

emWin

**Graphic Library with
Graphical User Interface**

Version 3.96

Manual Rev. 0



www.segger.com
solutions for embedded software

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER MICROCONTROLLER SYSTEME GmbH (the manufacturer) assumes no responsibility for any errors or omissions. The manufacturer makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. The manufacturer specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of the manufacturer. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2005 SEGGER Microcontroller Systeme GmbH, Hilden / Germany

Trademarks

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

Registration

Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available.

For registration, please provide the following:

- Company name and address
- Your name
- Your job title
- Your email address and telephone number
- Name and version of the product

Please send this information to: register@segger.com

Contact address

SEGGER Microcontroller Systeme GmbH
Heinrich-Hertz-Str. 4
D-40721 Hilden
Germany
Tel.+49 2103-2878-0
Fax.+49 2103-2878-28
Email: support@segger.com
Internet: <http://www.segger.com>

Manual versions

This manual describes the latest software version. The version number of the software can be found in the table 'Software versions' later in this chapter. If any error occurs, please inform us and we will try to assist you as soon as possible.

For further information on topics or routines not yet specified, please contact us.

Print date: 7/20/05

Manual version	Date	By	Explanation
3.96R0	050719	JE	<p>Chapter 7: 2-D Graphic Library - New function GUI_GetDrawMode() added.</p> <p>Chapter 8: Displaying bitmap files - Mentioned, that JPEG package is only part of color version.</p> <p>Chapter 9: Fonts - Antialiased SIF fonts added.</p> <p>Chapter 12: Memory devices - New function GUI_MEMDEV_WriteEx() added. - New function GUI_MEMDEV_WriteExAt() added.</p> <p>Chapter 14: Window manager - New function WM_GetCallback() added.</p> <p>Chapter 15: Widgets - Interface of SCROLLBAR_SetColor() changed. - Explanation added, how to determine the type of a widget. - New GRAPH widget added. - New function BUTTON_GetBitmap() added. - New function CHECKBOX_GetText() added. - New function DROPDOWN_SetColor() added. - New function DROPDOWN_SetDefaultColor() added. - New function DROPDOWN_SetDefaultScrollbarColor() added. - New function DROPDOWN_SetScrollbarColor() added. - New function LISTBOX_SetScrollbarColor() added. - New function RADIO_GetText() added.</p> <p>Chapter 24: Display drivers - LCD667XX driver: Support for Samsung S6D0110A added. - LCD07X1 driver: Support for Sitronix ST7541 added. - LCD66750 driver: Support for Hitachi HD66753 added.</p>
3.95R0	050701	JE	<p>Chapter 23 (Unicode) and 24 (Shift JIS) merged to chapter 23 (Foreign Language Support).</p> <p>Chapter 23: - Arabic support added.</p>

Manual version	Date	By	Explanation
3.94R0	050329	JE	<p>New chapter 3: Viewer - Virtual screen support added. Chapter 7: 2-D Graphic Library - New function GUI_SetClipRect() added. Chapter 8: Displaying bitmap files - GIF file support added. Chapter 10: Bitmap converter - GIF file support added. Chapter 11: Colors - Mentioned that custom palettes only available for modes up to 8bpp. Chapter 12: Memory devices - Reworked and support for 1bpp memory devices added. Chapter 14: Window manager - API function classification basic/advanced removed. - New function WM_SetWindowPos() added. - New function WM_GetScrollPosH() added. - New function WM_GetScrollPosV() added. - New function WM_SetScrollPosH() added. - New function WM_SetScrollPosV() added. Chapter 15: Widgets - BUTTON: New function BUTTON_SetDefaultFocusColor. - BUTTON: New function BUTTON_SetFocusColor. - BUTTON: New config option BUTTON_FOCUSCOLOR_DEFAULT. - CHECKBOX: New function CHECKBOX_SetBoxBkColor(). - CHECKBOX: New function CHECKBOX_SetDefaultFocusColor(). - CHECKBOX: New function CHECKBOX_SetFocusColor(). - CHECKBOX: Notification WM_NOTIFICATION_VALUE_CHANGED added. - CHECKBOX: Image of 'unchecked' state can now be set. - EDIT: New function EDIT_SetTextMode() added. - LISTBOX: New function LISTBOX_SetDefaultTextAlign() added. - LISTBOX: New function LISTBOX_SetTextAlign() added. - LISTBOX: New function LISTBOX_SetDefaultTextAlign() added. - LISTBOX: New function LISTBOX_SetTextAlign() added. - LISTBOX: New color index LISTBOX_CI_DISABLED added. - LISTVIEW: New color index LISTVIEW_CI_DISABLED added. - MENU: Keyboard support added. - PROGBAR: Vertical progress bar supported. - RADIO: New function RADIO_SetDefaultFocusColor() added. - RADIO: New function RADIO_SetFocusColor() added. - RADIO: Config options reworked. - SCROLLBAR: New function SCROLLBAR_SetColor() added. - SCROLLBAR: New function SCROLLBAR_SetDefaultColor() added. - SLIDER: New function SLIDER_SetDefaultFocusColor() added. - SLIDER: New function SLIDER_SetFocusColor() added. - Keyboard support explained for each widget. - Widget callback functions added. Chapter 22: Antialiasing - New function GUI_AA_DrawPolyOutlineEx() added. - Limitation of GUI_AA_DrawPolyOutline() explained. Chapter 23: Unicode - New function GUI_UC_ConvertUC2UTF8() added. - New function GUI_UC_ConvertUTF82UC() added. Chapter 25: Display drivers - New driver LCD0323 for Solomon SSD0323 controller added. - New driver LCD1200 for Toppoly C0C0 and C0E0 controller added. - New driver LCD13701 for Epson S1D13701 controller added. - New driver LCDNoritake for Noritake displays added. - New driver LCD667XX for Hitachi HD66772, HD66766 and HD66789 controller. - New driver LCDXylon added. - LCDLin driver: New config macro LCD_FILL_RECT added. - LCDLin driver: New config macro LCD_LIN_SWAP added. - LCDLin driver: 32 bit version now supports 1 and 2 bpp color depth. Chapter 28: Low level config - LCD_REVERSE_LUT added.</p>

Manual version	Date	By	Explanation
3.92R0	041021	JE	<p>Chapter 3: Simulator and Viewer - Subchapter added: Using the source code of the simulator.</p> <p>Chapter 9: Bitmap converter - Saving of bitmaps reworked.</p> <p>Chapter 13: Window manager - New functions added: WM_PaintWindowAndDescs(), WM_Update() and WM_UpdateWindowAndDescs().</p> <p>Chapter 14: Widgets - Flag GUI_MESSAGEBOX_CF_MODAL added for creating a modal message box.</p> <ul style="list-style-type: none"> - Notification codes added to MULTIPAGE widget. - New functions added: LISTVIEW_DisableRow(), LISTVIEW_EnableRow() and LISTVIEW_GetItemText(). <p>New Chapter 16: Virtual screens - New functions GUI_GetOrg() and GUI_SetOrg().</p> <p>Chapter 24: Display driver - Support for Sitronix ST7565 added to Page1bpp driver.</p> <ul style="list-style-type: none"> - LCDLin driver (8/16 bit): LCD_SET_LUT_ENTRY added. - LCDLin driver (32 bit): Explanation added how to migrate from LCDLin to LCDLin32. <p>Chapter 27: High-Level Configuration - GUI_TRIAL_VERSION added.</p>
3.90R0	040818	JE	<p>Chapter 6: 2-D Graphic library - GUI_DrawBitmapEx() does not render RLE-compressed bitmaps.</p> <p>Chapter 13: Window manager - Explanation added, which messages are not send to disabled windows.</p> <ul style="list-style-type: none"> - New function WM_GetPrevSibling() added. - New message WM_MOUSEOVER added. - New create flag WM_CF_MEMDEV_ON_REDRAW added. - WM_MOVE message: Explanation reworked. <p>Chapter 14: Widgets - MENU widget added.</p> <ul style="list-style-type: none"> - WM_NOTIFICATION_SCROLL_CHANGED added to DROPODOWN, LISTBOX, LISTVIEW and MULTIEDIT - BUTTON widget: Support for disabled state added. - New functions added: DROPODOWN_SetTextHeight(), DROPODOWN_SetTextAlign(), FRAMEWIN_AddMenu(), FRAMEWIN_SetResizeable(), LISTVIEW_GetBkColor(), LISTVIEW_GetTextColor(), LISTVIEW_SetItemTextColor(), LISTVIEW_SetItemBkColor() <p>Chapter 17: Pointer Input Devices - Explanation of structure GUI_PID_STATE changed.</p> <p>Chapter 23: Display driver - Support for Solomon SSD1815 added to Page1bpp driver</p> <p>Chapter 26: Low-Level Configuration - 'Real bus interface' renamed to 'Memory mapped interface'</p> <p>Chapter 29: Support - Warnings added to subchapter 'compiler warnings'. - Subchapter 'Contacting support' added.'</p>

Manual version	Date	By	Explanation
3.82R0	040714	JE	<p>Chapter 1: Introduction: - ANSI standard described more detailed.</p> <p>Chapter 6: 2-D Graphic Library - Default limitation for drawing bitmaps with 16 bit CPUs added.</p> <p>Chapter 8: Fonts - Fonts added: GUI_FontD48, GUI_FontD64, GUI_FontD80, GUI_FontD37x48, GUI_FontD48x64, GUI_FontD61x80</p> <p>Chapter 11: Memory devices - Subchapter 'Memory requirements' and 'Performance' added.</p> <p>Chapter 13: Window manager - Functions added: WM_IsCompletelyVisible(), WM_IsEnabled()</p> <p>- Message added: WM_NOTIFY_VIS_CHANGED</p> <p>- Config macro added: WM_SUPPORT_NOTIFY_VIS_CHANGED</p> <p>Chapter 14: Widgets - WM... functions moved to Chapter 13 - Functions added: BUTTON_GetDefaultTextAlign(), BUTTON_GetDefaultBkColor(), BUTTON_GetDefaultFont(), BUTTON_GetDefaultTextColor(), BUTTON_SetDefaultTextAlign(), BUTTON_SetDefaultBkColor(), BUTTON_SetDefaultFont(), BUTTON_SetDefaultTextColor(), CHECKBOX_GetDefaultBkColor(), CHECKBOX_GetDefaultFont(), CHECKBOX_GetDefaultSpacing(), CHECKBOX_GetDefaultTextAlign(), CHECKBOX_GetDefaultTextColor(), CHECKBOX_GetState(), CHECKBOX_SetBkColor(), CHECKBOX_SetDefaultBkColor(), CHECKBOX_SetDefaultFont(), CHECKBOX_SetDefaultSpacing(), CHECKBOX_SetDefaultTextAlign(), CHECKBOX_SetDefaultTextColor(), CHECKBOX_SetFont(), CHECKBOX_SetNumStates(), CHECKBOX_SetSpacing(), CHECKBOX_SetState(), CHECKBOX_SetText(), CHECKBOX_SetTextAlign(), CHECKBOX_SetTextColor(), LISTBOX_GetDefaultBkColor(), LISTBOX_GetDefaultTextColor(), LISTBOX_SetDefaultBkColor(), LISTBOX_SetDefaultTextColor() - Config macros added: BUTTON_ALIGN_DEFAULT, BUTTON.REACT_ON_LEVEL, CHECKBOX_BKCOLOR_DEFAULT, CHECKBOX_FONT_DEFAULT, CHECKBOX_IMAGE0_DEFAULT, CHECKBOX_IMAGE1_DEFAULT, CHECKBOX_SPACING_DEFAULT, CHECKBOX_TEXTALIGN_DEFAULT, CHECKBOX_TEXTCOLOR_DEFAULT</p> <p>Chapter 15: Dialogs - Explanation of 'blocking' and 'non blocking' improved.</p> <p>Chapter 23: Display drivers - LCDLin and LCDLin32 merged to one driver. - Support for S1D13A03, S1D13A04, S1D13A05 and S1D15710 added.</p> <p>Chapter 27: High level configuration - GUI_MAXBLOCKS and GUI_SUPPORT_LARGE_BITMAPS added.</p>
3.80R0	040503	JE	<p>Chapter 3: Simulator & Viewer - SIM_SetMag added.</p> <p>Chapter 13: Window manager - Sample added to WM_PID_STATE_CHANGED message. - WM_SET_FOCUS added - Config subchapter added (WM_SUPPORT_TRANSPARENCY) - WM_SetTransState() added.</p> <p>Chapter 14: Widgets - New functions added: EDIT_GetNumChars(), EDIT_SetSel(), EDIT_SetpfAddKeyEx(), EDIT_SetInsertMode(), EDIT_SetFloatValue(), RADIO_GetDefaultFont(), RADIO_SetDefaultTextColor(), RADIO_SetBkColor(), RADIO_SetDefaultFont(), RADIO_SetDefaultTextColor(), RADIO_SetFont(), RADIO_SetGroupID(), RADIO_SetText(), RADIO_SetTextColor(), SLIDER_SetNumTicks(). - Explanation added how to use snapping with the slider widget. - Information added to CHECKBOX_CreateEx(). - Description of RADIO_CreateEx() reworked.</p> <p>Chapter 23: Display drivers: - New driver LCDPage4bpp added (support for Sitronix ST7528) - New driver LCD1781 added (support for Solomon SSD1781). - Support for Epson S1D13700 added to LCDSlin driver. - Support for Novatec NT7502, Samsung S6B1713 and Philips PCD8544 added to LCDPage1bpp. - Explanation how to use hardware for display orientation added to LCD07X1 and LCDPage1bpp.</p> <p>Chapter 28: LCD driver API - Chapter moved as subchapter to chapter 23.</p>

