LCD_GetVXSizeEx() / LCD_GetVYSizeEx() parameter list

Return value

Virtual X/Y-size of the display.

LCD_GetXMag()

LCD_GetYMag()

Description

Returns the magnification factor in X- or Y-axis, respectively.

Prototype

```
int LCD_GetXMag(int Index);
int LCD_GetYMag(int Index);
```

Return value

Magnification factor in X- or Y-axis.

LCD_GetXMagEx()

LCD_GetYMagEx()

Description

Returns the magnification factor in X- or Y-axis, respectively.

Prototype

```
int LCD_GetXMagEx(int Index);
```



```
int LCD_GetYMagEx(int Index);
```

Parameter	Description
Index	Layer index.

Table 33.162: LCD_GetXMagEx() / LCD_GetYMagEx() parameter list

Return value

Magnification factor in X- or Y-axis.

LCD_GetXSize()

LCD_GetYSize()

Description

Returns the physical X- or Y-size, respectively, of the LCD in pixels.

Prototypes

```
int LCD_GetXSize(void)
int LCD_GetYSize(void)
```

Return value

Physical X/Y-size of the display.

LCD_GetXSizeEx()

LCD_GetYSizeEx()

Description

Returns the physical X- or Y-size, respectively, of the LCD in pixels.

Prototype

```
int LCD_GetXSizeEx(int Index);
int LCD_GetYSizeEx(int Index);
```

Parameter	Description
Index	Layer index.

Table 33.163: LCD_GetXSizeEx() / LCD_GetYSizeEx() parameter list

Return value

Physical X/Y-size of the display.

33.8.2.2 "Set" group

LCD_SetAlphaEx()

Description

Sets the layer alpha value of the given layer.

Prototype

```
int LCD_SetAlphaEx(int LayerIndex, int Alpha);
```

Parameter	Description
LayerIndex	Layer index.
Alpha	Alpha value (0-255) to be used. 0 means opaque, 255 fully transparent.

Table 33.164: LCD_SetAlphaEx() parameter list

Return value

0 on success, 1 on error.

Additional information

This feature could only be available if the hardware supports layer alpha blending and if the driver callback function reacts on `LCD_X_SETALPHA`.
Please note that the actual reaction on the given parameter(s) takes place in the driver callback function and depends on the customers implementation. The callback function is responsible for managing the appropriate SFRs to do the operation.

LCD_SetAlphaModeEx()

Description

Enables the layer alpha mode of the given layer.

Prototype

```
int LCD_SetAlphaModeEx(int LayerIndex, int AlphaMode);
```

Parameter	Description
LayerIndex	Layer index.
AlphaMode	1 for enabling layer alpha mode, 0 for pixel alpha mode (should be default).

Table 33.165: LCD_SetAlphaModeEx() parameter list

Return value

0 on success, 1 on error.

Additional information

This feature could only be available if the hardware supports layer alpha blending and if the driver callback function reacts on `LCD_X_SETALPHAMODE`.
Please note that the actual reaction on the given parameter(s) takes place in the driver callback function and depends on the customers implementation. The callback function is responsible for managing the appropriate SFRs to do the operation.
Default behavior of a layer should be pixel alpha mode.

LCD_SetChromaEx()

Description

Sets the colors to be used for the chroma mode.

Prototype

```
int LCD_SetChromaEx(int LayerIndex,
```



```
LCD_COLOR ChromaMin, LCD_COLOR ChromaMax);
```

Parameter	Description
LayerIndex	Layer index.
ChromaMin	See description below.
ChromaMax	See description below.

Table 33.166: LCD_SetChromaEx() parameter list

Return value

0 on success, 1 on error.

Additional information

This feature could only be available if the hardware supports chroma blending and if the driver callback function reacts on LCD_X_SETCHROMA.
The hardware implementations of chroma modes are very different. Because of that the function of the parameters ChromaMin and ChromaMax also could have different meanings. In many cases chroma blending only supports one specific transparent color. In that case only the first parameter ChromaMin should be used. Other systems support a range of color bits to be used or a color and a mask.
Please note that the actual reaction on the given parameter(s) takes place in the driver callback function and depends on the customers implementation. The callback function is responsible for managing the appropriate SFRs to do the operation.

LCD_SetChromaModeEx()

Description

Enables the chroma mode of the given layer.

Prototype

```
int LCD_SetChromaModeEx(int LayerIndex, int ChromaMode);
```

Parameter	Description
LayerIndex	Layer index.
ChromaMode	1 for enabling chroma mode, 0 for disabling (default)

Table 33.167: LCD_SetChromaModeEx() parameter list

Return value

0 on success, 1 on error.

Additional information

This feature could only be available if the hardware supports chroma blending and if the driver callback function reacts on LCD_X_SETCHROMAMODE.
Please note that the actual reaction on the given parameter(s) takes place in the driver callback function and depends on the customers implementation. The callback function is responsible for managing the appropriate SFRs to do the operation.

LCD_SetVisEx()

Description

Sets the visibility of the given layer.

Prototype

```
int LCD_SetVisEx(int LayerIndex, int OnOff);
```

Parameter	Description
LayerIndex	Layer index.
OnOff	1 for visible (default), 0 for invisible.

Table 33.168: LCD_SetVisEx() parameter list

Return value

0 on success, 1 on error.

Additional information

This function works properly only if the display driver callback function appropriately reacts to the command LCD_X_SETVIS. This in turn requires the display driver callback function to manage the appropriate SFRs accordingly. How to do this in detail is explained in the documentation of the display controller.

33.8.2.3 Configuration group

LCD_SetDevFunc()

Description

The function sets additional and / or user defined functions of the display driver.

Prototype

```
int LCD_SetDevFunc(int LayerIndex, int IdFunc, void (* pDriverFunc)(void));
```

Parameter	Description
LayerIndex	Layer index.
IdFunc	See table below.
pDriverFunc	Pointer to function which should be used.

Table 33.169: LCD_SetDevFunc() parameter list

Permitted values for element IdFunc	
LCD_DEVFUNC_COPYBUFFER	Can be used to set a custom defined routine for copying buffers. Makes only sense in combination with multiple buffers.
LCD_DEVFUNC_COPYRECT	Can be used to set a custom defined routine for copying rectangular areas.
LCD_DEVFUNC_DRAWBMP_1BPP	Can be used to se a custom routine for drawing 1bpp bitmaps. Makes sense if a custom routine should be used for drawing text and 1bpp bitmaps.
LCD_DEVFUNC_DRAWBMP_8BPP	Can be used to se a custom routine for drawing 1bpp bitmaps. Makes sense if a custom routine should be used for drawing 8bpp bitmaps.
LCD_DEVFUNC_FILLRECT	Can be used to set a custom defined routine for filling rectangles. Makes sense if for example a BitBLT engine should be used for filling operations. Can be used to set a custom defined routine for reading a single pixel from the display controller.
LCD_DEVFUNC_READMPIXELS	Can be used to set a custom defined routine for reading multiple pixels from the display controller.
LCD_DEVFUNC_READPIXEL	Can be used to set a custom defined routine for reading a single pixel from the display controller.

LCD_DEVFUNC_COPYBUFFER

Can be used to set up a function which copies a frame buffer to the desired location. This can make sense if for example a BitBLT engine is available to do the job.
The function pointed by pDriverFunc should be of the following type:


```
void CopyBuffer(int LayerIndex, int IndexSrc, int IndexDst);
```

Parameter	Description
LayerIndex	Layer index.
IndexSrc	Index of the source frame buffer to be copied.
IndexDst	Index of the destination frame buffer to be overwritten.

Table 33.170: LCD_DEVFUNC_COPYBUFFER parameter list

LCD_DEVFUNC_COPYRECT

Can be used to set up a function which copies a rectangular area of the screen to the desired location. This can make sense if for example a BitBLT engine is available to do the job.
The function pointed by `pDriverFunc` should be of the following type:

```
void CopyRect(int LayerIndex, int x0, int y0, int x1, int y1,
              int xSize,      int ySize);
```

Parameter	Description
LayerIndex	Layer index.
x0	Leftmost pixel of the source rectangle.
y0	Topmost pixel of the source rectangle.
x1	Leftmost pixel of the destination rectangle.
y1	Topmost pixel of the destination rectangle.
xSize	X-size of the rectangle.
ySize	Y-size of the rectangle

Table 33.171: LCD_DEVFUNC_COPYRECT parameter list

LCD_DEVFUNC_DRAWBMP_1BPP

Can be used to set up a function which draws 1bpp bitmaps which includes also text. This can make sense if for example a BitBLT engine is available to do the job.
The function pointed by `pDriverFunc` should be of the following type:

```
void DrawBMP1(int LayerIndex,
              int x, int y, U8 const * p, int Diff, int xSize, int ySize,
              int BytesPerLine, const LCD_PIXELINDEX * pTrans);
```

Parameter	Description
LayerIndex	Layer index.
x	Leftmost coordinate in screen coordinates of the bitmap to be drawn.
y	Topmost coordinate in screen coordinates of the bitmap to be drawn.
p	Pointer to the pixel data of the bitmap.
Diff	Offset to the first pixel pointed by parameter `p`. Supported values are 0-7.
xSize	xSize in pixels of the bitmap to be drawn.
ySize	ySize in pixels of the bitmap to be drawn.
BytesPerLine	Number of bytes of one line of bitmap data.
pTrans	Pointer to an array of color indices to be used to draw the bitmap data. The first color index defines the background color, the second color index defines the foreground color.

Table 33.172: LCD_DEVFUNC_DRAWBMP_1BPP parameter list

Return value

0 on success, 1 on error.

Additional information

Please note that it depends on the display driver which values for parameter `IdFunc` are supported or not.

LCD_DEVFUNC_DRAWBMP_8BPP

Can be used to set up a function which draws 8bpp palette based bitmaps. This can make sense if for example a BitBLT engine is available to do the job.
The function pointed by `pDriverFunc` should be of the following type:

```
void DrawBMP8(int LayerIndex,
              int x, int y, U8 const * p, int xSize, int ySize,
              int BytesPerLine, const LCD_PIXELINDEX * pTrans);
```

Parameter	Description
`LayerIndex`	Layer index.
`x`	Leftmost coordinate in screen coordinates of the bitmap to be drawn.
`y`	Topmost coordinate in screen coordinates of the bitmap to be drawn.
`p`	Pointer to the pixel data of the bitmap.
`xSize`	xSize in pixels of the bitmap to be drawn.
`ySize`	ySize in pixels of the bitmap to be drawn.
`BytesPerLine`	Number of bytes of one line of bitmap data.
`pTrans`	Pointer to an array of color indices to be used to draw the bitmap data. These colors are addressed by the index values of the pixels.

Table 33.173: LCD_DEVFUNC_DRAWBMP_8BPP parameter list

Return value

0 on success, 1 on error.

Additional information

Please note that it depends on the display driver which values for parameter `IdFunc` are supported or not.

LCD_DEVFUNC_FILLRECT

Can be used to set a custom function for filling operations. The function pointed by `pDriverFunc` should be of the following type:

```
void FillRect(int LayerIndex,  int x0, int y0, int x1, int y1,
              U32 PixelIndex);
```

Parameter	Description
`LayerIndex`	Layer index.
`x0`	Leftmost coordinate to be filled in screen coordinates.
`y0`	Topmost coordinate to be filled in screen coordinates.
`x1`	Rightmost coordinate to be filled in screen coordinates.
`y1`	Bottommost coordinate to be filled in screen coordinates.
`PixelIndex`	Color index to be used to fill the specified area.

Table 33.174: LCD_DEVFUNC_FILLRECT parameter list

LCD_DEVFUNC_READMPIXELS

Can be used to set a custom defined routine for reading multiple pixels from the display controller. The function pointed by `pDriverFunc` should be one of the following types:

```
void _ReadMPixels(int LayerIndex, U16 * pBuffer, U32 NumPixels);
void _ReadMPixels(int LayerIndex, U32 * pBuffer, U32 NumPixels);
```

Parameter	Description
`LayerIndex`	Layer index.
`pBuffer`	Pointer to the buffer in which the pixel data has to be stored.
`NumPixels`	Number pixels to read.

Table 33.175: LCD_DEVFUNC_READMPIXELS parameter list

The required funtion type depends on the configured color depth of the display driver. In 16bpp mode a U16 pointer is required for the buffer and for 18bpp up to 32bpp a U32 pointer is required.

LCD_DEVFUNC_READPIXEL

Can be used to set a custom defined routine for reading a single pixel from the display controller. The function pointed by pDriverFunc should be one of the following types:

```
U16 _ReadPixel(int LayerIndex);
U32 _ReadPixel(int LayerIndex);
```

Parameter	Description
LayerIndex	Layer index.

Table 33.176: LCD_DEVFUNC_READPIXEL parameter list

The required type of the return value depends on the configured color depth of the display driver. In 16bpp mode U16 is required and for 18bpp up to 32bpp U32 is required.