Manual version	Date	By	Explanation
3.76R0	040123	JE	<p>Chapter 12: Execution model - GUI_X_WAIT_EVENT and GUI_X_SIGNAL_EVENT added</p> <p>Chapter 13: Window manager - Message WM_INIT_DIALOG added.</p> <p>Chapter 14: Widgets - Notification codes added to SLIDER and SCROLLBAR - New functions added: EDIT_SetDefaultTextColor(), EDIT_SetDefaultBkColor(), EDIT_GetDefaultTextColor(), EDIT_GetDefaultBkColor(), EDIT_SetDefaultTextAlign(), DROPDOWN_SetScrollbarWidth() and LISTBOX_SetScrollbarWidth(). - WINDOW widget added. - MESSAGEBOX widget added. - MULTIPAGE widget added. - Creation flag DROPDOWN_CF_UP added. - Effect functions added.</p> <p>Chapter 17: Pointer input devices - From an interrupt routine callable functions denoted.</p> <p>Chapter 18: Keyboard input - From an interrupt routine callable functions denoted.</p> <p>Chapter 27: High level configuraton - Limitation for 16 bit CPU's removed.</p> <p>Chapter 29: Performance and resource usage - Memory requirements reworked.</p>
3.74R0	031219	JE	<p>Chapter 10: Colors: - New predefined color table (colors added)</p> <p>Chapter 11: Memory devices: - New function GUI_MEMDEV_CreateFixed() added.</p> <p>Chapter 13: Window manager: - Message WM_GET_ID added. - Explanation of window invalidation added. - Creation flags added: WM_CF_ANCHOR_LEFT, WM_CF_ANCHOR_TOP, WM_CF_ANCHOR_RIGHT and WM_CF_ANCHOR_BOTTOM.</p> <p>Chapter 14: Widgets: - Screenshots added to overview and FRAMEWIN widget. - FRAMEWIN_CreateButton-functions renamed to FRAMEWIN_AddButton... - TEXT_SetBkColor() and RADIO_SetBkColor() added.</p> <p>Chapter 17: Pointer input devices: - Explanation of runtime calibration added</p> <p>Chapter 26: Low-Level configuration: - Renamed to 'Low-Level configuration (LCDConf.h)'.</p> <p>Chapter 27: High-Level configuration: - Renamed to 'High-Level configuration (GUIConf.h)'. - Explanation of GUI_ALLOC_SIZE added.</p>

Manual version	Date	By	Explanation
3.72R0	031204	JE	<p>Chapter 2: Getting started: - Explanation added how to adapt the library batch files</p> <p>Chapter 4: Displaying text: - GUI_DispStringInRectEx added.</p> <p>Chapter 13: Window Manager: - Message documentation reworked. - Functions WM_SetID and WM_ForEachDesc added. - Explanation added how to react on a WM_PAINT message.</p> <p>Chapter 8: Fonts: - System independent font support added. - Standard fonts moved to this chapter.</p> <p>Chapter 14: Widget library: - BUTTON_SetTextAlign added. - DROPODOWN_SetItemSpacing added. - EDIT_GetFloatValue added. - EDIT_SetFloatMode added. - LISTBOX_SetItemSpacing added. - LISTVIEW_SetLBorder added. - LISTVIEW_SetRBorder added. - LISTVIEW_SetRowHeight added. - RADIO_SetBkColor added. - SLIDER_SetBkColor added.</p> <p>Chapter 17: Pointer input devices: - Devided into pointer input devices and keyboard input - Pointer input device documentation reworked</p> <p>Chapter 21: Unicode: - How to convert UTF-8 text to 'C' strings added.</p> <p>Chapter 23: Display driver: - Support for UltraChip UC1611 and Hitachi HD66750 added.</p> <p>Chapter 30: Support: - Reworked and renamed to Support.</p>
3.70R0	031010	JE	<p>Chapter 3: Viewer documentation improved.</p> <p>Chapter 10: Fixed palette modes and color index mask added.</p> <p>Chapter 22: Support for the following LCD controllers added: Fujitsu MB86290A (Crescent), Fujitsu MB86291 (Scarlet), Fujitsu MB86292 (Orchid), Fujitsu MB86293 (Coral Q), Fujitsu MB86294 (Coral B), Fujitsu MB86295 (Coral P), Samsung S6B33B1X and UltraChip UC1606.</p> <p>Chapter 22: Driver LCD180S1 renamed to LCD161620.</p> <p>Chapter 22: Hardware interface descriptions added.</p> <p>Chapter 16: Multi layer sample removed.</p> <p>Chapter 25: Configuration of SPI 3, SPI 4 and I2C bus interface added.</p> <p>Chapter 27: Keil 8051 compiler limitation added</p>
3.62R1	030916	JE	Chapter 23: Explanation of config options and GUI_VNC_X_StartServer changed.
3.62R0	030901	JE	<p>Chapter 13: Message data explained.</p> <p>Chapter 14: <WIDGET>_CreateEx added, <WIDGET>_Create and <WIDGET>_CreateAsChild obsolete.</p> <p>Chapter 14: Interface from some functions changed from int to unsigned int.</p> <p>Chapter 14.4: Functions added: CHECKBOX_SetDefaultImage, CHECKBOX_SetImage</p> <p>Chapter 14.5: Functions added: DROPODOWN_Collapse, DROPOWN_DeleteItem, DROPOWN_Expand, DROPOWN_InsertString, DROPOWN_SetAutoScroll</p> <p>Chapter 14.13: Functions added: RADIO_SetDefaultImage, RADIO_SetImage</p> <p>Chapter 23 added: VNC support.</p> <p>Chapter 24: SED1520-support added, new driver for Fujitsu MB87J2020 and MB87J2120 added.</p>
3.60R2	030718	RS	Chapter 14: Explanation of user draw function improved.
3.60R1	030716	JE	<p>Chapter 14.3: BUTTON_IsPressed() added</p> <p>Chapter 14: WIDGET_ITEM_GET_YSIZE added to structure WIDGET_ITEM_DRAW_INFO to enable items with different y sizes.</p>
3.60R0	030714	RS	<p>Chapter 11: GUI_MEMDEV_GetDataPtr() added</p> <p>Chapter 13: WM_CF_CONST_OUTLINE added</p>
3.58R1	030710	RS	Chapter 24.4: Usage of display orientation macros explained.
3.58R0	030701	JE	Chapter 22: LCD501 added.

Manual version	Date	By	Explanation
3.56R0	030626	JE	Chapter 22: LCD180S1 and LCD444 added. Chapter 22: LCD_CONTROLLER macro definition changed for several LCD controllers. Chapter 13: WM_MakeModal added. Chapter 25: AVR support added.
3.56R0	030626	JE	Chapter 22: LCD180S1 and LCD444 added. Chapter 22: LCD_CONTROLLER macro definition changed for several LCD controllers. Chapter 13: WM_MakeModal added. Chapter 25: AVR support added.
3.54R0	030603	JE	New chapter 7, "Displaying bitmap files" added.
3.52R0	030516	JE	Chapter 12: WM_GetFocussedWindow added Chapter 13: LISTVIEW_DeleteRow and LISTVIEW_DeleteColumn added, LISTVIEW_GetNumColumns renamed to LISTVIEW_GetNumColumns
3.50R0	030515	JE	Chapter 4: GUI_SetTextStyle added. Chapter 6: GUI_SetLineStyle, GUI_GetLineStyle added. Chapter 12: WM_BroadcastMessage added. Chapter 13: MULTIEDIT_SetPasswordMode and MULTIEDIT_GetTextSize added.
3.48R0	030415	JE	Chapter 12: WM_SetStayOnTop and WM_GetStayOnTop added. Chapter 13.7: FRAMEWIN_SetBorderSize added. Chapter 13.11: MULTIEDIT_SetMaxNumChars added. Chapter 14: Multi layer support revised.
3.46R0	030404	JE	New chapter Multi layer support added. Colors: 86661 mode added.
3.44R0	030319	JE	Chapter 18: Revised, UTF-8 support added. Chapter 13.9: LISTBOX_GetFont, LISTBOX_GetItemText, LISTBOX_GetMulti and LISTBOX_SetOwnerDraw added.w Chapter 13: New chapter 13.11 for MULTIEDIT widget added.
3.42R1	030303	JE	Chapter 6: GUI_SaveContext and GUI_RestoreContext added.
3.42R0	030227	JE	Chapter 12: WM_GetInvalidRect and WM_HasFocus added. Chapter 13.7: FRAMEWIN_SetTextColorEx, FRAMEWIN_GetDefaultTextColor and FRAMEWIN_SetDefaultTextColor added. Chapter 13.9: LISTBOX_GetItemDisabled, LISTBOX_SetScrollStepH, LISTBOX_SetItemDisabled, LISTBOX_SetItemSel, LISTBOX_SetScrollStepH, LISTBOX_SetDefaultScrollStepH and LISTBOX_SetDefaultScrollStepH added. Chapter 13.15: TEXT_SetTextColor and TEXT_SetDefaultTextColor added.
3.40R0	030217	JE	Chapter 3: Directory structure changed. Chapter 13: Functions added.
3.38R0	030206	JE	Chapter 2: Color and gray scale conversion functions marked as optional. Chapter 13.3: Functions added. Chapter 13.9: Functions added. Chapter 13.13: Functions added.
3.36R0	030121	JE	Chapter 12: Additional functions added. Chapter 13.7: Additional functions added. Chapter 23: Dynamic memory configuration added.
3.34R2	021212	JE	Chapter 12: WM_GetClientRectEx and WM_GetWindowRectEx added. Chapter 13: WM_GetInsideRect and WM_GetInsideRectEx added.
3.34R1	021127	JE	Chapter 4: Additional functions added. Chapter 6: Additional functions added. Chapter 10: Additional functions added. Chapter 12: Additional functions added. Chapter 13: Additional functions added.
3.34R0	020926	KG	Additional functions added to section 13.5 (Edit widget).
3.32R0	020920	KG	Additional function added to Chapter 12 (The Window Manager). BMP file support functions added to Chapter 6 (2-D Graphic Library).
3.30R1	020919	KG	Chapter 13 (Window Objects): additional functions documented; message notification codes added. Chapter 12 (The Window Manager) restructured slightly; WM_NOTIFY_PARENT documented.
3.30R0	020912	KG	Chapter 18 (Input Devices) moved to after Chapter 14 (Dialogs).
	020910	KG	Chapter 16 (Cursors) added. Additional function added to section 13.8 (Progress bar widget).

Manual version	Date	By	Explanation
3.28R1	020905 020903	KG KG RS	Additional controller supported by LCD15XX in Chapter 19 (LCD Drivers). Additional function added to Chapter 18 (Input Devices). EDIT widget: add. info on flags added.
3.28R0	020830 020827 020823	KG KG KG	Additional functions added to Chapter 3 (Simulator). Starter kit "Berni" added to section 1.7 (Starter kits). Additional functions added to section 13.7 (List box widget). Headline-level format changes throughout. Modified trial version screen shots in Chapter 3 (Simulator).
3.26R0	020820	KG	Chapter 18 (Touch-Screen Support) changed to Input Devices; mouse and keyboard support added; chapter restructured. Slight modifications to section 13.10 (Scroll bar widget).
3.24R4	020809	KG	Additional macros added to Chapter 18 (Touch-Screen Support). Chapter 3 (Simulator) modified; addition of use of simulator with trial version of emWin. Section 1.7 (Data types) revised.
3.24R3	020802	KG	Additional macros added to sections 22.8 (LCDMem) and 22.9 (LCD-MemC); same macros added to Chapter 20 (Low-Level Configuration).
3.24R2	020801	KG	Section 2.3 (Creating a library) revised, table and diagram added.
3.24R1	020730	KG	Minor changes throughout, including addition of () brackets to all API functions.
3.24R0	020726	KG	Chapter 9 (Colors) revised; modes 1, 2, and 444 added. Chapter 11 (Execution Model: Single Task/Multitask) added. Chapter 1 (Introduction to emWin) revised. Chapter 2 (Getting Started) revised. Chapter 18 (Time-Related Functions) changed to Timing and Execution-Related Functions; GUI_Exec() and GUI_Exec1() added. Small formatting changes throughout.
3.22R1	020723 020719	KG RS	Chapter 18 (emWin in Multitasking Environments) merged with Chapter 22 (High-Level Configuration). Chapter 4 (Tutorial) merged with Chapter 2 (Getting Started). Chapter 10 (Colors) revised, color mode table added. GUI_X_ explanations added. Widget description enhanced, screen shots added.
3.22R0	020716	KG	Chapter 13 (Window Objects) revised; SCROLLBAR and RADIO widgets added.
3.20R0	020627 020620 020618	JE KG KG	Chapter 2 (Getting Started) revised. Chapter 14 (Dialogs) revised. Chapter 13 (Window Objects) revised; CHECKBOX, SLIDER and TEXT widgets added. Chapter 14 (Dialogs) added.
3.14R3	020618 020611 020531 020524 020507	KG KG KG KG KG	Chapter 3 (Simulator) revised. Chapter 20 (Low-Level Configuration) revised. Chapter 22 (LCD Drivers) revised. Chapter 12 (The Window Manager) revised. Version control table added.
3.14R2	020503	KG	Chapter 11 (Memory Devices) revised. Chapter 14 (Antialiasing) revised. Chapter 9 (Bitmap Converter) revised.
3.14R1	020405	KG	Completely revised for language/grammar. Section 1.5 (Typographic conventions) updated. Chapter 8 changed to 7.6 (Font converter). Index revised.

Software versions

Software version	Date	By	Explanation
3.96	050719		<p>2-D Graphic Library - New function GUI_GetDrawMode() added.</p> <p>Fonts - Antialiased SIF fonts added.</p> <p>Memory devices - New function GUI_MEMDEV_WriteEx() added. - New function GUI_MEMDEV_WriteExAt() added.</p> <p>Window manager - New function WM_GetCallback() added.</p> <p>Widgets - Interface of SCROLLBAR_SetColor() changed. - New GRAPH widget added. - New function BUTTON_GetBitmap() added. - New function CHECKBOX_GetText() added. - New function DROPODOWN_SetColor() added. - New function DROPODOWN_SetDefaultColor() added. - New function DROPODOWN_SetDefaultScrollbarColor() added. - New function DROPODOWN_SetScrollbarColor() added. - New function LISTBOX_SetScrollbarColor() added. - New function RADIO_GetText() added.</p> <p>Foreign languages - Arabic support added.</p> <p>Display drivers - LCD667XX driver: Support for Samsung S6D0110A added. - LCD07X1 driver: Support for Sitronix ST7541 added. - LCD66750 driver: Support for Hitachi HD66753 added.</p>

Software version	Date	By	Explanation
3.94	050329	JE	<p>Viewer</p> <ul style="list-style-type: none"> - Virtual screen support added. <p>2-D Graphic Library</p> <ul style="list-style-type: none"> - New function GUI_SetClipRect() added. <p>Displaying bitmap files</p> <ul style="list-style-type: none"> - GIF file support added. <p>Bitmap converter</p> <ul style="list-style-type: none"> - GIF file support added. <p>Memory devices</p> <ul style="list-style-type: none"> - Support for 1bpp memory devices added. <p>Window manager</p> <ul style="list-style-type: none"> - New function WM_SetWindowPos() added. - New function WM_GetScrollPosH() added. - New function WM_GetScrollPosV() added. - New function WM_SetScrollPosH() added. - New function WM_SetScrollPosV() added. <p>Widgets</p> <ul style="list-style-type: none"> - BUTTON: New function BUTTON_SetDefaultFocusColor(). - BUTTON: New function BUTTON_SetFocusColor(). - BUTTON: New config option BUTTON_FOCUSCOLOR_DEFAULT. - CHECKBOX: New function CHECKBOX_SetBoxBkColor(). - CHECKBOX: New function CHECKBOX_SetDefaultFocusColor(). - CHECKBOX: New function CHECKBOX_SetFocusColor(). - CHECKBOX: Notification WM_NOTIFICATION_VALUE_CHANGED added. - CHECKBOX: Image of 'unchecked' state can now be set. - EDIT: New function EDIT_SetTextMode() added. - LISTBOX: New function LISTBOX_SetDefaultTextAlign() added. - LISTBOX: New function LISTBOX_SetTextAlign() added. - LISTBOX: New function LISTBOX_SetDefaultTextAlign() added. - LISTBOX: New function LISTBOX_SetTextAlign() added. - LISTBOX: New color index LISTBOX_CI_DISABLED added. - LISTVIEW: New color index LISTVIEW_CI_DISABLED added. - MENU: Keyboard support added. - PROGBAR: Vertical progress bar supported. - RADIO: New function RADIO_SetDefaultFocusColor() added. - RADIO: New function RADIO_SetFocusColor() added. - RADIO: Config options reworked. - SCROLLBAR: New function SCROLLBAR_SetColor() added. - SCROLLBAR: New function SCROLLBAR_SetDefaultColor() added. - SLIDER: New function SLIDER_SetDefaultFocusColor() added. - SLIDER: New function SLIDER_SetFocusColor() added. - Widget callback functions exported. <p>Antialiasing</p> <ul style="list-style-type: none"> - New function GUI_AA_DrawPolyOutlineEx() added. <p>Unicode</p> <ul style="list-style-type: none"> - New function GUI_UC_ConvertUC2UTF8() added. - New function GUI_UC_ConvertUTF82UC() added. <p>Display drivers</p> <ul style="list-style-type: none"> - New driver LCD0323 for Solomon SSD0323 controller added. - New driver LCD1200 for Toppoly C0C0 and C0E0 controller added. - New driver LCD13701 for Epson S1D13701 controller added. - New driver LCDNoritake for Noritake displays added. - New driver LCD667XX for Hitachi HD66772, HD66766 and HD66789 controller. - New driver LCDXylon added. - LCDLin driver: New config macro LCD_FILL_RECT added. - LCDLin driver: New config macro LCD_LIN_SWAP added. - LCDLin driver: 32 bit version now supports 1 and 2 bpp color depth.

Software version	Date	By	Explanation
3.92	041021	JE	<p>Simulator and Viewer - The simulation now can easily be used with other simulations.</p> <p>Window manager - New functions added: WM_PaintWindowAndDescs(), WM_Update() and WM_UpdateWindowAndDescs().</p> <p>Widgets - Flag GUI_MESSAGEBOX_CF_MODAL added for creating a modal message box. - Notification codes added to MULTIPAGE widget. - New functions added: LISTVIEW_DisableRow(), LISTVIEW_EnableRow() and LISTVIEW_GetItemText().</p> <p>Virtual screen support - New functions GUI_GetOrg() and GUI_SetOrg().</p> <p>Display driver - Support for Sitronix ST7565 added to Page1bpp driver. - LCDLin driver (8/16 bit): LCD_SET_LUT_ENTRY added.</p>
3.90	040818	JE	<p>Window manager - New function WM_GetPrevSibling() added. - New message WM_MOUSEOVER added. - New create flag WM_CF_MEMDEV_ON_REDRAW added.</p> <p>Widgets - MENU widget added. - WM_NOTIFICATION_SCROLL_CHANGED added to DROPODOWN, LISTBOX, LISTVIEW and MULTIEDIT</p> <p>BUTTON widget: Support for disabled state added. - New functions added: DROPODOWN_SetTextHeight(), DROPODOWN_SetTextAlign(), FRAMEWIN_AddMenu(), FRAMEWIN_SetResizable(), LISTVIEW_GetBkColor(), LISTVIEW_GetTextColor(), LISTVIEW_SetItemTextColor(), LISTVIEW_SetItemBkColor()</p> <p>Display driver - Support for Solomon SSD1815 added to Page1bpp driver</p>
3.82	040714	JE	<p>Window manager - Functions added: WM_IsCompletelyVisible(), WM_IsEnabled() - Message added: WM_NOTIFY_VIS_CHANGED - Config macro added: WM_SUPPORT_NOTIFY_VIS_CHANGED</p> <p>Widgets - Functions added: BUTTON_SetDefaultAlign(), BUTTON_SetDefaultBkColor(), BUTTON_SetDefaultFont(), BUTTON_SetDefaultTextColor(), BUTTON_SetDefaultAlign(), BUTTON_SetDefaultBkColor(), BUTTON_SetDefaultFont(), BUTTON_SetDefaultTextColor(), CHECKBOX_SetDefaultBkColor(), CHECKBOX_SetDefaultFont(), CHECKBOX_SetDefaultSpacing(), CHECKBOX_SetDefaultTextAlign(), CHECKBOX_SetDefaultTextColor(), CHECKBOX_SetState(), CHECKBOX_SetBkColor(), CHECKBOX_SetDefaultBkColor(), CHECKBOX_SetDefaultFont(), CHECKBOX_SetDefaultSpacing(), CHECKBOX_SetDefaultTextAlign(), CHECKBOX_SetDefaultTextColor(), CHECKBOX_SetFont(), CHECKBOX_SetNumStates(), CHECKBOX_SetSpacing(), CHECKBOX_SetState(), CHECKBOX_SetText(), CHECKBOX_SetTextAlign(), CHECKBOX_SetTextColor(), LISTBOX_SetDefaultBkColor(), LISTBOX_SetDefaultTextColor(), LISTBOX_SetDefaultBkColor(), LISTBOX_SetDefaultTextColor() - Config macros added: BUTTON_ALIGN_DEFAULT, BUTTON.REACT.ON.LEVEL, CHECKBOX_BKCOLOR_DEFAULT, CHECKBOX_FONT_DEFAULT, CHECKBOX_IMAGE0_DEFAULT, CHECKBOX_IMAGE1_DEFAULT, CHECKBOX_SPACING_DEFAULT, CHECKBOX_TEXTALIGN_DEFAULT, CHECKBOX_TEXTCOLOR_DEFAULT</p> <p>Display drivers - Support for S1D13A03, S1D13A04 and S1D13A05 added.</p>