LCD_SetMaxNumColors()

Description

Sets the maximum number of colors used in palette based bitmaps.

Prototype

```
int LCD_SetMaxNumColors(unsigned MaxNumColors);
```

Parameter	Description
MaxNumColors	Maximum number of colors used in palette based bitmaps. Default is 256.

Table 33.177: LCD_SetMaxNumColors() parameter list

Return value

0 on success, 1 on error.

Additional information

During the process of initialization emWin allocates a buffer required for converting color values of the bitmaps into index values for the controller. This buffer requires 4 bytes per color. If the system is short on RAM and only a few colors are used, this function could spare up to 1016 bytes of dynamically RAM.
Per default the buffer uses 1024 bytes of RAM. But if for example only 2 colors are used (typically b/w-configuration) only 8 bytes for 2 colors are required.
The function needs to be called by the routine GUI_X_Config().

LCD_SetSizeEx()

Description

Sets the physical size of the visible area of the given display/layer.

Prototype

```
int LCD_SetSizeEx(int LayerIndex, int xSize, int ySize);
```

Parameter	Description
LayerIndex	Layer index.
xSize	X-Size in pixels of the visible area of the given layer.
ySize	Y-Size in pixels of the visible area of the given layer.

Table 33.178: LCD_SetSizeEx() parameter list

Return value

0 on success, 1 on error.

Additional information

The function requires a display driver which is able to manage dynamically changes of the display size. If the display driver does not support this feature the function fails.

LCD_SetVRAMAddrEx()

Description

Sets the address of the video RAM.

Prototype

```
int LCD_SetVRAMAddrEx(int LayerIndex, void * pVRAM);
```

Parameter	Description
LayerIndex	Layer index.
pVRAM	Pointer to start address of video RAM.

Table 33.179: LCD_SetVRAMAddrEx() parameter list

Return value

0 on success, 1 on error.

Additional information

The function requires a display driver which is able to manage dynamically changes of the video RAM address. If the display driver does not support this feature the function fails.

LCD_SetVSizeEx()

Description

Sets the size of the virtual display area.

Prototype

```
int LCD_SetVSizeEx(int LayerIndex, int xSize, int ySize);
```

Parameter	Description
LayerIndex	Layer index.
xSize	X-Size in pixels of the virtual area of the given layer.
ySize	Y-Size in pixels of the virtual area of the given layer.

Table 33.180: LCD_SetVSizeEx() parameter list

Return value

0 on success, 1 on error.

Additional information

The function requires a display driver which is able to manage dynamically changes of the virtual display size. If the display driver does not support this feature the function fails.

33.8.2.4 Cache group

LCD_ControlCache()

Description

Locks, unlocks and flushes the cache of the display controller if it is supported.

Prototype

```
int LCD_ControlCache(int Cmd);
```

Parameter	Description
Cmd	See table below.

Table 33.181: LCD_ControlCache() parameter list

Permitted values for element Cmd	
LCD_CC_FLUSH	Flushes the cache. The content of the cache which has changed since the last flushing operation is output to the display.
LCD_CC_LOCK	Locks the cache. Drawing operations are cached, but not output to the display.
LCD_CC_UNLOCK	Unlocks the cache. The cached data is flushed immediatelly. Further drawing operations are cached and output. (Write Through)

Return value

0 on success, 1 on error.

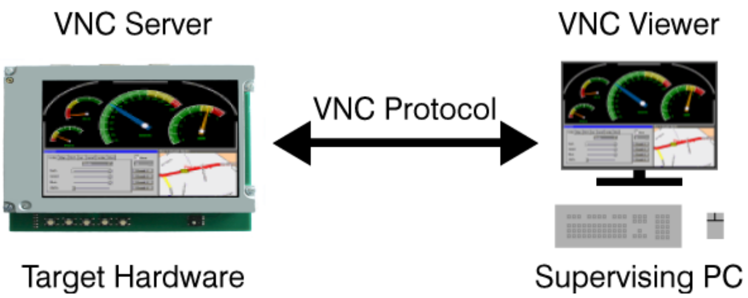
Additional information

The function requires a display driver which is able to manage dynamically changes of the virtual display size. If the display driver does not support this feature the function fails.
This function is automatically used for drawing operations of windows and strings.

Chapter 34

VNC Server

The emWin VNC server can be used for administration of the embedded target and a variety of other purposes. It supports compressed (hextile) encoding.
VNC stands for 'Virtual Network Computing'. It is, a client server system based on a simple display protocol which allows the user to view and control a computing 'desktop' environment from anywhere on the Internet and from a wide variety of machine architectures, communicating via TCP/IP.
In other words: The display contents of the embedded device are visible on the screen of the machine running the client (for example, your PC); your mouse and keyboard can be used to control the target.
This feature is available in the emWin simulation and trial versions.
emWin VNC support is available as a separate package and is therefore not included in the basic package. VNC support requires emWin color.



34.1 Introduction

VNC consists of two types of components. A server, which generates a display, and a viewer, which actually draws the display on your screen. The remote machine (target or simulation) can not only be viewed, but also controlled via mouse or keyboard.
The server and the viewer may be on different machines and on different architectures. The protocol which connects the server and viewer is simple, open, and platform independent. No state is stored at the viewer. Breaking the viewer's connection to the server and then reconnecting will not result in any loss of data. Because the connection can be remade from somewhere else, you have easy mobility. Using the VNC server, you may control your target from anywhere and you can make screenshots (for example, for a manual) from a "live" system.

34.1.1 Requirements

TCP/IP stack

Since the communication between the server and the viewer is based on a TCP/IP connection, VNC requires a TCP/IP stack. In the Win32 simulation environment, TCP/IP (Winsock) is normally present. In the target, a TCP/IP stack needs to be present. The TCP/IP stack is NOT part of emWin. The flexible interface ensures that any TCP/IP stack can be used.

Multi tasking

The VNC server needs to run as a separate thread. Therefore a multi tasking system is required to use the emWin VNC server.

34.1.2 Notes on this implementation

Supported client to server messages

The emWin VNC server supports pointer event messages and keyboard event messages.

Encoding

The server supports raw encoding and hextile encoding.

Performance

Most viewers support hextile encoding, which supports descent compression. A typical quarter VGA screen requires typically 20 - 50 kb of data. An implementation running on an ARM7 platform (50 MHZ, with Cache) requires app. 200 - 300 ms for an update of the entire screen.
The server handles incremental updates; in most cases the updated display area is a lot smaller than the entire display and less data needs to be transmitted. A typical ARM7 system therefore allows real time updates.

Multiple servers

The implementation is fully thread safe and reentrant; multiple VNC-servers can be started on the same CPU for different layers or displays. If your target (of course the same holds true for the simulation) has multiple displays or multiple layers, this can be a useful option. Only one VNC server may be started per layer at any given time; once the connection to a Viewer ends, another one can connect.

34.2 emWin VNC viewer

The emWin VNC viewer is part of the emWin basic package. It can be used to establish a VNC connection from an MS Windows system. The viewer uses the RFB protocol 3.3. It has been tested with different VNC servers including the emWin VNC server, as well as TightVNC and RealVNC.

34.2.1 How to use the VNC viewer

Once the VNC viewer was started, it will prompt for typing the network address of a VNC server to connect with:

Connecting to a VNC server using the simulation on the same PC

A VNC server running on the local host can be accessed by entering:

```
localhost
```

Alternatively the above dialog can just be In order to connect to the local host, it is sufficient to just hit the RETURN key or the "Connect" button while the text control is left empty.

Connecting to a VNC server running on a different PC or the target

In order to connect to a system in the network the IP address or the name has to be entered:

`192.168.1.14` or `Paul02`

Additionally the server index can be specified in order to connect to a certain server:

`192.168.1.14:1` or `Paul02:1`

Screenshot

The following screenshots show the viewer:

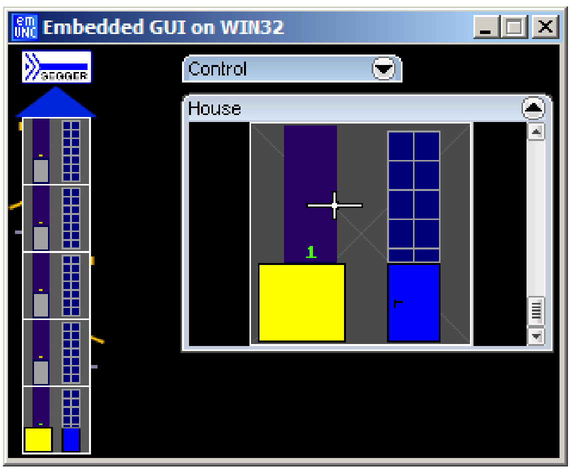
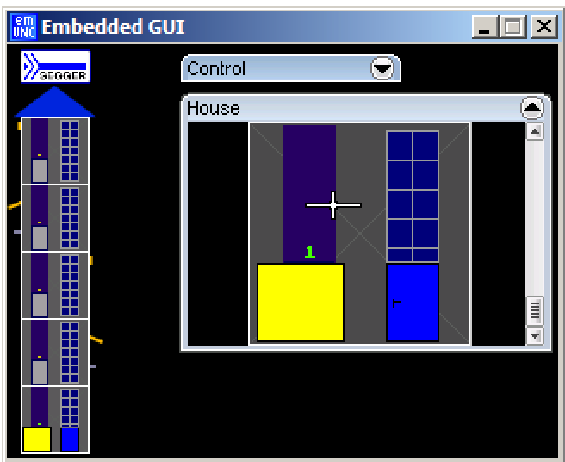
Connected to the simulation	Connected to the target
	

Table 34.1: VNC Viewer screenshots

34.3 emWin VNC server

34.3.1 Starting the emWin VNC server

The one and only thing to start the VNC server is to call the function GUI_VNC_X_StartServer():

```
void MainTask(void) {
  GUI_Init();
  GUI_VNC_X_StartServer(0,    // Layer index
                        0);   // Server index
  ...
}
```

The above function call creates a thread which listens on port 5900 for an incoming connection. After a connection has been detected GUI_VNC_Process() will be called.

Ports

The VNC server listens on port 590x, where x is the server index. So for most PC servers, the port will be 5900, because they use display 0 by default.

Example

A ready to use example (in executable form) is available on our website. The trial version also contains the VNC server; it takes no more than one line of code (using GUI_VNC_X_StartServer()) to activate it.

34.3.2 How the server starts...

When using the simulation, only the function GUI_VNC_X_StartServer() needs to be called. It creates a thread which listens on port 590x until an incoming connection is detected and then calls GUI_VNC_Process(), which is the implementation of the actual server.

34.3.3 Integration of the VNC server on the target

Before the function GUI_VNC_X_StartServer() can be used, it has to be adapted to the used TCP/IP stack and the multi tasking system. An implementation example is available under Sample\GUI_X\GUI_VNC_X_StartServer.c, which should require only smaller modifications. Since this example does not use dynamic memory allocation to allocate memory for the GUI_VNC_CONTEXT structure, which is described, this implementation allows starting only one server.

34.4 Requirements

ROM

About 4.9 kb on ARM7 with hextile encoding, about 3.5 kb without hextile encoding.

RAM

The VNC support does not use static data. For each instance one GUI_VNC_CONTEXT structure (app. 60 bytes) is used.

Others

Each instance needs one TCP/IP socket and one thread.

34.5 Limitations

The emWin VNC server does not support the 32bpp color format. This is usually not a problem, unless the viewer insists on being served with 32bpp. This can be solved by changing the color format in the viewer options.

34.6 Configuration options

Type	Macro	Default	Description
N	GUI_VNC_BUFFER_SIZE	1000	Size of the receive buffer. The buffer is located on the stack. Typically bigger sizes result in only minor accelerations. A reasonable buffer size is app. 200 bytes.
N	GUI_VNC_HEXTILE_VERSION	3	Configures the hextile algorithm to use: 1) Linear compression, single reads A: No additional RAM requirement. D: Slow reading of display data, poor compression result 2) Linear compression, block read A: Fast reading of display data D: Additional RAM buffer for pixel data needed, poor compression result 3) 2-D compression, block read A: Fast reading of display data, good compression result D: Additional RAM buffer for pixel data needed
B	GUI_VNC_LOCK_FRAME	0	Enabling this option is required for systems using an indirect interface to the display controller. If the VNC server sends the content of the display to the client, it calls the display driver to get the current content. With direct interface this is not a problem. When using an indirect interface this option ensures, that writing operations of the GUI task will not be interrupted by reading operations.
S	GUI_VNC_PROGNAME	See description.	This macro defines the name of the target shown in the title bar of the viewer. If using the viewer in the simulation the default is: "Embedded GUI on WIN32" On the target the default is: "Embedded GUI"
B	GUI_VNC_SUPPORT_HEXTILE	1	Enables or disables hextile encoding. Hextile encoding is faster but needs bigger code (app. 1.4K additional bytes).

Table 34.2: Configuration options

34.7 VNC Server API

The following table lists the available VNC-related functions in alphabetical order. Detailed function descriptions follow:

Routine	Description
GUI_VNC_AttachToLayer()	Attaches a VNC server to a layer. Without a MultiLayer configuration the given index must be 0.
GUI_VNC_EnableKeyboardInput()	Enables or disables keyboard input via VNC.
GUI_VNC_GetNumConnections()	Return the number of connections to the server.
GUI_VNC_Process()	The actual VNC server; initializes the communication with the viewer.
GUI_VNC_RingBell()	Ring a bell on the client if it has one.
GUI_VNC_SetLockFrame()	Configures the VNC server not to read the display while the GUI performs drawing operations.
GUI_VNC_SetPassword()	Sets the password required to connect with the server.
GUI_VNC_SetProgName()	Sets the text to be shown in the viewers title bar.
GUI_VNC_SetSize()	Sets the area to be transmitted to the client.
GUI_VNC_X_StartServer()	Routine to be called to start a VNC viewer.

Table 34.3: VNC Server API list

GUI_VNC_AttachToLayer()

Description

This function attaches the given layer to the VNC server. Normally, with single layer configurations, this parameter should be 0.

Prototype

```
void GUI_VNC_AttachToLayer(GUI_VNC_CONTEXT * pContext, int LayerIndex);
```

Parameter	Description
pContext	Pointer to a GUI_VNC_CONTEXT structure.
LayerIndex	Zero based index of layer to be handled by the server.

Table 34.4: GUI_VNC_AttachToLayer() parameter list

Return value

0 if the function succeed, != 0 if the function fails.

GUI_VNC_EnableKeyboardInput()

Description

Enables or disables keyboard input via VNC.

Prototype

```
void GUI_VNC_EnableKeyboardInput(int OnOff);
```

Parameter	Description
OnOff	1 for enabling keyboard input, 0 for disabling.

Table 34.5: GUI_VNC_EnableKeyboardInput() parameter list

GUI_VNC_GetNumConnections()

Description

Returns the number of currently existing connections to the server.

Prototype

```
int GUI_VNC_GetNumConnections(void);
```

Return value

Number of connections.

GUI_VNC_Process()

Description

The function sets the send and receive function used to send and receive data and starts the communication with the viewer.

Prototype

```
void GUI_VNC_Process(GUI_VNC_CONTEXT * pContext,
                     GUI_tSend         pfSend,
                     GUI_tReceive      pfReceive,
                     void            * pConnectInfo);
```

Parameter	Description
pContext	Pointer to a GUI_VNC_CONTEXT structure.

Table 34.6: GUI_VNC_Process() parameter list

Parameter	Description
pfSend	Pointer to the function to be used by the server to send data to the viewer.
pfReceive	Pointer to the function to be used by the server to read from the viewer.
pConnectInfo	Pointer to be passed to the send and receive function.

Table 34.6: GUI_VNC_Process() parameter list

Additional information

The GUI_VNC_CONTEXT structure is used by the server to store connection state information.
The send and receive functions should return the number of bytes successfully send/received to/from the viewer.
The pointer pConnectInfo is passed to the send and receive routines. It can be used to pass a pointer to a structure containing connection information or to pass a socket number.
The following types are used as function pointers to the routines used to send and receive bytes from/to the viewer:

```
typedef int (*GUI_tSend)    (const U8 * pData, int len, void * pConnectInfo);
typedef int (*GUI_tReceive)(       U8 * pData, int len, void * pConnectInfo);
```

Example

```
static GUI_VNC_CONTEXT _Context; /* Data area for server */

static int _Send(const U8* buf, int len, void * pConnectionInfo) {
  SOCKET Socket = (SOCKET)pConnectionInfo;
 ...
}
static int _Recv(U8* buf, int len, void * pConnectionInfo) {
  SOCKET Socket = (SOCKET)pConnectionInfo;
  ...
}
static void _ServerTask(void) {
  int Socket;
  ...
  GUI_VNC_Process(&_Context, _Send, _Recv, (void *)Socket);
  ...
}
```

GUI_VNC_RingBell()

Description

Ring a bell on the client if it has one.

Prototype

```
void GUI_VNC_RingBell(void);
```

GUI_VNC_SetLockFrame()

Description

Configures the VNC server not to read the display while the GUI performs drawing operations.

Prototype

```
void GUI_VNC_SetLockFrame(unsigned OnOff);
```

Parameter	Description
OnOff	If set to a value >0 frame locking will be enabled. Default is enabled frame locking.

Table 34.7: GUI_VNC_SetLockFrame() parameter list

Additional information

This can be configured at compile time by using the compile time switch GUI_VNC_LOCK_FRAME.

GUI_VNC_SetPassword()

Description

Sets a password required to connect to the server.

Prototype

```
void GUI_VNC_SetPassword(U8 * sPassword);
```

Parameter	Description
sPassword	Password required to connect to the server.

Table 34.8: GUI_VNC_SetPassword() parameter list

Additional information

Per default no password is required.
If a password is set the server creates a random challenge of 16 Bytes and encrypts it using DES. The unencrypted challenge is sent to the client and should return encrypted. If the client's response matches the encrypted challenge, authentification was successful.

GUI_VNC_SetProgName()

Description

Sets the title to be displayed in the title bar of the client window.

Prototype

```
void GUI_VNC_SetProgName(const char * sProgName);
```

Parameter	Description
sProgName	Title to be displayed in the title bar of the client window.

Table 34.9: GUI_VNC_SetProgName() parameter list

GUI_VNC_SetSize()

Description

Sets the display size to be transmitted to the client.

Prototype

```
void GUI_VNC_SetSize(unsigned xSize, unsigned ySize);
```

Parameter	Description
xSize	X-size to be used.
ySize	Y-size to be used.

Table 34.10: GUI_VNC_SetSize() parameter list

Additional information

Per default the server uses the layer size. The size passed to this function can be smaller or larger than the real display.

GUI_VNC_X_StartServer()

Description

Starts a VNC viewer with the given server index to display the given layer in the viewer.
The function has to be written by the customer because the implementation depends on the used TCP/IP stack and on the used operating system.
The emWin shipment contains an example implementation under `Sample\GUI_X\GUI_VNC_X_StartServer.c`. It could be used as a starting point for adapting it to other systems.

Prototype

```
int GUI_VNC_X_StartServer(int LayerIndex, int ServerIndex);
```

Parameter	Description
LayerIndex	Layer to be shown by the viewer.
ServerIndex	Server index.

Table 34.11: GUI_VNC_X_StartServer() parameter list

Additional information

There is no difference to start a VNC server in the simulation or on the target. In both cases you should call this function. The simulation contains an implementation of this function, the hardware implementation has to be done by the customer.

Chapter 35

Touch drivers

A touch driver supports a particular family of touch controllers and all touch pads which are connected to one of these controllers. The drivers can be configured by modifying their configuration files whereas the driver itself does not need to be modified. The configuration files contain all required information for the driver including how the hardware is accessed and how the controller(s) are connected to the display. This chapter provides an overview of the touch drivers available for emWin. It explains the following in terms of each driver:

- Which touch controllers can be accessed and which interface can be used.
- RAM requirements.
- Driver specific functions.
- How to access the hardware.
- Special configuration switches.
- Special requirements for particular touch controllers.

35.1 GUIMTDRV_TangoC32

The driver is written for the multi touch controller TangoC32 from PIXCIR. It is delivered along with the emWin MultiTouch feature.
The controller can be accessed via I2C interface. It provides an interrupt line which needs to be used by the application to generate an interrupt. Once the driver has been initialized right it automatically fills up the multi touch buffer of emWin.

Supported hardware

This driver works with the following controller:
- PIXCIR Tango C32

Driver initialization

A good place for initializing the touch driver is the routine `LCD_X_Config()`. This makes sure, that the touch driver and the display driver has been initialized before emWin is used by the application.

First part

The first part of initializing the driver is calling the drivers configuration function. It sets up the function pointers for hardware communication.

Second part

To be able to do its work the drivers execution function needs to be called when touching the screen. That should be done via interrupt routine. For that case the touch controller provides an interrupt line which is active if a touch event occurs. The one and only thing which then should be done in the interrupt routine is calling the drivers execution function `GUIMTDRV_TangoC32_Exec()`.

GUIMTDRV_TangoC32 API

The following table shows the available functions of the driver.

Routine	Description
`GUIMTDRV_TangoC32_Init()`	Configuration function.
`GUIMTDRV_TangoC32_Exec()`	Execution function.

Table 35.1: GUIMTDRV_TangoC32 API list

GUIMTDRV_TangoC32_Init()

Description

Passes a pointer to a `GUIMTDRV_TANGOC32_CONFIG` structure to the driver. This structure contains all required function pointers and values required by the driver.

Prototype

```
int GUIMTDRV_TangoC32_Init(GUIMTDRV_TANGOC32_CONFIG * pConfig);
```

Parameter	Description
`pConfig`	Pointer to a `GUIMTDRV_TANGOC32_CONFIG` structure described below.

Table 35.2: GUIMTDRV_TangoC32_Init() parameter list

Elements of structure GUIMTDRV_TANGOC32_CONFIG

Data type	Element
`void (*)(U8 SlaveAddr)`	`pf_I2C_Init`
`int (*)(U8 * pData, int Start, int Stop)`	`pf_I2C_Read`
`int (*)(U8 * pData, int NumItems, int Start, int Stop)`	`pf_I2C_ReadM`
`int (*)(U8 Data, int Start, int Stop)`	`pf_I2C_Write`
`int (*)(U8 * pData, int NumItems, int Start, int Stop)`	`pf_I2C_WriteM`
`U8`	`SlaveAddr`

Table 35.3: GUIMTDRV_TANGOC32_CONFIG element list

pf_I2C_Init()

Parameter	Description
`SlaveAddr`	I2C Slave address of touch controller to be used.

Table 35.4: pf_I2C_Init()

That pointer should point to a function which initializes the I2C communication. The element `SlaveAddr` is passed to the given function to set up the slave address of the touch controller device (normally 0x5C).

pf_I2C_Read()

Parameter	Description
`pData`	Pointer to an unsigned character to store the byte.
`Start`	Is set to 1 if bus communication starts, otherwise 0.
`Stop`	Is set to 1 if bus communication should end after the read operation, otherwise 0.

Table 35.5: pf_I2C_Read() parameter list

That function is responsible for reading one byte of data. The given pointer `pData` is used to store the value.
Returns 0 on success, otherwise 1.

pf_I2C_ReadM()

Parameter	Description
`pData`	Pointer to a buffer to be filled by the routine.
`NumItems`	Number of bytes to be read.
`Start`	Is set to 1 if bus communication starts, otherwise 0.
`Stop`	Is set to 1 if bus communication should end after the read operation, otherwise 0.

Table 35.6: pf_I2C_ReadM() parameter list

That function is responsible for reading multiple bytes of data. The given pointer `pData` is used to store the value.
Returns 0 on success, otherwise 1.

pf_I2C_Write()

Parameter	Description
`Data`	Byte to be written.
`Start`	Is set to 1 if bus communication starts, otherwise 0.
`Stop`	Is set to 1 if bus communication should end after the write operation, otherwise 0.

Table 35.7: pf_I2C_Write() parameter list

That function is responsible for writing one byte of data.
Returns 0 on success, otherwise 1.

pf_I2C_WriteM()

Parameter	Description
pData	Pointer to buffer to be written.
NumItems	Number of bytes to be written.
Start	Is set to 1 if bus communication starts, otherwise 0.
Stop	Is set to 1 if bus communication should end after the write operation, otherwise 0.

Table 35.8: pf_I2C_WriteM() parameter list

That function is responsible for writing multiple bytes of data. The given pointer pData is used to store the value.
Returns 0 on success, otherwise 1.

35.2 GUITDRV_ADS7846

Supported hardware

This driver works with the following controller:
- Texas Instruments ADS7846 touch screen controller

Driver initialization

A good place for initializing the touch driver is the routine `LCD_X_Config()`. This makes sure, that the touch driver and the display driver has been initialized before emWin is used by the application.

First part

The first part of initializing the driver is calling the drivers configuration function. It sets up the following things:

- Function pointers for hardware communication routines
- Touch panel orientation to be used
- Logical and physical AD values to be able to calculate the right position depending on the AD values of the controller

Second part

To be able to do its work the drivers execution function needs to be called periodically. We recommend an interval of 20-30 ms. The function call can be done from within a timer interrupt routine or from a separate task.

GUITDRV_ADS7846 API

The following table shows the available functions of the driver.

Routine	Description
`GUITDRV_ADS7846_Config()`	Configuration function.
`GUITDRV_ADS7846_Exec()`	Execution function.
`GUITDRV_ADS7846_GetLastVal()`	Retrieves the last stored values.

Table 35.9: GUITDRV_ADS7846 API list

GUITDRV_ADS7846_Config()

Description

Passes a pointer to a `GUITDRV_ADS7846_CONFIG` structure to the driver. This structure contains all required function pointers and values required by the driver.