Software version	Date	By	Explanation
3.80	0040503	JE	<p>Simulator & Viewer</p> <ul style="list-style-type: none"> - SIM_SetMag added. <p>Widgets</p> <ul style="list-style-type: none"> - New functions added: EDIT_GetNumChars(), EDIT_SetSel(), EDIT_SetpfAddKeyEx(), EDIT_SetInsertMode(), EDIT_SetFloatValue(), RADIO_SetDefaultFont(), RADIO_SetDefaultTextColor(), RADIO_SetBkColor(), RADIO_SetDefaultFont(), RADIO_SetDefaultTextColor(), RADIO_SetFont(), RADIO_SetGroupID(), RADIO_SetText(), RADIO_SetTextColor(), SLIDER_SetNumTicks(). <p>Display drivers:</p> <ul style="list-style-type: none"> - New driver LCDPage4bpp added (support for Sitronix ST7528) - New driver LCD1781 added (support for Solomon SSD1781). - Support for Novatec NT7502, Samsung S6B1713 and Philips PCD8544 added to LCDPage1bpp.
3.76	040123	JE	<p>Widgets</p> <ul style="list-style-type: none"> - Notification codes added to SLIDER and SCROLLBAR - EDIT_SetDefaultTextColor() added. - EDIT_SetDefaultBkColor() added. - EDIT_SetDefaultTextColor() added. - EDIT_SetDefaultBkColor() added. - EDIT_SetDefaultAlign() added. - DROPODOWN_SetScrollbarWidth() added. - LISTBOX_SetScrollbarWidth() added. <p>Memory allocation:</p> <ul style="list-style-type: none"> - Limitation for 16 bit CPU's removed.
3.74	031219	JE	<p>Colors:</p> <ul style="list-style-type: none"> - GUI_MAGENTA changed from 0x8b008b to 0xFF00FF <p>Memory devices:</p> <ul style="list-style-type: none"> - New function GUI_MEMDEV_CreateFixed() added. <p>Widgets:</p> <ul style="list-style-type: none"> - FRAMEWIN_CreateButton-functions renamed to FRAMEWIN_AddButton... - New functions TEXT_SetBkColor() and RADIO_SetBkColor() added.
3.72	031204	JE	<p>Displaying text:</p> <ul style="list-style-type: none"> - GUI_DispStringInRectEx added. <p>Window Manager:</p> <ul style="list-style-type: none"> - Functions WM_SetID and WM_ForEachDesc added. <p>Fonts:</p> <ul style="list-style-type: none"> - System independent font support added. <p>Widget library:</p> <ul style="list-style-type: none"> - BUTTON_SetTextAlign added. - DROPODOWN_SetItemSpacing added. - EDIT_SetFloatValue added. - EDIT_SetFloatMode added. - LISTBOX_SetItemSpacing added. - LISTVIEW_SetLBorder added. - LISTVIEW_SetRBorder added. - LISTVIEW_SetRowHeight added. - RADIO_SetBkColor added. - SLIDER_SetBkColor added. <p>Display driver:</p> <ul style="list-style-type: none"> - Support for UltraChip UC1611 and Hitachi HD66750 added.
3.70	031010	JE	Support for the following LCD controllers added: Fujitsu MB86290A (Cresmon), Fujitsu MB86291 (Scarlet), Fujitsu MB86292 (Orchid), Fujitsu MB86293 (Coral Q), Fujitsu MB86294 (Coral B), Fujitsu MB86295 (Coral P), Samsung S6B33B1X and UltraChip UC1606.
3.62	030901	JE	<p>VNC support added.</p> <p>Functions added to DROPODOWN, CHECKBOX and RADIO widget</p> <p>Support for Epson SED1520 added.</p> <p>Support for Fujitsu MB87J2020 and MB87J2120 added.</p>
3.60	030714	RS	<p>GUI_MEMDEV_GetDataPtr() added</p> <p>WM_CF_CONST_OUTLINE added</p> <p>BUTTON_IsPressed added.</p> <p>WIDGET_ITEM_GET_YSIZE added to WIDGET_ITEM_DRAW_INFO structure.</p>
3.58	030701	JE	<p>New LCD driver added.</p> <p>Clipping of window manager accelerated.</p> <p>Drawing of buttons accelerated.</p>

Software version	Date	By	Explanation
3.56	030626	JE	Modal windows added. Improved AVR support. Additional LCD drivers added.
3.54	030603	RS	JPEG file support added.
3.52	030516	JE	Functions added to LISTVIEW widget. WM_GetFocussedWindow added to window manager.
3.50	030509	RS	Core functions addad. Functions addad to window manager.
3.48	030415	JE	Functions added to window manager. Functions added to widgets.
3.46	030404	JE	Multi layer support added.
3.44	030319	JE	UTF-8 support added MULTIEDIT widget added. User drawn LISTBOX added.
3.42	030227	JE	Functions added to window manager. Functions added to FRAMEWIN widget. Multi selection mode added to LISTBOX widget.
3.40	030217	JE	Multi selection mode added to LISTBOX widget. Activation of frame windows changed.
3.38	030206	JE	Functions added to SCROLLBAR widget, BUTTON widget and LISTBOX widget.
3.36	030120	JE	Functions added to FRAMEWIN widget and window manager. Dynamic memory support added.
3.34	020926	RS	EDIT_SetCursorAtChar() and EDIT_SetCursorAtPixel() added.
3.32	020920	RS	WM_SetpfPollPID() added. BMP file support added.
3.30	020912	RS	Cursor support added.
3.28	020903	RS	GUI_TOUCH_StoreStateEx() added. Edit widget: flags added. Simulation: add. options for simulation of colored monochrome displays added.
3.28	020830	RS	SIM_SetLCDColorBlack() and SIM_SetLCDColorWhite() added. New starter kit "Berni". LISTBOX_AddString() and LISTBOX_GetNumItems() added.
3.26	020820	RS	Mouse and keyboard support added.
3.24	020809	RS	Macros GUI_TOUCH_XSIZE and GUI_TOUCH_YSIZE added. Simulation project structure changed. Macros LCD_TIMERINIT0, LCD_TIMERINIT1, LCD_ON and LCD_OFF added. Support for 444 color mode added. GUI_Exec() and GUI_Exec1() added.
3.22	020719	RS	Scrollbars, radio buttons added.
3.20	020618	RS	Dialog boxes added. Slider and check box widgets added. 3D effects added.

Table of Contents

1	Introduction to emWin	49
1.1	Purpose of this document	49
1.2	Assumptions	49
1.3	Requirements.....	50
1.3.1	Target system (hardware)	50
1.3.2	Development environment (compiler).....	50
1.4	Features.....	50
1.5	Samples and demos.....	51
1.6	Starter kits	52
1.7	How to use this manual	52
1.8	Typographic conventions for syntax	53
1.9	Screen and coordinates	53
1.10	How to connect the LCD to the microcontroller	53
1.11	Data types.....	54
2	Getting Started.....	55
2.1	Recommended directory structure.....	56
2.1.1	Subdirectories	56
2.1.2	Include directories	56
2.2	Adding emWin to the target program	57
2.3	Creating a library.....	57
2.3.1	Adapting the library batch files to a different system	57
2.4	"C" files to include in the project.....	59
2.5	Configuring emWin	60
2.6	Initializing emWin	60
2.7	Using emWin with target hardware	60
2.8	The "Hello world" sample program	61
	Adding functionality to the "Hello world" program	62
3	Simulator.....	63
3.1	Using the simulator.....	64
3.1.1	Using the simulator in the trial emWin version	64
3.1.1.1	Directory structure.....	64
3.1.1.2	Visual C++ workspace.....	64
3.1.1.3	Compiling the demo program	64
3.1.1.4	Compiling the samples	65
3.1.2	Using the simulator with the emWin source	66
3.1.2.1	Directory structure.....	66
3.1.2.2	Visual C++ workspace.....	66
3.1.2.3	Compiling the application.....	66
3.2	Device simulation	68
3.2.1	Device simulator API.....	68
	SIM_GUI_SetLCDColorBlack(), SIM_GUI_SetLCDColorWhite()	69
	SIM_GUI_SetLCDPos()	70
	SIM_GUI_SetMag().....	70
	SIM_GUI_SetTransColor().....	70
3.2.2	Hardkey simulation	71
3.2.2.1	Hardkey simulator API.....	72
	SIM_HARDKEY_GetNum().....	72
	SIM_HARDKEY_GetState().....	72

SIM_HARDKEY_SetCallback()	73
SIM_HARDKEY_SetMode()	73
SIM_HARDKEY_SetState()	74
3.3 Integrating the emWin simulation into an existing simulation	75
3.3.1 Directory structure	75
3.3.2 Using the simulation library	75
3.3.2.1 Modifying WinMain	75
3.3.2.2 Sample application	75
3.3.3 Integration into the embOS Simulation	77
3.3.3.1 WinMain	77
3.3.3.2 Target program (main)	77
3.3.4 GUI simultion API	79
SIM_GUI_CreateLCDInfoWindow()	79
SIM_GUI_CreateLCDWindow()	79
SIM_GUI_Exit()	80
SIM_GUI_Init()	80
SIM_GUI_SetLCDWindowHook()	80
4 Viewer	83
4.1 Using the viewer	84
4.1.1 Using the simulator and the viewer	84
4.1.2 Using the viewer with virtual pages	85
4.1.3 Always on top	85
4.1.4 Open further windows of the display output	85
4.1.5 Zooming	85
4.1.6 Copy the output to the clipboard	86
4.1.7 Using the viewer with multiple displays	86
4.1.8 Using the viewer with multiple layers	87
5 Displaying Text	89
5.1 Basic routines	90
5.2 Text API	90
5.3 Routines to display text	91
GUI_DispChar()	91
GUI_DispCharAt()	92
GUI_DispChars()	92
GUI_DispNextLine()	92
GUI_DispString()	93
GUI_DispStringAt()	93
GUI_DispStringAtCEOL()	93
GUI_DispStringHCenterAt()	94
GUI_DispStringInRect()	94
GUI_DispStringInRectEx()	94
GUI_DispStringLen()	95
5.4 Selecting text drawing modes	96
GUI_GetTextMode()	97
GUI_SetTextMode()	97
GUI_SetTextStyle()	98
5.5 Selecting text alignment	98
GUI_GetTextAlign()	98
GUI_SetLBorder()	98
GUI_Set.TextAlign()	98
5.6 Setting the current text position	99
GUI_GotoXY(), GUI_GotoX(), GUI_GotoY()	99
5.7 Retrieving the current text position	100
GUI_GetDispPosX()	100
GUI_GetDispPosY()	100
5.8 Routines to clear a window or parts of it	100
GUI_Clear()	100
GUI_DisposeCEOL()	100

6	Displaying Values	103
6.1	Value API	104
6.2	Displaying decimal values	104
	GUI_DispDec()	104
	GUI_DispDecAt()	105
	GUI_DispDecMin()	105
	GUI_DispDecShift()	106
	GUI_DispDecSpace()	106
	GUI_DispSDec()	106
	GUI_DispSDecShift()	107
6.3	Displaying floating-point values	108
	GUI_DispFloat()	108
	GUI_DispFloatFix()	109
	GUI_DispFloatMin()	109
	GUI_DispSFloatFix()	110
	GUI_DispSFloatMin()	110
6.4	Displaying binary values	110
	GUI_DispBin()	110
	GUI_DispBinAt()	111
6.5	Displaying hexadecimal values	111
	GUI_DispHex()	111
	GUI_DispHexAt()	112
6.6	Version of emWin	112
	GUI_GetVersionString()	112
7	2-D Graphic Library	113
7.1	Graphic API	114
7.2	Drawing modes	115
	GUI_GetDrawMode()	115
	GUI_SetDrawMode()	115
	Example	116
7.3	Query current client rectangle	116
	GUI_GetClientRect()	116
7.4	Basic drawing routines	117
	GUI_ClearRect()	117
	GUI_DrawPixel()	117
	GUI_DrawPoint()	117
	GUI_DrawRect()	118
	GUI_DrawRectEx()	118
	GUI_FillRect()	118
	GUI_FillRectEx()	118
	GUI_InvertRect()	119
7.5	Drawing bitmaps	119
	GUI_DrawBitmap()	119
	GUI_DrawBitmapExp()	120
	GUI_DrawBitmapEx()	120
	GUI_DrawBitmapMag()	121
	GUI_DrawStreamedBitmap()	121
7.6	Drawing lines	121
	GUI_DrawHLine()	121
	GUI_DrawLine()	122
	GUI_DrawLineRel()	122
	GUI_DrawLineTo()	122
	GUI_DrawPolyLine()	123
	GUI_DrawVLine()	123
	GUI_GetLineStyle()	123
	GUI_MoveRel()	124
	GUI_MoveTo()	124
	GUI_SetLineStyle()	124
7.7	Drawing polygons	125

	GUI_DrawPolygon().....	125
	GUI_EnlargePolygon()	125
	GUI_FillPolygon()	126
	GUI_MagnifyPolygon()	126
	GUI_RotatePolygon()	127
	Example	127
7.8	Drawing circles	129
	GUI_DrawCircle().....	129
	GUI_FillCircle()	129
7.9	Drawing ellipses	130
	GUI_DrawEllipse().....	130
	GUI_FillEllipse()	130
	Example	131
7.10	Drawing arcs	131
	GUI_DrawArc().....	131
	Example	131
7.11	Drawing graphs	132
	GUI_DrawGraph()	132
	Example	133
7.12	Saving and restoring the GUI-context	133
	GUI_RestoreContext()	133
	GUI_SaveContext()	133
7.13	Clipping	134
	GUI_SetClipRect()	134
8	Displaying bitmap files	135
8.1	BMP file support	136
8.1.1	BMP file API.....	136
	GUI_BMP_Draw()	136
	GUI_BMP_GetXSize()	136
	GUI_BMP_GetYSize()	137
	GUI_BMP_Serialize()	137
	GUI_BMP_SerializeEx()	137
8.2	JPEG file support	139
8.2.1	Supported JPEG compression methods	139
8.2.2	Converting a JPEG file to 'C' source.....	139
8.2.3	Displaying JPEG files.....	139
8.2.4	Memory usage	140
8.2.5	JPEG file API.....	140
	GUI_JPEG_Draw()	140
	GUI_JPEG_GetInfo()	140
8.3	GIF file support	142
8.3.1	Converting a GIF file to 'C' source.....	142
8.3.2	Displaying GIF files.....	142
8.3.3	Memory usage	142
8.3.4	GIF file API.....	142
	GUI_GIF_Draw()	142
	GUI_GIF_GetXSize()	143
	GUI_GIF_GetYSize()	143
9	Fonts	145
9.1	Supported font types	146
9.2	Font API.....	146
9.3	Selection of a font	146
	GUI_GetFont()	147
	GUI_SetFont()	147
	GUI_SIF_CreateFont()	147
	GUI_SIF_DeleteFont()	148
9.4	Font-related functions	149
	GUI_GetCharDistX()	149

	GUI_GetFontDistY()	149
	GUI_GetFontInfo()	149
	GUI_GetFontSizeY()	150
	GUI_GetStringDistX()	150
	GUI_GetTextExtend()	150
	GUI_GetYDistOfFont()	150
	GUI_GetYSizeOfFont()	150
	GUI_IsInFont()	151
9.5	Character sets	152
9.5.1	ASCII	152
9.5.2	ISO 8859-1 Western Latin character set	152
9.5.3	Unicode	154
9.6	Font converter	155
9.6.1	Adding fonts	155
9.7	Standard fonts	156
9.7.1	Font identifier naming convention	156
9.7.2	Font file naming convention	157
9.7.3	Measurement, ROM-size and character set of fonts	157
9.7.4	Proportional fonts	158
	GUI_Font8_ASCII	158
	GUI_Font8_1	158
	GUI_Font10_ASCII	158
	GUI_Font10_1	159
	GUI_Font10S_ASCII	159
	GUI_Font10S_1	159
	GUI_Font13_ASCII	160
	GUI_Font13_1	160
	GUI_Font13H_ASCII	160
	GUI_Font13H_1	161
	GUI_Font13HB_ASCII	161
	GUI_Font13HB_1	161
	GUI_Font16_ASCII	162
	GUI_Font16_1	162
	GUI_Font16_HK	163
	GUI_Font16_1HK	163
	GUI_Font16B_ASCII	164
	GUI_Font16B_1	164
	GUI_FontComic18B_ASCII	164
	GUI_FontComic18B_1	165
	GUI_Font24_ASCII	165
	GUI_Font24_1	166
	GUI_Font24B_ASCII	167
	GUI_Font24B_1	167
	GUI_Font32_ASCII	168
	GUI_Font32_1	168
	GUI_Font32B_ASCII	169
	GUI_Font32B_1	170
	GUI_FontComic24B_ASCII	171
	GUI_FontComic24B_1	172
	GUI_FontD32	173
	GUI_FontD48	173
	GUI_FontD64	174
	GUI_FontD80	175
9.7.5	Monospaced fonts	176
	GUI_Font4x6	176
	GUI_Font6x8	176
	GUI_Font6x9	176
	GUI_Font8x8	177
	GUI_Font8x9	177
	GUI_Font8x10_ASCII	177
	GUI_Font8x12_ASCII	178

GUI_Font8x13_ASCII.....	178
GUI_Font8x13_1.....	178
GUI_Font8x15B_ASCII.....	179
GUI_Font8x15B_1.....	179
GUI_Font8x16	179
GUI_Font8x17	180
GUI_Font8x18	180
GUI_Font8x16x1x2.....	181
GUI_Font8x16x2x2.....	181
GUI_Font8x16x3x3.....	182
GUI_FontD24x32	183
GUI_FontD36x48	183
GUI_FontD48x64	184
GUI_FontD60x80	185
10 Bitmap Converter	187
10.1 What it does.....	188
10.2 Loading a bitmap	188
10.2.1 Supported file formats	188
10.2.2 Loading from a file	188
10.2.3 Using the clipboard.....	188
10.3 Generating "C" files from bitmaps.....	189
10.4 Color conversion	191
10.5 Compressed bitmaps	192
10.6 Using a custom palette	192
10.7 BmpCvt.exe: Command line usage	193
10.7.1 Format for commands.....	193
10.7.2 Valid command line options	193
10.8 Example of a converted bitmap	195
Compressing the file	196
11 Colors.....	199
11.1 Predefined colors.....	200
11.2 The color bar test routine	200
11.3 Fixed palette modes	201
1 mode: 1 bpp (black and white).....	202
2 mode: 2 bpp (4 grayscales).....	202
4 mode: 4 bpp (16 grayscales).....	202
111 mode: 3 bpp (2 levels per color)	202
111 mode: 3 bpp (2 levels per color), red and blue swapped.....	202
222 mode: 6 bpp (4 levels per color)	203
222 mode: 6 bpp (4 levels per color), red and blue swapped.....	203
233 mode: 8 bpp	203
233 mode: 8 bpp, red and blue swapped	203
323 mode: 8 bpp	204
323 mode: 8 bpp, red and blue swapped	204
332 mode: 8 bpp	205
332 mode: 8 bpp, red and blue swapped	205
44412 mode:.....	205
44416 mode:.....	206
44412 mode: red and blue swapped	206
44416 mode: red and blue swapped	206
444121 mode:.....	206
555 mode: 15 bpp.....	206
555 mode: 15 bpp, red and blue swapped	206
565 mode: 16 bpp.....	206
565 mode: 16 bpp, red and blue swapped	206
556 mode: 16 bpp.....	207
556 mode: 16 bpp, red and blue swapped	207
8666 mode: 8bpp, 6 levels per color + 16 grayscales.....	207

	86661 mode: 8bpp, 6 levels per color + 16 grayscales + transparency	207
11.4	Custom palette modes.....	208
11.5	Modifying the color lookup table at run time.....	208
11.6	Color API.....	208
11.7	Basic color functions	209
	GUI_GetBkColor().....	209
	GUI_GetBkColorIndex().....	209
	GUI_GetColor()	209
	GUI_GetColorIndex()	210
	GUI_SetBkColor().....	210
	GUI_SetBkColorIndex()	210
	GUI_SetColor()	210
	GUI_SetColorIndex()	211
11.8	Index & color conversion	211
	GUI_CalcColorDist().....	211
	GUI_CalcVisColorError()	211
	GUI_Color2Index()	211
	GUI_Color2VisColor()	212
	GUI_ColorIsAvailable()	212
	GUI_Index2Color()	212
11.9	Lookup table (LUT) group	212
	GUI_InitLUT()	212
	GUI_SetLUTColor()	213
	GUI_SetLUTEEntry()	213
12	Memory Devices	215
12.1	Using memory devices: an illustration	216
12.2	Supported color depth (bpp)	216
12.3	Memory devices and the window manager	217
12.4	Memory requirements	217
12.5	Performance	218
12.6	Basic functions	218
12.7	In order to be able to use memory devices.....	218
12.8	Multi layer / multi display configurations	219
12.9	Configuration options	219
12.9.1	GUI_USE_MEMDEV_1BPP_FOR_SCREEN	219
12.10	Memory device API	219
12.11	Basic functions	220
	GUI_MEMDEV_Clear()	220
	GUI_MEMDEV_CopyFromLCD().....	220
	GUI_MEMDEV_CopyToLCD()	220
	GUI_MEMDEV_CopyToLCDAA()	221
	GUI_MEMDEV_CopyToLCDAt()	221
	GUI_MEMDEV_Create()	221
	GUI_MEMDEV_CreateEx()	222
	GUI_MEMDEV_CreateFixed()	222
	GUI_MEMDEV_Delete()	224
	GUI_MEMDEV_GetDataPtr()	224
	GUI_MEMDEV_GetXSize()	225
	GUI_MEMDEV_GetYSize()	225
	GUI_MEMDEV_ReduceYSize()	225
	GUI_MEMDEV_Select()	225
	GUI_MEMDEV_SetOrg()	226
	GUI_MEMDEV_Write()	226
	GUI_MEMDEV_WriteAlpha()	226
	GUI_MEMDEV_WriteAlphaAt()	226
	GUI_MEMDEV_WriteAt()	227
	GUI_MEMDEV_WriteEx()	227
	GUI_MEMDEV_WriteExAt()	227
	GUI_SelectLCD()	228
	Example for using a memory device	228

12.12	Banding memory device	230
	GUI_MEMDEV_Draw()	230
	Example for using a banding memory device	231
12.13	Auto device object	233
	GUI_MEMDEV_CreateAuto()	233
	GUI_MEMDEV_DeleteAuto()	233
	GUI_MEMDEV_DrawAuto()	234
	Example for using an auto device object	235
12.14	Measurement device object	238
	GUI_MEASDEV_Create()	238
	GUI_MEASDEV_ClearRect()	238
	GUI_MEASDEV_Delete()	238
	GUI_MEASDEV_GetRect()	238
	GUI_MEASDEV_Select()	238
	Example	239
13	Execution Model: Single Task / Multitask	241
13.1	Supported execution models	242
13.2	Single task system (superloop)	242
13.2.1	Description	242
13.2.2	Superloop example (without emWin)	242
13.2.3	Advantages	242
13.2.4	Disadvantages	242
13.2.5	Using emWin	242
13.2.6	Superloop example (with emWin)	243
13.3	Multitask system: one task calling emWin	243
13.3.1	Description	243
13.3.2	Advantages	243
13.3.3	Disadvantages	243
13.3.4	Using emWin	243
13.4	Multitask system: multiple tasks calling emWin	244
13.4.1	Description	244
13.4.2	Advantages	244
13.4.3	Disadvantages	244
13.4.4	Using emWin	244
13.4.5	Recommendations	244
13.4.6	Example	244
13.5	GUI configuration macros for multitasking support	245
	GUI_MAXTASK	245
	GUI_OS	245
	GUI_X_SIGNAL_EVENT	245
	GUI_X_WAIT_EVENT	246
13.6	Kernel interface routine API	246
	GUI_X_InitOS()	247
	GUI_X_GetTaskID()	247
	GUI_X_Lock()	247
	GUI_X_SignalEvent()	247
	GUI_X_Unlock()	248
	GUI_X_WaitEvent()	248
	Examples	248
14	The Window Manager (WM)	251
14.1	Explanation of terms	252
14.2	Callback mechanism, invalidation and rendering	253
14.2.1	Rendering without callbacks	253
14.2.2	Rendering using callbacks	254
14.2.3	Background window redrawing and callback	254
14.2.4	Invalidation	255
14.2.5	Rendering of transparent windows	255
14.3	Messages	256