Prototype

```
void GUITDRV_ADS7846_Config(GUITDRV_ADS7846_CONFIG * pConfig);
```

Parameter	Description
`pConfig`	Pointer to a `GUITDRV_ADS7846_CONFIG` structure described below.

Table 35.10: GUITDRV_ADS7846_Config() parameter list

Elements of structure GUITDRV_ADS7846_CONFIG

Data type	Element	Description
void (*)(U8 Data)	pfSendCmd	Hardware routine for sending a byte to the controller via its SPI interface.
U16 (*)(void)	pfGetResult	Hardware routine for getting the AD conversion result of the controller via its SPI interface. The driver uses the 12 bit conversion mode. Per conversion the controller uses 16 clocks. Only the first 12 bits contain the result to be returned by this routine.
char (*)(void)	pfGetBusy	Hardware routine for getting the busy state of the controller. The routine should return 1 if the controller is busy and 0 if not.
void (*)(char OnOff)	pfSetCS	Routine for toggling the CS signal of the controller. When receiving 1 the signal should become high and vice versa.
unsigned	Orientation	One or more "OR" combined values of the table below.
int	xLog0	Logical X value 0 in pixels.
int	xLog1	Logical X value 1 in pixels.
int	xPhys0	A/D converter value for xLog0.
int	xPhys1	A/D converter value for xLog1.
int	yLog0	Logical Y value 0 in pixels.
int	yLog1	Logical Y value 1 in pixels.
int	yPhys0	A/D converter value for yLog0.
int	yPhys1	A/D converter value for yLog1.
char (*)(void)	pfGetPENIRQ	If the PENIRQ line of the touch controller is connected to a port of the target hardware a touch event can be detected by the driver. Upon polling the driver's exec routine the driver can check if a touch event is ready to be sampled by checking the PENIRQ line. Without PENIRQ line the driver will always try to sample a touch event even if no touch happened which will consume time even if not necessary. Without PENIRQ it is the responsibility of the user's pfGetResult() routine to return 0xFFFF if the measured AD value is out of bounds. If both, the PENIRQ and the touch pressure recognition are enabled first the PENIRQ will signal that there is a touch event. Afterwards the touch pressure measurement is used to confirm that this was a valid touch and the touch had enough pressure to deliver good measurements. The routine should return 1 if a touch event is recognized and 0 if not.
int	PressureMin	Minimum pressure threshold. A measured pressure below this value means we do not have a valid touch event.
int	PressureMax	Maximum pressure threshold. A measured pressure above this value means we do not have a valid touch event.
int	PlateResistanceX	Resistance of the X-plate of the touch screen. This value is needed for calculation of the touch pressure.

Table 35.11: GUITDRV_ADS7846_CONFIG element list

Permitted values for element Orientation	
GUI_MIRROR_X	Mirroring the X-axis
GUI_MIRROR_Y	Mirroring the Y-axis
GUI_SWAP_XY	Swapping X- and Y-axis

GUITDRV_ADS7846_Exec()

Description

Execution function of the touch driver.

Prototype

```
char GUITDRV_ADS7846_Exec(void);
```

Additional information

We recommend to call the routine each 20-30 ms. If the routine detects a valid touch event it stores the result into the touch buffer via a function call to `GUI_TOUCH_StoreStateEx()`.
Please note that the driver needs some function pointers to be filled correctly to be able to communicate with the external peripheral. The correct assignment of these function pointers is checked during driver configuration and leads to an abort to `GUI_Error()` on missing pointers.

GUITDRV_ADS7846_GetLastVal()

Description

Retrieves the last stored values for some internal variables that might be needed for calibration of the driver without knowing its internals.

Prototype

```
void GUITDRV_ADS7846_GetLastVal(GUITDRV_ADS7846_LAST_VAL * p);
```

Parameter	Description
p	Pointer to a GUITDRV_ADS7846_LAST_VAL structure.

Table 35.12: GUITDRV_ADS7846_GetLastVal() parameter list

Elements of structure GUITDRV_ADS7846_LAST_VAL

Data type	Element	Description
int	xPhys	Last measured x value
int	yPhys	Last measured y value
int	z1Phys	Last measured z1 value
int	z2Phys	Last measured z2 value
int	PENIRQ	Last sampled PENIRQ state if PENIRQ callback has been set
int	Pressure	Last measured touch pressure if touch pressure measurement is enabled

Table 35.13: GUITDRV_ADS7846_LAST_VAL element list

Additional information

This function is an optional function and not required to be able to use the driver.

Chapter 36

Timing- and execution-related functions

Some widgets, as well as our demonstration code, require time-related functions. The other parts of the emWin graphic library do not require a time base.
The demonstration code makes heavy use of the routine `GUI_Delay()`, which delays for a given period of time. A unit of time is referred to as a tick.

36.1 Timing and execution API

The table below lists the available timing- and execution-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
GUI_Delay()	Delays the user application for the specified period of time.
GUI_Error()	Shows a message box and stops execution.
GUI_Exec()	Executes all pending emWin tasks.
GUI_Exec1()	Executes the next pending emWin task.
GUI_GetTime()	Returns the current system time.

Table 36.1: Timing and execution API list

GUI_Delay()

Description

Delays for a specified period of time.

Prototype

```
void GUI_Delay(int Period);
```

Parameter	Description
Period	Period in ticks until function should return.

Table 36.2: GUI_Delay() parameter list

Additional information

The time unit (tick) is usually milliseconds (depending on GUI_X_ functions).
GUI_Delay() only executes idle functions for the given period. If the Window Manager is used, the delay time is used for the updating of invalid windows (through execution of WM_Exec()).
This function will call GUI_X_Delay().

GUI_Error()

Description

This function is called by emWin in case of serious errors which causes the system to stop execution. It gets a pointer to a string which should contain a short error description. It should contain module and function where the error occurred and a short description. The simulation automatically shows a message box with error description in debug mode. To be able to intercept these major errors on the target system, the function GUI_SetOnErrorFunc() can be used to set up a custom routine which is called by GUI_Error().

Prototype

```
void GUI_Error(const char * s);
```

Parameter	Description
s	Error string which is passed on to the function GUI_X_ErrorOut() and to the user defined error handling function.

Table 36.3: GUI_Error() parameter list

Additional information

Detailed information on how to set up a user defined error handling function can be found in the description of the function GUI_SetOnErrorFunc in the chapter Configuration on page 1210.

GUI_Exec()

Description

Executes callback functions (typically redrawing of windows).

Prototype

```
int GUI_Exec(void);
```

Return value

0 if there were no jobs performed.
1 if a job was performed.

Additional information

This function will automatically call `GUI_Exec1()` repeatedly until it has completed all jobs -- essentially until a 0 value is returned.
Normally this function does not need to be called by the user application. It is called automatically by `GUI_Delay()`.

GUI_Exec1()

Description

Executes a callback function (one job only -- typically redrawing a window).

Prototype

```
int GUI_Exec1(void);
```

Return value

0 if there were no jobs performed.
1 if a job was performed.

Additional information

This routine may be called repeatedly until 0 is returned, which means all jobs have been completed.
This function is called automatically by `GUI_Exec()`.

GUI_GetTime()

Description

Returns the current system time.

Prototype

```
GUI_TIMER_TIME GUI_GetTime(void);
```

Return value

The current system time in ticks.

Additional information

This function calls `GUI_X_GetTime()`. GUI_TIMER_TIME is explained under "GUI_TIMER_TIME" on page 1192.

36.2 Timer API

The table below lists the available timer-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Description
GUI_TIMER_Create()	Creates a timer.
GUI_TIMER_Delete()	Deletes the given timer.
GUI_TIMER_Restart()	Restarts the given timer.
GUI_TIMER_SetPeriod()	Sets the timer period.

Table 36.4: Timer API list

GUI_TIMER_Create()

Description

Creates a timer. When the timer expires the timer callback function is called.

Prototype

```
GUI_TIMER_HANDLE GUI_TIMER_Create(GUI_TIMER_CALLBACK * cb,
                                  GUI_TIMER_TIME       Time,  U32 Context,
                                  U16                  Flags);
```

Parameter	Description
cb	Pointer to the user defined timer callback function which is called when the timer expires. Prototype is shown below.
Time	Destination time. The created timer expires when the system time exceeds this value.
Context	Timer context which is returned unchanged via timer callback function.
Flags	Not used. Reserved for future use.

Table 36.5: GUI_TIMER_Create() parameter list

GUI_TIMER_CALLBACK

```
typedef void GUI_TIMER_CALLBACK(GUI_TIMER_MESSAGE * pTM);
```

Parameter	Description
pTM	Pointer to a GUI_TIMER_MESSAGE structure which is explained below. Changes which are done from within the callback function are not applied. In order to have another context, a new timer should be created.

Table 36.6: GUI_TIMER_CALLBACK parameter list

Elements of structure GUI_TIMER_MESSAGE

Data type	Element	Description
GUI_TIMER_TIME	Time	Contains the time value when the timer expired.
U32	Context	User defined context value which was specified at creation of the timer.
GUI_TIMER_HANDLE	hTimer	Handle of the expired timer.

Table 36.7: GUI_TIMER_MESSAGE element list

GUI_TIMER_TIME

This define can be set to the desired type in the file `GUIConf.h`. The default type is `int`.

Return value

Handle to the created timer. 0, if no timer was created.

Additional information

Timers are not deleted automatically. To delete a timer the function GUI_TIMER_Delete() can be used. Restarting a timer can be achieved with GUI_TIMER_Restart().

GUI_TIMER_Delete()

Description

Deletes the given timer.

Prototype

```
void GUI_TIMER_Delete(GUI_TIMER_HANDLE hObj);
```

Parameter	Description
hObj	Timer handle.

Table 36.8: GUI_TIMER_Delete() parameter list

Additional information

Timers are deleted immediately. After deleting a timer the according callback function will not be triggered.

GUI_TIMER_Restart()

Description

Restarts the given timer.

Prototype

```
void GUI_TIMER_Restart(GUI_TIMER_HANDLE hObj);
```

Parameter	Description
hObj	Timer handle.

Table 36.9: GUI_TIMER_Restart() parameter list

GUI_TIMER_SetPeriod()

Description

Sets the timer period. The period defines the time which has to pass until the callback function is triggered again.

Prototype

```
void GUI_TIMER_SetPeriod(GUI_TIMER_HANDLE hObj, GUI_TIMER_TIME Period);
```

Parameter	Description
hObj	Timer handle.
Period	Timer period.

Table 36.10: GUI_TIMER_SetPeriod() parameter list

Additional information

This period is used only when the timer is restarted.

Chapter 37

Performance and Resource Usage

High performance combined with low resource usage has always been a major design consideration. emWin runs on 8/16/32-bit CPUs. Depending on which modules are being used, even single-chip systems with less than 64 Kbytes ROM and 2 Kbytes RAM can be supported by emWin. The actual performance and resource usage depends on many factors (CPU, compiler, memory model, optimization, configuration, display controller interface, etc.). This chapter contains benchmarks and information about resource usage in typical systems which can be used to obtain sufficient estimates for most target systems.

37.1 Performance

The following chapter shows driver benchmarks on different targets and performance values of image drawing operations.

37.1.1 Driver benchmark

We use a benchmark test to measure the speed of the display drivers on available targets. This benchmark is in no way complete, but it gives an approximation of the length of time required for common operations on various targets.

Configuration and performance table

CPU	LCD Controller (Driver)	bpp	Bench1 Filling	Bench2 Small fonts	Bench3 Big fonts	Bench4 Bitmap 1bpp	Bench5 Bitmap 2bpp	Bench6 Bitmap 4bpp	Bench7 Bitmap 8bpp	Bench8 DDP bitmap
V850SB1 (20MHz)	S1D13806 (1300)	8	16.7M	339K	1.59M	1.52M	240K	459K	83K	1.25M
V850SB1 (20MHz)	S1D13806 (1300)	16	8.33M	326K	1.45M	1.49M	391K	388K	214K	806K
ARM720T (50MHz)	(internal) (3200)	16	7.14M	581K	1.85M	1.96M	694K	645K	410K	2.94M
ARM926EJ-S (200MHz)	(internal) (3200)	16	123M	3.79M	5.21M	7.59M	2.27M	2.21M	1.77M	15.2M

Table 37.1: Configuration and performance table

M - Megapixels / second
K - Kilopixels / second

Bench1: Filling

Bench the speed of filling. An area of 64*64 pixels is filled with different colors.

Bench2: Small fonts

Bench the speed of small character output. An area of 60*64 pixels is filled with small-character text.

Bench3: Big fonts

Bench the speed of big character output. An area of 65*48 pixels is filled with big-character text.

Bench4: Bitmap 1bpp

Bench the speed of 1bbp bitmaps. An area of 58*8 pixels is filled with a 1bpp bitmap.

Bench 5: Bitmap 2bpp

Bench the speed of 2bbp bitmaps. An area of 32*11 pixels is filled with a 2bpp bitmap.