14.3.1	Message structure	256
14.3.2	List of messages.....	256
14.3.3	System-defined messages	257
	WM_CREATE	257
	WM_DELETE	257
	WM_GET_ID	257
	WM_INIT_DIALOG	257
	WM_KEY.....	257
	WM_MOVE.....	258
	WM_NOTIFY_PARENT	258
	WM_NOTIFY_VIS_CHANGED	258
	WM_PAINT	259
	WM_SET_FOCUS	259
	WM_SET_ID.....	259
	WM_SIZE	259
14.3.4	Pointer input device (PID) messages	260
	WM_MOUSEOVER	260
	WM_PID_STATE_CHANGED.....	260
	WM_TOUCH	260
	WM_TOUCH_CHILD	261
	Example.....	262
14.3.5	System-defined notification codes	263
	WM_NOTIFICATION_CHILD_DELETED	263
14.3.6	Application-defined messages.....	263
14.4	Configuration options	264
	WM_SUPPORT_NOTIFY_VIS_CHANGED.....	264
	WM_SUPPORT_TRANSPARENCY	264
14.5	WM API	265
14.5.1	Using the WM API functions	267
14.6	Basic functions	267
	WM_Activate()	267
	WM_AttachWindow().....	267
	WM_AttachWindowAt().....	268
	WM_BringToBottom().....	268
	WM_BringToFront().....	268
	WM_BroadcastMessage().....	268
	WM_ClrHasTrans().....	269
	WM_CreateWindow()	269
	WM_CreateWindowAsChild()	271
	WM_Deactivate().....	272
	WM_DefaultProc()	272
	WM_DeleteWindow().....	272
	WM_DetachWindow()	273
	WM_DisableWindow()	273
	WM_EnableWindow()	273
	WM_Exec().....	273
	WM_Exec1()	274
	WM_ForEachDesc()	274
	WM_GetCallback()	275
	WM_GetActiveWindow()	275
	WM_GetClientRect().....	276
	WM_GetClientRectEx()	276
	WM_GetDesktopWindow()	276
	WM_GetDesktopWindowEx()	276
	WM_GetDialogItem()	277
	WM_GetFirstChild().....	277
	WM_GetFocussedWindow()	277
	WM_GetHasTrans().....	278
	WM_GetInvalidRect()	278
	WM_GetNextSibling()	278
	WM_GetOrgX(), WM_GetOrgY().....	279

	WM_GetParent()	279
	WM_GetPrevSibling().....	279
	WM_GetStayOnTop()	279
	WM_GetUserData()	280
	WM_GetWindowOrgX(), WM_GetWindowOrgY().....	280
	WM_GetWindowRect()	280
	WM_GetWindowRectEx().....	281
	WM_GetWindowSizeX(), WM_GetWindowSizeY()	281
	WM_HasCaptured()	281
	WM_HasFocus()	281
	WM_HideWindow()	282
	WM_InvalidateArea()	282
	WM_InvalidateRect()	282
	WM_InvalidateWindow()	283
	WM_IsCompletelyVisible()	283
	WM_IsEnabled()	283
	WM_IsVisible()	284
	WM_IsWindow().....	284
	WM_MakeModal().....	284
	WM_MoveChildTo()	285
	WM_MoveTo()	285
	WM_MoveWindow()	285
	WM_NotifyParent()	285
	WM_Paint()	286
	WM_PaintWindowAndDescs()	286
	WM_ReleaseCapture()	286
	WM_ResizeWindow()	286
	WM_SelectWindow().....	287
	WM_SendMessage()	287
	WM_SendToParent()	287
	WM_SetDesktopColor()	287
	WM_SetDesktopColorEx().....	288
	WM_SetCallback()	288
	WM_SetCapture()	288
	WM_SetCreateFlags()	289
	WM_SetFocus	289
	WM_SetHasTrans().....	289
	WM_SetId()	290
	WM_SetpfPollPID()	290
	WM_SetSize()	290
	WM_SetWindowPos()	291
	WM_SetXSize()	291
	WM_SetYSize()	291
	WM_SetStayOnTop()	291
	WM_SetTransState()	292
	WM_SetUserClipRect()	292
	WM_SetUserData()	293
	WM_ShowWindow()	293
	WM_Update()	294
	WM_UpdateWindowAndDescs()	294
	WM_ValidateRect()	294
	WM_ValidateWindow()	294
14.7	Memory device support (optional)	296
	WM_DisableMemdev()	296
	WM_EnableMemdev()	296
14.8	Widget related functions	297
	WM_GetClientWindow()	297
	WM_GetId()	297
	WM_GetInsideRect()	297
	WM_GetInsideRectEx()	298
	WM_GetScrollPosH()	298

WM_GetScrollPosV()	298
WM_GetScrollState()	298
WM_SetScrollPosH()	299
WM_SetScrollPosV()	299
WM_SetScrollState()	300
14.9 Example	301
15 Window Objects (Widgets)	303
15.1 Some basics	304
15.1.1 Available widgets	304
15.1.2 Understanding the redrawing mechanism	305
15.1.3 How to use widgets	305
15.2 General widget API	306
15.2.1 API reference: WM routines used for widgets	306
15.2.2 API reference: routines common to all widgets	307
<WIDGET>_Callback()	307
<WIDGET>_CreateIndirect()	307
WIDGET_SetDefaultEffect()	308
WIDGET_SetDefaultEffect()	308
WIDGET_SetDefaultEffect_3D()	309
WIDGET_SetDefaultEffect_None()	309
WIDGET_SetDefaultEffect_Simple()	310
WIDGET_SetEffect()	310
15.2.3 User drawn widgets	310
15.3 BUTTON: Button widget	312
15.3.1 Configuration options	312
BUTTON.REACT_ON_LEVEL	312
BUTTON_BKCOLOR0_DEFAULT, BUTTON_BKCOLOR1_DEFAULT	312
15.3.2 Notification codes	313
15.3.3 Keyboard reaction	313
15.3.4 BUTTON API	313
BUTTON_Create()	314
BUTTON_CreateAsChild()	314
BUTTON_CreateEx()	315
BUTTON_CreateIndirect()	315
BUTTON_GetBitmap()	315
BUTTON_GetBkColor()	316
BUTTON_GetDefaultBkColor()	316
BUTTON_GetDefaultFont()	316
BUTTON_GetDefaultTextAlign()	317
BUTTON_GetDefaultTextColor()	317
BUTTON_GetFont()	317
BUTTON_GetText()	317
BUTTON_GetTextColor()	318
BUTTON_IsPressed()	318
BUTTON_SetBitmap()	318
BUTTON_SetBitmapEx()	319
BUTTON_SetBkColor()	319
BUTTON_SetBMP()	320
BUTTON_SetBMPEx()	320
BUTTON_SetDefaultBkColor()	320
BUTTON_SetDefaultFocusColor()	321
BUTTON_SetDefaultFont()	321
BUTTON_SetDefaultTextAlign()	321
BUTTON_SetDefaultTextColor()	322
BUTTON_SetFocusColor()	322
BUTTON_SetFocussable()	322
BUTTON_SetFont()	323
BUTTON_SetPressed()	323
BUTTON_SetStreamedBitmap()	323
BUTTON_SetStreamedBitmapEx()	323

BUTTON_SetText()	324
BUTTON_SetTextAlign()	324
BUTTON_SetTextColor()	324
15.3.5 Examples	325
Using the BUTTON widget	325
Advanced use of the BUTTON widget	326
15.4 CHECKBOX: Check box widget	330
15.4.1 Configuration options	330
15.4.2 Notification codes	330
15.4.3 Keyboard reaction	331
15.4.4 CHECKBOX API	331
CHECKBOX_Check()	332
CHECKBOX_Create()	332
CHECKBOX_CreateIndirect()	332
CHECKBOX_CreateEx()	332
CHECKBOX_GetDefaultBkColor()	333
CHECKBOX_GetDefaultFont()	333
CHECKBOX_GetDefaultSpacing()	334
CHECKBOX_GetDefault TextAlign()	334
CHECKBOX_GetDefaultTextColor()	334
CHECKBOX_GetState()	334
CHECKBOX_GetText()	335
CHECKBOX_IsChecked()	335
CHECKBOX_SetBkColor()	335
CHECKBOX_SetBoxBkColor()	336
CHECKBOX_SetDefaultBkColor()	336
CHECKBOX_SetDefaultFocusColor()	337
CHECKBOX_SetDefaultFont()	337
CHECKBOX_SetDefaultImage()	337
CHECKBOX_SetDefaultSpacing()	338
CHECKBOX_SetDefault TextAlign()	338
CHECKBOX_SetDefaultTextColor()	338
CHECKBOX_SetFocusColor()	339
CHECKBOX_SetImage()	339
CHECKBOX_SetNumStates()	340
CHECKBOX_SetSpacing()	340
CHECKBOX_SetState()	340
CHECKBOX_SetText()	341
CHECKBOX_Set TextAlign()	341
CHECKBOX_SetTextColor()	342
CHECKBOX_Unccheck()	342
15.5 DROPODOWN: Dropdown widget	344
15.5.1 Configuration options	344
15.5.2 Notification codes	344
15.5.3 Keyboard reaction	344
15.5.4 DROPODOWN API	345
DROPODOWN_AddString()	345
DROPODOWN_Collapse()	346
DROPODOWN_Create()	346
DROPODOWN_CreateEx()	346
DROPODOWN_CreateIndirect()	347
DROPODOWN_DecSel()	347
DROPOWN_DeleteItem()	347
DROPOWN_Expand()	347
DROPOWN_GetNumItems()	348
DROPOWN_GetSel()	348
DROPOWN_IncSel()	348
DROPOWN_InsertString()	348
DROPOWN_SetAutoScroll()	349
DROPOWN_SetBkColor()	349
DROPOWN_SetColor()	350

DROPDOWN_SetDefaultColor()	350
DROPDOWN_SetDefaultFont()	350
DROPDOWN_SetDefaultScrollbarColor()	351
DROPDOWN_SetFont()	351
DROPDOWN_SetScrollbarColor()	351
DROPDOWN_SetScrollbarWidth()	352
DROPDOWN_SetSel()	352
DROPDOWN_SetItemSpacing()	352
DROPDOWN_SetTextAlign()	353
DROPDOWN_SetTextColor()	353
DROPDOWN_SetTextHeight()	353
15.6 EDIT: Edit widget	355
15.6.1 Configuration options	355
15.6.2 Notification codes	355
15.6.3 Keyboard reaction	356
15.6.4 EDIT API	356
EDIT_AddKey()	357
EDIT_Create()	357
EDIT_CreateAsChild()	357
EDIT_CreateEx()	358
EDIT_CreateIndirect()	359
EDIT_GetDefaultBkColor()	359
EDIT_GetDefaultFont()	359
EDIT_GetDefaultTextAlign()	359
EDIT_GetDefaultTextColor()	360
EDIT_GetFloatValue()	360
EDIT_GetNumChars	360
EDIT_GetText()	360
EDIT_GetValue()	361
EDIT_SetBinMode()	361
EDIT_SetBkColor()	361
EDIT_SetCursorAtChar()	362
EDIT_SetCursorAtPixel()	362
EDIT_SetDecMode()	362
EDIT_SetDefaultBkColor()	363
EDIT_SetDefaultFont()	363
EDIT_SetDefaultTextAlign()	363
EDIT_SetDefaultTextColor()	363
EDIT_SetFloatMode()	363
EDIT_SetFloatValue	364
EDIT_SetFont()	364
EDIT_SetHexMode()	364
EDIT_SetInsertMode	365
EDIT_SetMaxLen()	365
EDIT_SetpfAddKeyEx	365
EDIT_SetSel	366
EDIT_SetText()	366
EDIT_SetTextAlign()	367
EDIT_SetTextColor()	367
EDIT_SetTextMode()	367
EDIT_SetValue()	367
GUI_EditBin()	368
GUI_EditDec()	368
GUI_EditHex()	368
GUI>EditString()	369
15.6.5 Example	369
15.7 FRAMEWIN: Frame window widget	372
15.7.1 Structure of the frame window	373
15.7.2 Configuration options	374
15.7.3 Keyboard reaction	374
15.7.4 FRAMEWIN API	374