Bench6: Bitmap 4bpp

Bench the speed of 4bbp bitmaps. An area of 32*11 pixels is filled with a 4bpp bitmap.

Bench7: Bitmap 8bpp

Bench the speed of 8bbp bitmaps. An area of 32*11 pixels is filled with a 8bpp bitmap.

Bench8: Device-dependent bitmap, 8 or 16 bpp

Bench the speed of bitmaps 8 or 16 bits per pixel. An area of 64*8 pixels is filled with a bitmap. The color depth of the tested bitmap depends on the configuration. For configurations <= 8bpp, a bitmap with 8 bpp is used; 16bpp configurations use a 16-bpp bitmap.

37.1.2 Image drawing performance

The purpose of the following table is to show the drawing performance of the various image formats supported by emWin. The measurement for the following table has been done on an ARM922T CPU (Sharp LH7A404) running with 200MHz and with 15 bpp display color depth (fixed palette = 555) using GUIDRV_Lin:

Image format	Megapixels / second
Internal bitmap format: 1bpp C file	17.186
Internal bitmap format: 4bpp C file	3.897
Internal bitmap format: 8bpp C file	4.017
Internal bitmap format: 8bpp C file, without palette	4.478
Internal bitmap format: 16bpp C file, high color 555	13.363
Internal bitmap format: 16bpp C file, high color 565	1.336
Internal bitmap format: 24bpp C file, true color 888	1.671
Internal bitmap format: RLE4 C file	6.144
Internal bitmap format: RLE8 C file	6.806
Internal bitmap format: RLE16 C file	3.740
BMP file 8bpp	4.115
BMP file 16bpp	1.134
BMP file 24bpp	1.544
BMP file 32bpp	1.525
BMP file RLE4	6.998
BMP file RLE8	6.345
GIF file	1.285
JPEG file, gray	0.516
JPEG file, gray, progressive	0.438
JPEG file, H1V1	0.402
JPEG file, H1V1, progressive	0.280
JPEG file, H2V2	0.602
JPEG file, H2V2, progressive	0.431

Table 37.2: Image drawing performance

37.2 Memory requirements

The operation area of emWin varies widely, depending primarily on the application and features used. In the following sections, memory requirements of different modules are listed as well as memory requirement of example applications. The memory requirements of the GUI components have been measured on a system as follows:

ARM7, IAR Embedded Workbench V4.42A, Thumb mode, Size optimization

37.2.1 Memory requirements of the GUI components

The following table shows the memory requirements of the main components of emWin. These values depend a lot on the compiler options, the compiler version and the used CPU. Note that the listed values are the requirements of the basic functions of each module and that there are several additional functions available which have not been considered in the table:

Component	ROM	RAM	Description
Window Manager	+ 6.2 Kbytes	+ 2.5 Kbyte	Additional memory requirements of a 'Hello world' application when using the Window Manager.
Memory Devices	+ 4.7 Kbytes	+ 7 Kbytes	Additional memory requirements of a 'Hello world' application when using Memory Devices.
Antialiasing	+ 4.5 Kbytes	+ 2 * LCD_XSIZE	Additional memory requirements for the antialiasing software item.
Driver	+ 2 - 8 Kbytes	20 Bytes	The memory requirements of the driver depend on the configured driver and if a data cache is used or not. With a data cache, the driver requires more RAM. For details, refer to the chapter "Display drivers" on page 1055.
MultiLayer	+ 2 - 8 Kbytes	-	If working with a MultiLayer or a MultiDisplay configuration additional memory for each additional layer is required, because each layer requires its own driver.
Core	5.2 Kbytes	80 Bytes	Memory requirements of a typical 'Hello world' application without using additional software items.
Core / JPEG	12 Kbytes	38 Kbytes	Basic routines for drawing JPEG files.
Core / GIF	3.3 Kbytes	17 Kbytes	Basic routines for drawing GIF files.
Core / Sprites	4.7 Kbytes	16 Bytes	Routines for drawing sprites and cursors.
Core / Fonts	(see description)	-	Details of the ROM requirements of the standard fonts shipped with emWin can be found in the chapter "Fonts" on page 207.
Widgets	4.5 Kbytes	-	This is the approximately basic ROM requirement for the widgets depending on the individual core functions used by the widgets.
Widget / BUTTON	1 Kbytes	40 Bytes	*1
Widget / CHECKBOX	1 Kbytes	52 Bytes	*1
Widget / DROPDOWN	1.8 Kbytes	52 Bytes	*1
Widget / EDIT	2.2 Kbytes	28 Bytes	*1
Widget / FRAMEWIN	2.2 Kbytes	12 Bytes	*1
Widget / GRAPH	2.9 Kbytes	48 Bytes	*1
Widget / GRAPH_DATA_XY	0.7 Kbytes	-	*1
Widget / GRAPH_DATA_YT	0.6 Kbytes	-	*1
Widget / HEADER	2.8 Kbytes	32 Bytes	*1
Widget / LISTBOX	3.7 Kbytes	56 Bytes	*1
Widget / LISTVIEW	3.6 Kbytes	44 Bytes	*1
Widget / MENU	5.7 Kbytes	52 Bytes	*1
Widget / MULTIEDIT	7.1 Kbytes	16 Bytes	*1
Widget / MULTIPAGE	3.9 Kbytes	32 Bytes	*1
Widget / PROGBAR	1.3 Kbytes	20 Bytes	*1

Table 37.3: Memory requirements of the GUI components

Component	ROM	RAM	Description
Widget / RADIOBUTTON	1.4 Kbytes	32 Bytes	*1
Widget / SCROLLBAR	2 Kbytes	14 Bytes	*1
Widget / SLIDER	1.3 Kbytes	16 Bytes	*1
Widget / TEXT	0.4 Kbytes	16 Bytes	*1

Table 37.3: Memory requirements of the GUI components

*1. The listed memory requirements of the widgets contain the basic routines required for creating and drawing the widget. Depending on the specific widget there are several additional functions available which are not listed in the table.

37.2.2 Stack requirements

The basic stack requirement is app. 600 bytes. If using the Window Manager additional 600 bytes should be calculated. For Memory Devices further additional 200 bytes are recommended. Please note that the stack requirement also depends on the application, the used compiler and the CPU.

37.3 Memory requirements of example applications

This section shows the requirements of some example applications. The following table contains the summary of the memory requirements. The values are in bytes unless specified other:

Example	GUI core	Fonts	Application	Startup code	Library	Total	GUI core	Application	Stack	Total
	ROM						RAM			
Hello world	5.9 kB	1.8 kB	38 B	0.3 kB	0.1 kB	**8.1 kB**	62 B	-	272 B	**334 B**
Window application	43 kB	12.5 kB	2.7 kB	0.3 kB	1.5 kB	**60 kB**	5.2 kB	40 B	1.4 kB	**6.6 kB**

Table 37.4: Memory requirements of example applications

For details about the examples, refer to the following sections.

37.4 Optimizing Footprint

The amount of RAM and ROM required by emWin could be optimized in some cases. This chapter shows when it is possible to spare some RAM and/or ROM and how that could be achieved.

37.4.1 Optimizing RAM requirement

In general the application should not allocate much more memory as required by the application. But unfortunately it is not possible providing a simple formula which can be used for that. If the simulation is used it is possible to open a window which shows the current amount of used/free memory of the executing application by right-clicking the simulation window. For details please also refer to “View system info” on page 56.
All of the below shown RAM optimizations can be done with a precompiled library.

Systems using bitmaps with less than 256 colors

If less than 256 colors are used by bitmaps the size of the buffer required for bitmap palette conversion could be reduced. Per default palettes with up to 256 colors can be converted. That requires 256 * 4 = 1024 bytes. If the bitmaps used by the application use less than 256 colors the size of the buffer could be reduced. That could be done by calling the function `LCD_SetMaxNumColors()` with the maximum number of colors used by bitmaps. Details can be found in the description of “LCD_SetMaxNumColors()” on page 1167.

Systems using a display driver with indirect interface

If the system is short on RAM and a driver with indirect interface is used it is recommended not to use a display driver cache. If the display controller supports reading back frame buffer data it should be possible to use the driver without a cache. Details about cache configuration can be found in the respective display driver description.

Systems using multi tasking support (GUI_OS == 1)

If multiple tasks are configured emWin uses a maximum of 4 tasks per default. That requires app. 110 bytes per task which makes 4 * 110 = 440 bytes. If less than 4 GUI-tasks are used that can be done by fine tuning the maximum number of tasks by calling the function `GUITASK_SetMaxTask()` from `GUI_X_Config()`. For details please also refer to the function “GUITASK_SetMaxTask()” on page 1206.

37.4.2 Optimizing ROM requirement

In general here can not be done much on emWin side. But may some features could be disabled which spare a few KByte of ROM requirement.
The below shown optimizations can only be done when compiling the source code of emWin.

Using the Window Manager without transparent windows

If the application does not require transparent windows the source of emWin can be compiled with the following option set in the configuration file `GUIConf.h`:

```
#define WM_SUPPORT_TRANSPARENCY 0
```

Disable text rotation

If the application does not use the functions for drawing rotated text the code required for that operations can be disabled by inserting the following define in the configuration file `GUIConf.h`:

```
#define GUI_SUPPORT_ROTATION 0
```


37.4.3 Features with appreciable additional RAM requirement

The following table shows the features requiring additional RAM:

Module	Description
Language module (GUI_LANG_...)	The amount of additional RAM requirement depends on the kind of use of the language model and on the size of the text files used.
Alpha blending	If alpha blending is used the module automatically allocates 3 buffers with the maximum virtual display size in x and a color depth of 32 bpp.
Orientation device	If a driver does not support changing the display orientation the orientation device could be used. But please note that this device uses a much memory as required to hold a copy of the complete frame buffer.

Table 37.5: Features with appreciable additional RAM requirement

This table does not contain the RAM requirement of each single module but shows the most appreciable ones.

Chapter 38

Configuration

Before emWin can be used on a target system, the software needs to be configured. Configuring means modifying the configuration files which usually reside in the (sub)directory `Config`. We try to keep the configuration as simple as possible, but there are some configuration routines which need to be modified in order for the system to work properly.
The following items need to be configured:

- Memory area to be used by emWin
- Display driver to be used for drawing operations
- Color conversion routines to be used
- Display controller initialization
- Hardware accelleraion

The following chapter explains the configuration of emWin in detail.

38.1 What needs to be configured?

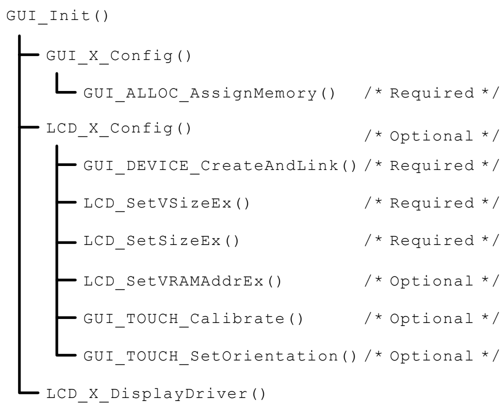
The configuration is basically divided into two parts: GUI-configuration and LCD-configuration. GUI-configuration means configuration of available features, default colors and -fonts and the configuration of available memory. The LCD-configuration is more hardware dependent and has to define the physical size of the display, the display driver and the color conversion routines to be used. For details about color conversion routines, refer to the chapter "Colors" on page 291.
If a hardware is used which offers acceleration features as for example available with the ChromeART accelerator of some of the STM32 devices, the chapter
A further part is configuring the simulation. But this is not required for the target hardware and not part of this chapter. For details about configuring the simulation, refer to the chapter "Simulation" on page 51.

38.2 Run-time- and compile-time configuration

There are C and include files to be configured. The configuration in the header files is fixed at compile time and can not be changed whereas the configuration done in the C files can be changed at run-time. This makes it possible to create a library which is largely configuration independent and can be used with any display and any driver. This requires that the configuration routines described in this chapter are not part of the library but of the application.

38.3 Initialization process of emWin

The illustration shows the process of initialization. To initialize emWin, the application only has to call GUI_Init(). The configuration routines explained below are called during the internal initialization process.



GUI_X_Config()

It is called at the very first beginning of the initialization process to make sure that memory is assigned to emWin. Within this routine `GUI_ALLOC_AssignMemory()` must be called to assign a memory block to emWin and set the average memory block size. The functions are explained later in this chapter.

LCD_X_Config()

This function is called immediately after `GUI_X_Config()`. The main purpose of this routine is creating a display driver device and selecting the color conversion routines. Further it is responsible for setting the display size. If a touch screen is used it should also be configured here.

LCD_X_DisplayDriver()

At a later point of the initialization process the function `LCD_X_DisplayDriver()` is called. It is called directly by the display driver. During the initialization process the task of this routine is putting the display controller into operation. A detailed description of the routine follows later in this chapter.