FRAMEWIN_AddButton()	375
FRAMEWIN_AddCloseButton()	376
FRAMEWIN_AddMaxButton()	376
FRAMEWIN_AddMenu()	377
FRAMEWIN_AddMinButton()	378
FRAMEWIN_Create()	378
FRAMEWIN_CreateAsChild()	379
FRAMEWIN_CreateEx()	379
FRAMEWIN_GetBorderSize()	380
FRAMEWIN_GetDefaultBarColor()	380
FRAMEWIN_GetDefaultBorderSize()	381
FRAMEWIN_GetDefaultClientColor()	381
FRAMEWIN_GetDefaultFont()	381
FRAMEWIN_GetDefaultTextColor()	381
FRAMEWIN_GetDefaultTitleHeight()	382
FRAMEWIN_GetTitleHeight()	382
FRAMEWIN_IsMaximized()	382
FRAMEWIN_IsMinimized()	382
FRAMEWIN_Maximize()	383
FRAMEWIN_Minimize()	383
FRAMEWIN_Restore()	384
FRAMEWIN_SetActive()	384
FRAMEWIN_SetBarColor()	385
FRAMEWIN_SetBorderSize()	386
FRAMEWIN_SetClientColor()	386
FRAMEWIN_SetDefaultBarColor()	386
FRAMEWIN_SetDefaultBorderSize()	387
FRAMEWIN_SetDefaultClientColor()	387
FRAMEWIN_SetDefaultFont()	387
FRAMEWIN_SetDefaultTextColor()	387
FRAMEWIN_SetDefaultTitleHeight()	388
FRAMEWIN_SetFont()	388
FRAMEWIN_SetMoveable()	388
FRAMEWIN_SetResizeable()	389
FRAMEWIN_SetText()	389
FRAMEWIN_SetTextAlign()	390
FRAMEWIN_SetTextColor()	390
FRAMEWIN_SetTextColorEx()	391
FRAMEWIN_SetTitleHeight()	391
FRAMEWIN_SetTitleVis()	392
15.8 GRAPH: Graph widget	393
15.8.1 Structure of the graph widget	393
15.8.2 Creating and deleting a graph widget	394
15.8.3 Drawing process	394
15.8.4 Supported types of curves	394
15.8.4.1 GRAPH_DATA_XY	395
15.8.4.2 GRAPH_DATA_YT	395
15.8.5 Configuration options	395
15.8.5.1 Graph widget	395
15.8.5.2 Scale object	395
15.8.6 Keyboard reaction	395
15.8.7 GRAPH API	395
15.8.7.1 Common routines	396
GRAPH_AttachData()	396
GRAPH_AttachScale()	397
GRAPH_CreateEx()	397
GRAPH_SetBorder()	398
GRAPH_SetColor()	398
GRAPH_SetGridDistX(), GRAPH_SetGridDistY()	399
GRAPH_SetGridFixedX()	399
GRAPH_SetGridOffY()	400

	GRAPH_SetGridVis()	401
	GRAPH_SetLineStyleH(), GRAPH_SetLineStyleV	401
	GRAPH_SetVSizeX(), GRAPH_SetVSizeY()	402
	GRAPH_SetUserDraw().....	403
15.8.7.2	GRAPH_DATA_YT related routines.....	404
	GRAPH_DATA_YT_AddValue()	404
	GRAPH_DATA_YT_Create()	404
	GRAPH_DATA_YT_SetOffY()	405
15.8.7.3	GRAPH_DATA_XY related routines.....	406
	GRAPH_DATA_XY_AddPoint()	406
	GRAPH_DATA_XY_Create()	406
	GRAPH_DATA_XY_SetOffX(), GRAPH_DATA_XY_SetOffY().....	407
	GRAPH_DATA_XY_SetPenSize().....	407
15.8.7.4	Scale related routines	408
	GRAPH_SCALE_Create()	408
	GRAPH_SCALE_SetFactor()	409
	GRAPH_SCALE_SetFont()	409
	GRAPH_SCALE_SetNumDecs()	410
	GRAPH_SCALE_SetOff()	410
	GRAPH_SCALE_SetPos().....	411
	GRAPH_SCALE_SetTextColor()	411
	GRAPH_SCALE_SetTickDist().....	412
15.9	HEADER: Header widget	413
15.9.1	Configuration options	413
15.9.2	Notification codes	413
15.9.3	Keyboard reaction	413
15.9.4	HEADER API.....	414
	HEADER.AddItem()	414
	HEADER.Create()	415
	HEADER.CreateAttached().....	415
	HEADER.CreateEx()	416
	HEADER.CreateIndirect()	416
	HEADER.GetDefaultBkColor()	416
	HEADER.GetDefaultBorderH()	417
	HEADER.GetDefaultBorderV()	417
	HEADER.GetDefaultCursor()	417
	HEADER.GetDefaultFont()	417
	HEADER.GetDefaultTextColor()	417
	HEADER.GetHeight()	418
	HEADER.GetItemWidth().....	418
	HEADER.GetNumItems().....	418
	HEADER.SetBitmap().....	418
	HEADER.SetBitmapEx()	419
	HEADER.SetBkColor()	419
	HEADER.SetBMP()	420
	HEADER.SetBMPEx()	420
	HEADER.SetDefaultBkColor()	420
	HEADER.SetDefaultBorderH()	421
	HEADER.SetDefaultBorderV()	421
	HEADER.SetDefaultCursor()	421
	HEADER.SetDefaultFont()	422
	HEADER.SetDefaultTextColor()	422
	HEADERSetFont()	422
	HEADER_SetHeight()	422
	HEADER_SetTextAlign()	423
	HEADER_SetItemText()	423
	HEADER_SetItemWidth()	423
	HEADER_SetStreamedBitmap()	424
	HEADER_SetStreamedBitmapEx()	424
	HEADER_SetTextColor()	424
15.9.5	Example	425

15.10	LISTBOX: List box widget	426
15.10.1	Configuration options.....	426
15.10.2	Notification codes.....	426
15.10.3	Keyboard reaction	426
15.10.4	LISTBOX API	427
	LISTBOX_AddString()	428
	LISTBOX_Create().....	428
	LISTBOX_CreateAsChild().....	428
	LISTBOX_CreateEx()	429
	LISTBOX_CreateIndirect()	429
	LISTBOX_DecSel()	430
	LISTBOX_DeleteItem()	430
	LISTBOX_GetDefaultBkColor()	430
	LISTBOX_GetDefaultFont()	430
	LISTBOX_GetDefaultScrollStepH()	431
	LISTBOX_GetDefaultTextAlign()	431
	LISTBOX_GetDefaultTextColor()	431
	LISTBOX_GetFont()	431
	LISTBOX_GetItemDisabled()	432
	LISTBOX_GetItemSel()	432
	LISTBOX_GetItemText()	432
	LISTBOX_GetMulti()	433
	LISTBOX_GetNumItems().....	433
	LISTBOX_GetScrollStepH()	433
	LISTBOX_GetSel().....	433
	LISTBOX_GetTextAlign()	434
	LISTBOX_IncSel()	434
	LISTBOX_InsertString()	434
	LISTBOX_InvalidateItem()	434
	LISTBOX_OwnerDraw().....	435
	LISTBOX_SetAutoScrollH()	435
	LISTBOX_SetAutoScrollV()	436
	LISTBOX_SetBkColor().....	436
	LISTBOX_SetDefaultBkColor()	436
	LISTBOX_SetDefaultFont()	437
	LISTBOX_SetDefaultScrollStepH()	437
	LISTBOX_SetDefaultTextAlign()	437
	LISTBOX_SetDefaultTextColor()	437
	LISTBOX_SetFont()	438
	LISTBOX_SetItemDisabled()	438
	LISTBOX_SetItemSel()	438
	LISTBOX_SetItemSpacing()	439
	LISTBOX_SetMulti().....	439
	LISTBOX_SetOwnerDraw()	439
	LISTBOX_SetScrollbarColor()	440
	LISTBOX_SetScrollbarWidth()	441
	LISTBOX_SetScrollStepH()	441
	LISTBOX_SetSel()	441
	LISTBOX_SetString()	441
	LISTBOX_SetTextAlign()	442
	LISTBOX_SetTextColor()	442
15.10.5	Examples	443
	Using the LISTBOX widget	443
	Using the LISTBOX widget with and without frames	444
15.11	LISTVIEW: Listview widget	448
15.11.1	Configuration options.....	449
15.11.2	Notification codes.....	449
15.11.3	Keyboard reaction	449
15.11.4	LISTVIEW API	450
	LISTVIEW_AddColumn()	450
	LISTVIEW_AddRow()	451

	LISTVIEW_Create().....	451
	LISTVIEW_CreateAttached()	452
	LISTVIEW_CreateEx()	452
	LISTVIEW_CreateIndirect()	453
	LISTVIEW_DecSel()	453
	LISTVIEW_DeleteColumn()	453
	LISTVIEW_DeleteRow()	453
	LISTVIEW_DisableRow()	454
	LISTVIEW_EnableRow()	454
	LISTVIEW_GetBkColor()	455
	LISTVIEW_GetFont()	455
	LISTVIEW_GetHeader()	455
	LISTVIEW_GetItemText()	456
	LISTVIEW_GetNumColumns()	456
	LISTVIEW_GetNumRows()	456
	LISTVIEW_GetSel()	456
	LISTVIEW_GetTextColor()	457
	LISTVIEW_IncSel()	457
	LISTVIEW_SetBkColor()	457
	LISTVIEW_SetColumnWidth()	458
	LISTVIEW_SetDefaultBkColor()	458
	LISTVIEW_SetDefaultFont()	458
	LISTVIEW_SetDefaultGridColor()	459
	LISTVIEW_SetDefaultTextColor()	459
	LISTVIEW_SetFont()	459
	LISTVIEW_SetGridVis()	460
	LISTVIEW_SetItemBkColor()	460
	LISTVIEW_SetItemText()	461
	LISTVIEW_SetItemTextColor()	461
	LISTVIEW_SetLBorder()	462
	LISTVIEW_SetRBorder()	462
	LISTVIEW_SetRowHeight()	462
	LISTVIEW_SetSel()	463
	LISTVIEW_Set.TextAlign()	463
	LISTVIEW_SetTextColor()	464
15.11.5	Example	464
15.12	MENU: Menu widget	465
15.12.1	Menu messages	466
	WM_MENU	466
15.12.2	Data structures	466
	MENU_ITEM_DATA	466
15.12.3	Configuration options	468
15.12.4	Keyboard reaction	468
15.12.5	MENU API	469
	MENU_AddItem()	469
	MENU_Attach()	470
	MENU_CreateEx()	470
	MENU_DeleteItem()	471
	MENU_DisableItem()	472
	MENU_EnableItem()	472
	MENU_GetDefaultBkColor()	473
	MENU_GetDefaultBorderSize()	473
	MENU_GetDefaultEffect()	473
	MENU_GetDefaultFont()	474
	MENU_GetDefaultTextColor()	474
	MENU_GetItem()	474
	MENU_GetItemText()	475
	MENU_GetNumItems()	475
	MENU_InsertItem()	475
	MENU_SetBkColor()	476
	MENU_SetBorderSize()	477