38.4 Run-time configuration

The following table shows the available run-time configuration files located in the subfolder `Config`:

Configuration file	Purpose
`GUIConf.c`	Configuration of available memory.
`LCDConf.c`	Configuration of the display size, the display driver and the color conversion routines.
`SIMConf.c`	Configuration of the simulation (not part of this chapter).
`GUI_X.c`	Configuration of timing routines.

Table 38.1: Configuration files

38.4.1 Customizing GUIConf.c

The purpose of this module is to provide emWin with the function `GUI_X_Config()` which is responsible for assigning a memory block to the memory management system. This requires knowledge about the memory requirement of the used components. The separate chapter 'Performance and Resource Usage' contains a detailed description of the memory requirements (RAM and ROM) of the individual emWin modules.
Per default `GUIConf.c` is located in the (sub)directory `Config` and contains the routine `GUI_X_Config()` which is responsible to assign a memory block to emWin. It is not cogently required to leave it in the file `GUIConf.c`. The routine `GUI_X_Config()` can be located anywhere in the application.

GUI_X_Config()

Description

Calling this function is the very first thing done by the initialization process. It is responsible to assign a memory block to emWin. This block is managed by the internal memory management system. The memory block needs to be accessible 8, 16 and 32 bit wise.

Prototype

```
void GUI_X_Config(void);
```

Additional information

Note that not the complete memory block can be used by the application, because a small overhead of the memory is used by the management system itself. Each memory block requires approximately 12 bytes for management purpose.

38.4.1.1 API functions to be used in GUI_X_Config()

The following table shows the API functions which must be called within `GUI_X_Config()`:

Routine	Description
`GUI_ALLOC_AssignMemory()`	Assigns a memory block for the memory management system.
`GUI_SetOnErrorFunc()`	Sets a hook function which is called by the GUI on major errors which causes the system to stop.
`GUITASK_GetMaxTask()`	Returns the maximum number of emWin tasks.
`GUITASK_SetMaxTask()`	Sets the maximum number of tasks from which emWin can be accessed when multitasking is enabled.

Table 38.2: GUI_X_Config() API list

GUI_ALLOC_AssignMemory()

Description

The function assigns the one and only memory block to emWin which is used by the internal memory management system. This function should be called typically from `GUI_X_Config()`.

Prototype

```
void GUI_ALLOC_AssignMemory(void * p, U32 NumBytes);
```

Parameter	Description
p	Pointer to the memory block which should be used by emWin.
NumBytes	Size of the memory block in bytes.

Table 38.3: GUI_ALLOC_AssignMemory() parameter list

Additional information

The complete memory block can be used by the application, because a small overhead of the memory is used by the management system itself. The assigned memory is used by emWin to manage data required e.g. for buffers for drawing operations, for windows, for Memory Devices or for a display driver cache.
It is never used as frame buffer.

GUI_SetOnErrorFunc()

Description

Sets the hook function which is called from GUI_Error().

Prototype

```
void GUI_SetOnErrorFunc(void (* pFunc)(const char * s));
```

Parameter	Description
pFunc	Pointer to the function which should be called by GUI_Error().

Table 38.4: GUI_SetOnErrorFunc() parameter list

Additional information

The hook function gets a short error description in the string passed to the routine. It should contain the module and the function where the error occurred and a short description.
The description of the function `GUI_Error()` can be found on page 1190.

GUITASK_GetMaxTask()

Description

Returns the maximum number of possible tasks when multitasking is enabled.

Prototype

```
int GUITASK_GetMaxTask(void);
```

GUITASK_SetMaxTask()

Description

Sets the maximum number of tasks from which emWin can be accessed when multitasking is enabled.

Prototype

```
void GUITASK_SetMaxTask(int MaxTask);
```

Parameter	Description
MaxTask	Number of tasks from which emWin is used at most.

Table 38.5: GUITASK_SetMaxTask() parameter list

Additional information

This function is intended to be called from `GUI_X_Config()`. It is necessary to use this function when working with a pre-compiled library. Otherwise `GUI_MAXTASK` can be defined. Further information can be found under “GUI_MAXTASK” on page 369.

38.4.2 Customizing LCDConf.c

The purpose of this module is to provide emWin with the required display configuration routine and the callback function for the display driver. These are the following functions:

Routine	Description
LCD_X_Config()	Configuration routine for creating the display driver device, setting the color conversion routines and the display size.
LCD_X_DisplayDriver()	Callback routine called by the display driver for putting the display controller into operation.

Table 38.6: Functions to implement in LCDConf.c

LCD_X_Config()

Description

As described in the table above this routine is responsible to create a display driver device, set the right color conversion routines and for configuring the physical display size.

Prototype

```
void LCD_X_Config(void);
```

Additional information

Depending on the used display driver it could also be required to set the video RAM address, initialize a custom palette or some else. For information about any additional requirements, refer to “Detailed display driver descriptions” on page 1075. The functions available for configuration purpose in this routine are listed and explained later in this chapter.

Example

The following shows a typical example implementation:

```
//
// Set display driver and color conversion for 1st layer
//
GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, GUICC_565, 0, 0);
//
// Display driver configuration
//
LCD_SetSizeEx    (0, 320, 240);
LCD_SetVSizeEx   (0, 320, 240);
LCD_SetVRAMAddrEx(0, (void *)0x200000);
```


LCD_X_DisplayDriver()

Description

This is the callback function of the display driver. It is called for several purposes. During the process of initialization only a few are of interest, actually the display controller initialization and the setting of the video RAM address.

Prototype

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData);
```

Parameter	Description
LayerIndex	Zero based layer index.
Cmd	A detailed list of the commands which could be passed to the function can be found under "Display driver callback function" on page 1073
pData	Pointer to a data structure of a type that depends on Cmd

Table 38.7: LCD_X_DisplayDriver() parameter list

Elements of structure LCD_X_SETVRAMADDR_INFO

Data type	Element	Description
void *	pVRAM	Pointer to the start address of the video RAM.

Table 38.8: LCD_X_SETVRAMADDR_INFO element list

Return value

The routine should return -2 if an error occurs, -1 if the command is not handled by the function and 0 if the command has been successfully executed.

Additional information

For more information about the commands passed to the routine by the display driver, refer to "Display drivers" on page 1055.

Examples

The folder `Sample\LCDConf` contains a lot of example implementations of this routine which can be used as starting point.

38.4.2.1 API functions to be used in LCD_X_Config()

The following table shows the API functions which are available for configuration purpose within `LCD_X_Config()`:

Routine	Description
GUI_DEVICE_CreateAndLink()	Creates a display driver device and associates the color conversion routines to be used.
GUI_TOUCH_SetOrientation()	Sets the orientation of the touch screen. This routine is only required if a touch screen is used which does not operates in its default orientation.
GUI_TOUCH_Calibrate()	Calibrates the touch screen.
LCD_SetLUTEx()	Initializes the lookup table with the given palette. This function is required only if a custom palette has to be used. The description of this function can be found in the chapter "Colors" on page 291.
LCD_SetSizeEx()	Required to set the physical size of the display.
LCD_SetVRAMAddrEx()	Sets the address of the video RAM. It is only required if a display driver with linear mapped video RAM is used.
LCD_SetVSizeEx()	Required only if the virtual display size is different to the physical size.

Table 38.9: LCD_X_Config() API list

Aside from the function `LCD_SetLUTEx()` the descriptions of the `LCD_...()` functions can be found in the chapter "Display drivers" on page 1055.

The descriptions of the `GUI_TOUCH_...()` functions can be found in the chapter “Touch screen driver” on page 976.

GUI_DEVICE_CreateAndLink()

Description

This routine creates the display driver device, sets the color conversion routines to be used for accessing the display and it links the driver device into the device list of the given layer. `LCD_X_Config()` is called immediately after `GUI_X_Config()`. This makes sure that the memory configuration already has been done and the driver is able to allocate memory.
The required memory for a display driver device is app. 50 bytes + the driver specific memory. For details about the memory requirements of the individual display drivers, refer to the chapter “Display drivers” on page 1055.

Prototype

```
GUI_DEVICE * GUI_DEVICE_CreateAndLink(const GUI_DEVICE_API * pDeviceAPI,
                                      const LCD_API_COLOR_CONV   * pColorConvAPI,
                                      U16 Flags, int LayerIndex);
```

Parameter	Description
pDeviceAPI	Pointer to the display driver to be used. The chapter ’Display drivers’ contains a table of the available display drivers.
pColorConvAPI	Pointer to the color conversion routines to be used. The chapter ’Colors’ contains a table with the available color conversion routines.
Flags	Should be zero.
LayerIndex	Layer which should be managed by the driver.

Table 38.10: GUI_DEVICE_CreateAndLink() parameter list

Return value

On success the function returns a pointer to the created device object, otherwise it returns NULL.

Additional information

Note that the used driver also determines the display orientation in some cases. This differs from driver to driver. For details about the display orientation, refer to the chapter “Display drivers” on page 1055.

38.4.3 Customizing GUI_X.c

This file is the location of the timing routines, the debugging routines and the kernel interface routines:

38.4.3.1 Timing routines

GUI_X_Delay()

Description

Returns after a specified time period in milliseconds.

Prototype

```
void GUI_X_Delay(int Period);
```

Parameter	Description
Period	Period in milliseconds.

Table 38.11: GUI_X_Delay() parameter list

GUI_X_ExecIdle()

Description

Called only from non-blocking functions of the Window Manager.

Prototype

```
void GUI_X_ExecIdle(void);
```

Additional information

Called when there are no longer any messages which require processing. In this case the GUI is up to date.

GUI_X_GetTime()

Description

Used by `GUI_GetTime` to return the current system time in milliseconds.

Prototype

```
int GUI_X_GetTime(void)
```

Return value

The current system time in milliseconds, of type integer.

38.4.3.2 Debug routines

GUI_X_ErrorOut(), GUI_X_Warn(), GUI_X_Log()

Description

These routines are called by emWin with debug information in higher debug levels in case a problem (Error) or potential problem is discovered. The routines can be blank; they are not required for the functionality of emWin. In a target system, they are typically not required in a release (production) build, since a production build typically uses a lower debug level.
Fatal errors are output using `GUI_X_ErrorOut()` if (`GUI_DEBUG_LEVEL` >= 3)
Warnings are output using `GUI_X_Warn()` if (`GUI_DEBUG_LEVEL` >= 4)
Messages are output using `GUI_X_Log()` if (`GUI_DEBUG_LEVEL` >= 5)

Prototypes

```
void GUI_X_ErrorOut(const char * s);
void GUI_X_Warn(const char * s);
void GUI_X_Log(const char * s);
```

Parameter	Description
`s`	Pointer to the string to be sent.

Table 38.12: GUI_X_ErrorOut() / GUI_X_Warn() / GUI_X_Log() parameter list

Additional information

This routine is called by emWin to transmit error messages or warnings, and is required if logging is enabled. The GUI calls this function depending on the configuration macro GUI_DEBUG_LEVEL. The following table lists the permitted values for GUI_DEBUG_LEVEL:

Value	Symbolic name	Description
0	GUI_DEBUG_LEVEL_NOCHECK	No run-time checks are performed.
1	GUI_DEBUG_LEVEL_CHECK_PARA	Parameter checks are performed to avoid crashes. (Default for target system)
2	GUI_DEBUG_LEVEL_CHECK_ALL	Parameter checks and consistency checks are performed.
3	GUI_DEBUG_LEVEL_LOG_ERRORS	Errors are recorded.
4	GUI_DEBUG_LEVEL_LOG_WARNINGS	Errors and warnings are recorded. (Default for PC-simulation)
5	GUI_DEBUG_LEVEL_LOG_ALL	Errors, warnings and messages are recorded.

Table 38.13: GUI_DEBUG_LEVEL permitted values

38.4.3.3 Kernel interface routines

Detailed descriptions for these routines may be found in ’Execution Model: Single Task/Multitask’.

38.5 Compile time configuration

The following table shows the available compile time configuration files located in the subfolder Config:

Configuration file	Purpose
GUIConf.h	Configuration of possible number of used layers, default fonts and colors and available features (e.g. Widgets).
LCDConf.h	Configuration of the used display driver(s).

Table 38.14: Configuration files

In case a precompiled emWin library is used, changing the configuration files will not have any effect until the library is compiled again with the required settings. This applies to all of the defines explained in the following sections.

38.5.1 Customizing GUIConf.h

As described above the file should contain the configuration of available features and the configuration of the default font. Each emWin shipment comes with a GUIConf.h file which includes a basic configuration which can be used as a starting point.

38.5.1.1 Configuring the available features of emWin

The following table shows the available configuration macros:

Type	Macro	Default	Description
B	GUI_OS	0	Activate to enable multitasking support with multiple tasks calling emWin (see the chapter "Execution Model: Single Task / Multitask" on page 363.
B	GUI_SUPPORT_CURSOR	(see expl.)	Per default cursors are enabled if either GUI_SUPPORT_TOUCH or GUI_SUPPORT_MOUSE has been enabled. If cursors should be shown without enabling one of these options it should be set to 1.
B	GUI_SUPPORT_MEMDEV	0	Enables optional Memory Device support.
B	GUI_SUPPORT_MOUSE	0	Enables the optional mouse support.
B	GUI_SUPPORT_ROTATION	1	Enables text rotation support.
B	GUI_SUPPORT_SPY	0	Enables support for emWinSPY.
B	GUI_SUPPORT_TOUCH	0	Enables optional touch-screen support.
T	GUI_TIMER_TIME	int	Defines the type which is used for time values by the emWin Timer functionality.
B	GUI_WINSUPPORT	0	Enables optional Window Manager support.

Table 38.15: Feature configuration macros

38.5.1.2 Default font and default color configuration

The following table shows the available configuration macros:

Type	Macro	Default	Description
N	GUI_DEFAULT_BKCOLOR	GUI_BLACK	Define the default background color.
N	GUI_DEFAULT_COLOR	GUI_WHITE	Define the default foreground color.
S	GUI_DEFAULT_FONT	&GUI_Font6x8	Defines which font is used per default after `GUI_Init()`. If you do not use the default font, it makes sense to change to a different default, as the default font is referenced by the code and will therefore always be linked. Please also refer to `GUI_SetDefaultFont()` which can be used for runtime configuration of the default font.

Table 38.16: Font and color configuration

The default colors and fonts of the widgets which are part of the optional Window Manager can also be configured. For details, refer to the chapter "Widgets (window objects)" on page 449.

38.5.1.3 Advanced GUI configuration options

The following table shows the available configuration macros:

Type	Macro	Default	Description
S	GUI_DEBUG_LEVEL	1 (target) 4 (simulation)	Defines the debug level, which determines how many checks (assertions) are performed by emWin and if debug errors, warnings and messages are output. Higher debug levels generate bigger code.
N	GUI_MAXTASK	4	Define the maximum number of tasks from which emWin is called to access the display when multitasking support is enabled (see the chapter "Execution Model: Single Task / Multitask" on page 363.
F	GUI_MEMCPY	---	This macro allows replacement of the memcpy function.
F	GUI_MEMSET	---	Replacement of the memset function of the GUI.
N	GUI_NUM_LAYERS	1	Defines the maximum of available layers/ displays.
B	GUI_TRIAL_VERSION	0	Marks the compiler output as evaluation version.
B	GUI_WINSUPPORT	0	Enables optional Window Manager support.
N	GUI_PID_BUFFER_SIZE	5	Maximum number of PID events managed by the input buffer.
N	GUI_KEY_BUFFER_SIZE	10	Maximum number of key events managed by the input buffer.

Table 38.17: Advanced configuration

GUI_MEMCPY

This macro allows replacement of the memcpy function of the GUI. On a lot of systems, memcpy takes up a considerable amount of time because it is not optimized by the compiler manufacturer. emWin contains an alternative memcpy routine, which has been optimized for 32 bit CPUs. On a lot of systems this routine should generate faster code than the default memcpy routine. However, this is still a generic C routine, which in a lot of systems can be replaced by faster code, typically using either a different C routine, which is better optimized for the particular CPU or by writing a routine in Assembly language.
To use the optimized emWin routine add the following define to the file `GUIConf.h`:

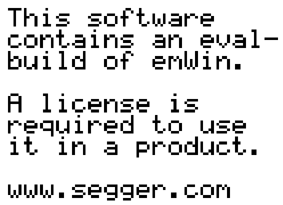

```
#define GUI_MEMCPY(pSrc, pDest, NumBytes) GUI__memcpy(pSrc, pDest, NumBytes)
```

GUI_MEMSET

This macro allows replacement of the memset function of the GUI. On a lot of systems, memset takes up a considerable amount of time because it is not optimized by the compiler manufacturer. We have tried to address this by using our own `memset()` Routine `GUI__memset`. However, this is still a generic C routine, which in a lot of systems can be replaced by faster code, typically using either a different C routine, which is better optimized for the particular CPU, by writing a routine in Assembly language or using the DMA.
If you want to use your own memset replacement routine, add the define to the `GUIConf.h` file.

GUI_TRIAL_VERSION

This macro can be used to mark the compiler output as an evaluation build. It should be defined if the software is given to a third party for evaluation purpose (typically with evaluation boards).
Note that a special license is required to do this; the most common licenses do not permit redistribution of emWin in source or object code (relinkable) form. Contact sales@segger.com if you would like to do this.
If GUI_TRIAL_VERSION is defined, the following message is shown when calling `GUI_Init()`:



This message is always shown in the upper left corner of the display and is normally visible for 1 second. The timing is implemented by a call `GUI_X_Delay(1000)`. The functionality of emWin is in no way limited if this switch is active.

Example

```
#define GUI_TRIAL_VERSION 1
```

38.5.2 Customizing LCDConf.h

This file contains general configuration options required for compiling the display driver(s) which need not to be changed at run-time. The available configuration options depend on the used display driver. For details about the available configuration options, refer to the chapter “Display drivers” on page 1055. The detailed driver description shows the available configuration options for each display driver.

38.6 Hardware acceleration

If a CPU or an LCD-controller offers hardware acceleration features, as available for example on the STM32F4-device with ChromeART accelerator, some or even most of them could be used by emWin for accelerating the process of drawing operations. To be able to achieve that different mechanisms need to be used dependent on the kind of hardware acceleration which should be used.
The following table shows a rough classification of the features which could be accelerated:

Group	Purpose
Color conversion	Some hardware like ChromeART offers the possibility of converting colors from one format to a different format. For that many of the color conversion routines offers the possibility to set a hardware function for the process of color conversion.
Fill, copy and bitmap drawing	Those operations could be accelerated by setting custom drawing functions using LCD_SetDevFunc().
Alpha blending	A custom function could be used to mix up the given foreground and background colors in accordance to the alpha values of the colors.
Mixing colors	A custom function could be used to mix up the given foreground and background colors with the given intensity.
Alpha text drawing	Drawing of antialiased text could be achieved by setting a custom function.
Palette conversion	Conversion of bitmap palettes could be done by a custom function.
Drawing bitmaps within memory devices	Some internal menory device operations could be accelerated by setting a custom function.

Table 38.18: Hardware acceleration

The following table shows the available routines:

Routine	Description
Color conversion	
GUICC_M1555I_SetCustColorConv(), GUICC_M565_SetCustColorConv(), GUICC_M4444I_SetCustColorConv(), GUICC_M888_SetCustColorConv(), GUICC_M8888I_SetCustColorConv	Setting up custom color conversion routines for the according fixed palette modes.
Filling, copy operations and bitmap drawing	
LCD_SetDevFunc()	Please refer to the function description in Chapter 33
Alpha blending	
GUI_SetFuncAlphaBlending()	Setting up a custom defined function for doing alpha blending operations
Mixing colors	
GUI_SetFuncMixColors()	Setting up a custom defined function for blending a single background color value with the given color using the given intensity.
GUI_SetFuncMixColorsBulk()	Setting up a custom defined function for bulk blending operations.
Alpha text drawing	
GUI_AA_SetpfDrawCharAA4()	Setting up a custom function for drawing 4bpp alpha characters.
Palette conversion	

Table 38.19: Window Manager API list

Routine	Description
GUI_SetFuncGetpPalConvTable()	Setting up a function for converting a palette of a bitmap into index values.
Drawing bitmaps within memory devices	
GUI_MEMDEV_SetDrawMemdev16bppFunc()	Setting up a function for drawing 16bpp bitmaps into 16bpp memory devices.

Table 38.19: Window Manager API list

38.6.1 Color conversion

As explained in detail in “Colors” on page 291, color conversion in emWin is required for converting a color value into an index value and vice versa. That is done by the routines pointed by the fixed palette mode structures. These structures have pointers for converting single items and also for bulk conversion of multiple items. The most important color conversions offers the possibility for setting custom defined routines for bulk conversion.

GUICC_M1555I_SetCustColorConv(), GUICC_M565_SetCustColorConv(), GUICC_M4444I_SetCustColorConv(), GUICC_M888_SetCustColorConv(), GUICC_M8888I_SetCustColorConv

Description

These routines can be used to set custom routines for bulk color conversion for the according color conversion.

Prototype

```
void GUICC_XXX_SetCustColorConv(
  tLCDDEV_Color2IndexBulk * pfColor2IndexBulk,
  tLCDDEV_Index2ColorBulk * pfIndex2ColorBulk);
```

Parameter	Description
pfColor2IndexBulk	Routine to be used for converting multiple colors into index values.
pfIndex2ColorBulk	Routine to be used for converting multiple index values into colors.

Table 38.20: parameter list

Additional information

The definition of tLCDDEV_Color2IndexBulk is as follows:

```
typedef void tLCDDEV_Color2IndexBulk(
  LCD_COLOR * pColor,
  void * pIndex,
  U32 NumItems,
  U8 SizeOfIndex
);
```

The definition of tLCDDEV_Index2ColorBulk is as follows:

```
typedef void tLCDDEV_Index2ColorBulk(
  void * pIndex,
  LCD_COLOR * pColor,
  U32 NumItems,
  U8 SizeOfIndex
);
```

38.6.2 Filling, copy operations and bitmap drawing

As already explained before that could be achieved by the function `LCD_SetDevFunc()` explained in Chapter 33.

38.6.3 Alpha blending

The alpha blending function of emWin is used for mixing up the given foreground colors with the given background colors by considering the alpha values in the upper bytes of the given colors. A custom defined routine could be used to achieve that operation.

GUI_SetFuncAlphaBlending()

Description

Sets a custom defined routine for alpha blending operations. That routine is called by emWin if multiple foreground colors should be mixed up with the background.

Prototype

```
void GUI_SetFuncAlphaBlending(void (* pFunc)(LCD_COLOR * pColorFG,
                                             LCD_COLOR * pColorBG,
                                             LCD_COLOR * pColorDst,
                                             U32 NumItems));
```

Parameter	Description
pFunc	Pointer to the function to be used.
pColorFG	Parameter of pFunc: Should point to the foreground color array.
pColorBG	Parameter of pFunc: Should point to the background color array.
pColorDest	Parameter of pFunc: Should point to a buffer for the result.
NumItems	Parameter of pFunc: Should contain the number of colors to be mixed up.

Table 38.21: GUI_SetFuncAlphaBlending() parameter list

Additional information

.

38.6.4 Mixing colors

Mixing up colors here means mixing up the given background with the given foreground using the given intensity.

GUI_SetFuncMixColors()

Description

Sets a custom defined routine for mixing up single colors.

Prototype

```
void GUI_SetFuncMixColors(LCD_COLOR (* pFunc)(LCD_COLOR Color,
                                              LCD_COLOR BkColor,
                                              U8 Intens));
```

Parameter	Description
pFunc	Pointer to the function to be used.
Color	Color to be blended with the given intensity.
BkColor	Color of background.
Intens	Intensity to be used for the blending operation.

Table 38.22: GUI_SetFuncMixColors() parameter list

GUI_SetFuncMixColorsBulk()

Description

Sets up a custom defined function for bulk mixing operations. That is mainly used for fading memory devices. It mixes up the given background area with the given foreground area using the desired intensity.

Prototype

```
void GUI_SetFuncMixColorsBulk(void (* pFunc)(U32 * pFG,
                                             U32 * pBG,
```



```
                                                   U32 * pDst,
                                                   unsigned OffFG,
                                                   unsigned OffBG,
                                                   unsigned OffDest,
                                                   unsigned xSize,
                                                   unsigned ySize,
                                                   U8 Intens));
```

Parameter	Description
pFunc	Pointer to the function to be used.
pFG	Parameter of pFunc: Should point to the foreground color array.
pBG	Parameter of pFunc: Should point to the background color array.
pDst	Parameter of pFunc: Should point to the destination buffer for the result.
OffFG	Parameter of pFunc: (currently not used)
OffBG	Parameter of pFunc: Additional offset in pixels (xSizeBG - xSizeFG) to be added for incrementing the background pointer at the end of a line.
OffDest	Parameter of pFunc: (currently not used)
xSize	Parameter of pFunc: xSize of area to be converted.
ySize	Parameter of pFunc: ySize of area to be converted.
Intens	Parameter of pFunc: Intensity to be used when blending the foreground over the background.

Table 38.23: GUI_SetFuncMixColorsBulk() parameter list

38.6.5 Alpha text drawing

If transparent mode is active, no memory device is selected and no clipping is required a custom function could be used for drawing antialiased characters. In all other cases automatically the default function is used by emWin. Alpha text drawing means that the given character image consists of intensity values which need to be used to mix up the current background with the current foreground color.

GUI_AA_SetpfDrawCharAA4()

Description

That function could be used to set up a custom defined function for drawing alpha blending characters with 4 bits per pixel. The intensities to be used are stored in a byte array passed to the function. Each pixel is stored in one nibble. The leftmost pixel is stored in the uppermost nibble.

Prototype

```
void GUI_AA_SetpfDrawCharAA4(int (* pfunc)(int LayerIndex,
                                            int xPos, int yPos,
                                            U8 const * p,
                                            int xSize, int ySize,
                                            int BytesPerLine));
```

Parameter	Description
pFunc	Pointer to the function to be used.
LayerIndex	Parameter of pFunc: Destination layer of drawing operation.
xPos	Parameter of pFunc: X-Position in screen coordinates to be used.

Table 38.24: GUI_AA_SetpfDrawCharAA4() parameter list

Parameter	Description
yPos	Parameter of pFunc: Y-Position in screen coordinates to be used.
p	Parameter of pFunc: Pointer to an array of bytes containing the intensity values to be used.
xSize	Parameter of pFunc: X-size of character to be drawn.
ySize	Parameter of pFunc: Y-size of character to be drawn.
BytesPerLine	Parameter of pFunc: Bytes per line of the intensity array.

Table 38.24: GUI_AA_SetpfDrawCharAA4() parameter list

38.6.6 Palette conversion

Palettes need to be converted when drawing device independent bitmaps with a color depth of not more than 8 bits per pixel. The conversion routine converts the colors of the bitmap palette into index values to be used for drawing into the frame buffer. For that conversion process, normally done by emWin, a custom function can be set.

GUI_SetFuncGetpPalConvTable()

Description

The name of the function tells something of what it exactly does. It sets a funtion pointer to a custom function, which converts an array of colors into an array of index values. It should return a pointer to the first entry of the index table.

Prototype

```
void GUI_SetFuncGetpPalConvTable(
  LCD_PIXELINDEX * (* pFunc)(
    const LCD_LOGPALETTE * pLogPal,
    const GUI_BITMAP * pBitmap,
    int LayerIndex
  )
);
```

Parameter	Description
pFunc	Pointer to the function to be used.
pLogPal	Parameter of pFunc: Pointer to the array of colors to be converted.
pBitmap	Parameter of pFunc: Pointer to the bitmap to be drawn.
LayerIndex	Parameter of pFunc: Layer index of drawing operation.

Table 38.25: GUI_SetFuncGetpPalConvTable() parameter list

Elements of structure LCD_LOGPALETTE

Data type	Element	Description
int	NumEntries	Number of color values located in the palette.
char	HasTrans	0 - No transparency should be used. 1 - Pixels with index 0 are treated as transparent.
const LCD_COLOR	pPalEntries	Pointer to the array of color values.

Table 38.26: LCD_LOGPALETTE element list

Return value

A pointer to the array of index values of type `LCD_PIXELINDEX`.

Additional information

The process of drawing a bitmap within emWin and the mechanism of color conversion should be well known when using that function.

38.6.7 Drawing bitmaps within memory devices

A custom function for drawing bitmaps with a color depth of 16bpp into memory devices with a color depth of 16bpp can be used. The task of that function is copying the data of the bitmap to be drawn into the destination memory device.

GUI_MEMDEV_SetDrawMemdev16bppFunc()

Description

Sets a custom function for the above described job.

Prototype

```
void GUI_MEMDEV_SetDrawMemdev16bppFunc(
  GUI_DRAWMEMDEV_16BPP_FUNC * pfDrawMemdev16bppFunc);
```

Parameter	Description
pfDrawMemdev16bppFunc	Pointer to the function to be used.

Table 38.27: GUI_MEMDEV_SetDrawMemdev16bppFunc() parameter list

Additional information

The definition of `GUI_DRAWMEMDEV_16BPP_FUNC` is as follows:

```
typedef void GUI_DRAWMEMDEV_16BPP_FUNC (
  void * pDst,
  const void * pSrc,
  int xSize,
  int ySize,
  int BytesPerLineDst,
  int BytesPerLineSrc
);
```

Parameter of GUI_DRAWMEMDEV_16BPP_FUNC

Parameter	Description
pDst	Destination pointer for upper left pixel to be drawn.
pSrc	Source pointer for upper left pixel to be drawn.
xSize	X-size in pixels of area to be drawn.
ySize	Y-size in pixels of area to be drawn.
BytesPerLineDst	Stride value in bytes of destination area.
BytesPerLineSrc	Stride value in bytes of source bitmap.

Table 38.28: GUI_DRAWMEMDEV_16BPP_FUNC() parameter list

38.7 Request available memory

The following functions allow control of memory usage at runtime. They can be used to e.g. prevent waste of memory.

Routine	Description
GUI_ALLOC_GetNumFreeBytes()	Returns the actual number of free bytes.
GUI_ALLOC_GetNumUsedBytes()	Returns the actual number of bytes used by the application.

Table 38.29: Memory management status API list

GUI_ALLOC_GetNumFreeBytes()

Description

This function returns the number of bytes which can be used for emWin functions.

Prototype

```
I32 GUI_ALLOC_GetNumFreeBytes(void);
```

Return value

Number of free bytes.

GUI_ALLOC_GetNumUsedBytes()

Description

This function returns the number of bytes which are already used by emWin functions.

Prototype

```
I32 GUI_ALLOC_GetNumUsedBytes(void);
```

Return value

Number of used bytes.

Chapter 39

Support

This chapter should help if any problem occurs. This could be a problem with the tool chain, with the hardware, the use of the GUI functions or with the performance and it describes how to contact emWin support.

39.1 Problems with tool chain (compiler, linker)

The following shows some of the problems that can occur with the use of your tool chain. The chapter tries to show what to do in case of a problem and how to contact the emWin support if needed.

39.1.1 Compiler crash

You ran into a tool chain (compiler) problem, not a problem of emWin. If one of the tools of your tool chain crashes, you should contact your compiler support:

```
"Tool internal error, please contact support"
```

39.1.2 Compiler warnings

The code of emWin has been tested on different target systems and with different compilers. We spend a lot of time on improving the quality of the code and we do our best to avoid compiler warnings. But the sensitivity of each compiler regarding warnings is different. So we can not avoid compiler warnings for unknown tools.

Warnings you should not see

These kinds of warnings should not occur:

```
"Function has no prototype"
"Incompatible pointer types"
"Variable used without having been initialized"
"Illegal redefinition of macro"
```

Warnings you may see

Warnings such as the ones below should be ignored:

```
"Integer conversion, may lose significant bits"
"Statement not reached"
"Descriptionless statements were deleted during op-timization"
"Condition is always true/false"
"Unreachable code"
```

Most compilers offer a way to suppress selected warnings.

Warning "Parameter not used"

Depending of the used configuration sometimes not all of the parameters of the functions are used. To avoid compiler warnings regarding this problem you can define the macro `GUI_USE_PARA` in the file `GUIConf.h` like the following example:

```
#define GUI_USE_PARA(para) (void)para
```

emWin uses this macro wherever necessary to avoid this type of warning.

39.1.3 Compiler errors

emWin assumes that the used compiler is ANSI C compatible. The compiler should cover at least one of the following standards:

- ISO/IEC/ANSI 9899:1990 (C90) with support for C++ style comments (//)
- ISO/IEC 9899:1999 (C99)
- ISO/IEC 14882:1998 (C++)

Limited number of arguments in a function pointer call

But some compilers are not 100% ANSI C compatible and have for example a limitation regarding the number of arguments in a function pointer call:

```
typedef int tFunc(int a, int b, int c, int d, int e,
                  int f, int g, int h, int i, int j);

static int _Func(int a, int b, int c, int d, int e,
                 int f, int g, int h, int i, int j) {
  return a + b + c + d + e + f + g + h;
}

static void _Test(void) {
  int Result;
  tFunc * pFunc;
  pFunc = _Func;
  Result = pFunc(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
}
```

If the example above can not be compiled, only the core version of emWin can be used. The additional packages of emWin like the Window Manager or the Memory Device module sometimes need to pass up to 10 parameters with a function pointer call. The core package of emWin needs only up to 2 parameters in a function pointer call. But you can also use emWin if your compiler only supports one argument in a function pointer call. If so some functions are not available, for example rotating text or UTF-8 encoding. For details about how to configure emWin in this case take a look at the chapter 'High-Level Configuration'.

39.1.4 Linker problems

Undefined externals

If your linker shows the error message "Undefined external symbols...", check if the following files have been included to the project or library:

- All source files shipped with emWin
- In case of a simple bus interface: One of the hardware routines located in the folder `Sample\LCD_X_Port`? For details about this, refer to the chapter "Configuration" on page 1203.
- One of the files located in the folder `Sample\GUI_X`. Details about this can be found in the chapter "Configuration" on page 1203.

Executable to large

Some linkers are not able to link only the modules/functions referenced by the project. This results is an executable with a lot of unused code. In this case the use of a library would be very helpful. For details about how to build an emWin library, refer to the chapter "Getting Started" on page 41.

39.2 Problems with hardware/driver

If your tools are working fine but your display does not work may one of the following helps to find the problem.

Stack size to low?

Make sure that there have been configured enough stack. Unfortunately we can not estimate exactly how much stack will be used by your configuration and with your compiler. Further the required stack size depends a lot on the application.

Initialization of the display wrong?

Please check if the controller initialization has been adapted to your needs.

Display interface configured wrong?

When starting to work with emWin and the display does not show something you should use an oscilloscope to measure the pins connected with the display/controller. If there is a problem, check the following:

- If using a simple bus interface: Probably the hardware routines have not been configured correctly. If possible use an emulator and step through these routines.
- If using a full bus interface: Probably the register/memory access have not been configured correctly.

39.3 Problems with API functions

If your tool chain and your hardware works fine but the API functions do not function as documented, make a small example as described in “Contacting support” on page 1227. This allows us to easily reproduce the problem and solve it quickly.

39.4 Problems with the performance

If there is any performance problem with emWin it should be determined, which part of the software causes the problem.

Does the driver causes the problem?

To determine the cause of the problem the first step should be writing a small test routine which executes some test code and measures the time used to execute this code. Starting point should be the file `ProblemReport.c` described above. To measure the time used by the real hardware driver the shipment of emWin contains the driver `LCDNull.c`. This driver can be used if no output to the hardware should be done. To activate the driver the `LCD_CONTROLLER` macro in `LCDConf.h` as follows:

```
#define LCD_CONTROLLER -2
```

The difference between the used time by the real driver and the `LCDNull` driver shows the execution time spent in the real hardware driver.

Driver not optimized?

If there is a significant difference between the use of the real driver and the `LCDNull` driver the cause of the problem could be a not optimized driver mode. If using one of the following macros: `LCD_MIRROR_X, LCD_MIRROR_Y, LCD_SWAP_XY or LCD_CACHE` the driver may not be optimized for the configured mode. In this case, contact our support, we should be able to optimize the code.

Slow display controller?

Also, refer to the chapter “Display drivers” on page 1055. If using a slow display controller like the Epson SED1335 this chapter may answer the question, why the driver works slow.

39.5 Contacting support

If you need to contact the emWin support, send the following information to the support:

- A detailed description of the problem may written as comment in the example code.
- The configuration files `GUIConf.c`, `GUIConf.h`, `LCDConf.c`, `LCDConf.h`.
- An example source file which can be compiled in the simulation without any additional files as described in the following.
- If there are any problems with the tool chain, also send the error message of the compiler/linker.
- If there are any problems with the hardware/driver and a simple bus interface is used, also send the hardware routines including the configuration.

Problem report

The following file can be used as a starting point when creating a problem report. Also fill in the CPU, the used tool chain and the problem description. It can be found under `Sample\Tutorial\ProblemReport.c`:

```
/*********************************************************************
*                 SEGGER Microcontroller GmbH & Co. KG               *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                       emWin problem report                         *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File                 : ProblemReport.c
CPU                  :
Compiler/Tool chain  :
Problem description  :
----------------------------------------------------------------------
*/

#include "GUI.h"
/*  Add further GUI header files here as required. */


/*******************************************************************
*
*       Static code
*
********************************************************************
*
* Please insert helper functions here if required.
*/

/*******************************************************************
*
*       MainTask
*/
void MainTask(void) {
  GUI_Init();
  /*
    To do:  Insert the code here which demonstrates the problem.
  */
  while (1);   /* Make sure program does not terminate */
}
```


39.6 FAQ's

Q: I use a different LCD controller. Can I still use emWin?
A: Yes. The hardware access is done in the driver module and is completely independent of the rest of the GUI. The appropriate driver can be easily written for any controller (memory-mapped or bus-driven). Please get in touch with us.

Q: Which CPUs can I use emWin with?
A: emWin can be used with any CPU (or MPU) for which a C compiler exists. Of course, it will work faster on 16/32-bit CPUs than on 8-bit CPUs.

Q: Is emWin flexible enough to do what I want to do in my application?
A: emWin should be flexible enough for any application. If for some reason you do not think it is in your case, please contact us. Believe it or not, the source code is available.

Q: Does emWin work in a multitask environment?
A: Yes, it has been designed with multitask kernels in mind.

Index

C

E

G

H

I

X

Y

Z