MENU_SetDefaultBkColor()	477
MENU_SetDefaultBorderSize()	478
MENU_SetDefaultEffect()	478
MENU_SetDefaultFont()	478
MENU_SetDefaultTextColor()	479
MENU_SetFont()	479
MENU_SetItem()	480
MENU_SetOwner()	480
MENU_SetTextColor()	480
15.13 MESSAGEBOX: Message box widget	482
15.13.1 Configuration options	482
15.13.2 Keyboard reaction	482
15.13.3 MESSAGEBOX API	482
GUI_MessageBox()	482
MESSAGEBOX_Create()	483
15.14 MULTIEDIT: Multi line text widget	484
15.14.1 Configuration options	485
15.14.2 Notification codes	485
15.14.3 Keyboard reaction	485
15.14.4 MULTIEDIT API	485
MULTIEDIT_AddKey()	486
MULTIEDIT_Create()	486
MULTIEDIT_CreateEx()	487
MULTIEDIT_GetPrompt()	488
MULTIEDIT.GetText()	488
MULTIEDIT.GetTextSize()	488
MULTIEDIT_SetAutoScrollH()	488
MULTIEDIT_SetAutoScrollV()	489
MULTIEDIT_SetBkColor()	489
MULTIEDIT_SetBufferSize()	490
MULTIEDIT_SetCursorOffset()	490
MULTIEDIT_SetFont()	490
MULTIEDIT_SetInsertMode()	490
MULTIEDIT_SetMaxNumChars()	491
MULTIEDIT_SetPasswordMode()	491
MULTIEDIT_SetPrompt()	491
MULTIEDIT_SetReadOnly()	492
MULTIEDIT_SetText()	492
MULTIEDIT_SetTextColor()	492
MULTIEDIT_SetWrapWord()	493
MULTIEDIT_SetWrapNone()	493
15.15 MULTIPAGE: Multiple page widget	494
15.15.1 Configuration options	495
15.15.2 Notification codes	495
15.15.3 Keyboard reaction	495
15.15.4 MULTIPAGE API	495
MULTIPAGE_AddPage()	496
MULTIPAGE_CreateEx()	496
MULTIPAGE_CreateIndirect()	497
MULTIPAGE_DeletePage()	497
MULTIPAGE_DisablePage()	497
MULTIPAGE_EnablePage()	498
MULTIPAGE_GetDefaultAlign()	498
MULTIPAGE_GetDefaultBkColor()	499
MULTIPAGE_GetDefaultFont()	500
MULTIPAGE_GetDefaultTextColor()	500
MULTIPAGE_GetSelection()	500
MULTIPAGE_GetWindow()	500
MULTIPAGE_IsPageEnabled()	501
MULTIPAGE_SelectPage()	501
MULTIPAGE_SetAlign()	501

	MULTIPAGE_SetBkColor()	502
	MULTIPAGE_SetDefaultAlign()	503
	MULTIPAGE_SetDefaultBkColor()	503
	MULTIPAGE_SetDefaultFont().....	503
	MULTIPAGE_SetDefaultTextColor().....	503
	MULTIPAGE_SetFont().....	504
	MULTIPAGE_SetText().....	504
	MULTIPAGE_SetTextColor()	505
15.16	PROGBAR: Progress bar widget	506
15.16.1	Configuration options	506
15.16.2	Keyboard reaction	506
15.16.3	PROGBAR API	506
	PROGBAR_Create()	506
	PROGBAR_CreateAsChild().....	507
	PROGBAR_CreateEx()	507
	PROGBAR_CreateIndirect()	508
	PROGBAR_SetBarColor()	508
	PROGBAR_SetFont()	508
	PROGBAR_SetMinMax().....	508
	PROGBAR_SetText()	509
	PROGBAR_Set.TextAlign()	509
	PROGBAR_SetTextPos()	509
	PROGBAR_SetValue().....	510
15.16.4	Examples	510
	Using the PROGBAR widget	510
	Advanced use of the PROGBAR widget	511
15.17	RADIO: Radio button widget	514
15.17.1	Configuration options	514
15.17.2	Notification codes	514
15.17.3	Keyboard reaction	515
15.17.4	RADIO API	515
	RADIO_Create()	515
	RADIO_CreateEx().....	516
	RADIO_CreateIndirect()	516
	RADIO_Dec()	517
	RADIO_GetDefaultFont()	517
	RADIO_GetDefaultTextColor()	517
	RADIO_GetText()	518
	RADIO_GetValue()	518
	RADIO_Inc()	518
	RADIO_SetBkColor()	519
	RADIO_SetDefaultFocusColor()	519
	RADIO_SetDefaultFont()	520
	RADIO_SetDefaultImage()	520
	RADIO_SetDefaultTextColor()	520
	RADIO_SetFocusColor()	521
	RADIO_SetFont()	521
	RADIO_SetGroupID()	522
	RADIO_SetImage()	522
	RADIO_SetText()	523
	RADIO_SetTextColor()	523
	RADIO_SetValue()	524
15.18	SCROLLBAR: Scroll bar widget	525
15.18.1	Configuration options	525
15.18.2	Notification codes	525
15.18.3	Keyboard reaction	525
15.18.4	SCROLLBAR API	525
	SCROLLBAR_AddValue()	526
	SCROLLBAR_Create()	526
	SCROLLBAR_CreateAttached()	527
	SCROLLBAR_CreateEx()	527

SCROLLBAR_CreateIndirect()	528
SCROLLBAR_Dec()	528
SCROLLBAR_GetDefaultWidth()	528
SCROLLBAR_GetValue()	529
SCROLLBAR_Inc()	529
SCROLLBAR_SetColor()	529
SCROLLBAR_SetDefaultColor()	530
SCROLLBAR_SetDefaultWidth()	530
SCROLLBAR_SetNumItems()	530
SCROLLBAR_SetPageSize()	531
SCROLLBAR_SetState()	531
SCROLLBAR_SetValue()	531
SCROLLBAR_SetWidth()	531
15.19 SLIDER: Slider widget	533
15.19.1 Configuration options	533
15.19.2 Notification codes	533
15.19.3 Keyboard reaction	533
15.19.4 SLIDER API	533
SLIDER_Create()	534
SLIDER_CreateEx()	534
SLIDER_CreateIndirect()	535
SLIDER_Dec()	535
SLIDER_GetValue()	535
SLIDER_Inc()	535
SLIDER_SetBkColor()	536
SLIDER_SetDefaultFocusColor()	536
SLIDER_SetFocusColor()	537
SLIDER_SetNumTicks()	537
SLIDER_SetRange()	537
SLIDER_SetValue()	538
SLIDER_SetWidth()	538
15.19.5 Example	539
15.20 TEXT: Text widget	540
15.20.1 Configuration options	540
15.20.2 Keyboard reaction	540
15.20.3 TEXT API	540
TEXT_Create()	540
TEXT_CreateAsChild()	541
TEXT_CreateEx()	541
TEXT_CreateIndirect()	542
TEXT_GetDefaultFont()	542
TEXT_SetBkColor()	542
TEXT_SetDefaultFont()	543
TEXT_SetDefaultTextColor()	543
TEXT_SetFont()	543
TEXT_SetText()	543
TEXT_SetTextAlign()	544
TEXT_SetTextColor()	544
15.21 WINDOW: Window widget	545
15.21.1 Configuration options	545
15.21.2 Keyboard reaction	545
15.21.3 WINDOW API	545
WINDOW_CreateIndirect()	545
WINDOW_SetDefaultBkColor()	545
16 Dialogs	547
16.1 Dialog basics	548
16.2 Creating a dialog	548
16.2.1 Resource table	548
16.2.2 Dialog procedure	549
16.2.2.1 Initializing the dialog	550

16.2.2.2	Defining dialog behavior	551
16.3	Dialog API	552
16.4	Dialog boxes	552
	GUI_CreateDialogBox()	552
	GUI_ExecCreatedDialog()	552
	GUI_ExecDialogBox()	552
	GUI_EndDialog()	553
17	Virtual screen / Virtual pages	555
17.1	Introduction	556
17.2	Requirements	556
17.3	Configuration	557
	LCD_SET_ORG	557
	LCD_VXSIZE, LCD_VYSIZE	557
17.3.1	Sample configuration	557
17.4	Samples	558
17.4.1	Basic sample	558
17.4.2	Real time sample using the window manager	559
17.4.3	Dialog sample using the window manager	560
17.5	Virtual screen API	562
	GUI_GetOrg()	562
	GUI_SetOrg()	562
18	Multi layer / multi display support	563
18.1	Introduction	564
18.1.1	Limitations	564
18.1.2	Selecting a layer for drawing operations	564
18.1.3	Selecting a layer for a window	564
18.1.3.1	Moving a window from one layer to an other	565
18.2	Using multi layer support	567
18.2.1	Using the viewer with multiple layers	567
18.2.2	Using transparency	567
18.2.3	Multi layer sample	567
18.2.4	Pitfalls	567
18.3	Using multi display support	568
18.3.1	Enabling multi display support	568
18.3.2	Multi display sample	568
18.4	Configuring multi layer support	568
18.5	Configuring multi display support	569
18.6	Multi layer API	571
	GUI_SelectLayer()	571
	GUI_SetLUTColorEx()	571
	LCD_GetNumDisplays()	571
19	Pointer Input Devices	573
19.1	Description	574
19.2	Pointer input device API	575
	GUI_PID_GetState()	575
	GUI_PID_StoreState()	575
19.3	Mouse driver	577
19.3.1	Generic mouse API	577
	GUI_MOUSE_GetState()	577
	GUI_MOUSE_StoreState()	577
19.3.2	PS2 mouse driver	579
19.3.2.1	Using the PS2 mouse driver	579
19.3.2.2	PS2 mouse driver API	579
	GUI_MOUSE_DRIVER_PS2_Init()	579
	GUI_MOUSE_DRIVER_PS2_OnRx()	579
19.4	Touch-screen drivers	580
19.4.1	Generic touch-screen API	580

GUI_TOUCH_GetState()	580
GUI_TOUCH_StoreState()	580
GUI_TOUCH_StoreStateEx()	581
19.4.2 The analog touch screen driver	581
19.4.2.1 Setting up the analog touch screen	581
19.4.2.2 Runtime calibration	583
19.4.2.3 Hardware routines	584
GUI_TOUCH_X_ActivateX(), GUI_TOUCH_X_ActivateY()	584
GUI_TOUCH_X_MeasureX(), GUI_TOUCH_X_MeasureY()	584
19.4.2.4 Driver API for analog touch-screens	585
GUI_TOUCH_Calibrate()	585
GUI_TOUCH_Exec()	586
GUI_TOUCH_SetDefaultCalibration()	586
19.4.2.5 Configuring the analog touch-screen driver	586
19.5 Joystick input sample	588
20 Keyboard Input	589
20.1 Description	590
20.1.1 Driver layer API	590
GUI_StoreKeyMsg()	590
GUI_SendKeyMsg()	591
20.1.2 Application layer API	591
GUI_ClearKeyBuffer()	591
GUI_GetKey()	592
GUI_StoreKey()	592
GUI_WaitKey()	592
21 Cursors	593
21.1 Available cursors	594
21.2 Cursor API	594
GUI_CURSOR_Hide()	594
GUI_CURSOR_Select()	595
GUI_CURSOR_SetPosition()	595
GUI_CURSOR_Show()	595
22 Antialiasing	597
22.1 Introduction	598
22.1.1 Quality of antialiasing	598
22.1.2 Antialiased Fonts	598
22.1.3 High-resolution coordinates	599
22.2 Antialiasing API	600
22.3 Control functions	600
GUI_AA_DisableHiRes()	600
GUI_AA_EnableHiRes()	600
GUI_AA_GetFactor()	601
GUI_AA_SetFactor()	601
22.4 Drawing functions	601
GUI_AA_DrawArc()	601
GUI_AA_DrawLine()	602
GUI_AA_DrawPolyOutline()	602
GUI_AA_DrawPolyOutlineEx()	603
GUI_AA_FillCircle()	603
GUI_AA_FillPolygon()	604
22.5 Examples	605
Different antialiasing factors	605
Lines placed on high-resolution coordinates	606
Moving pointer using high-resolution antialiasing	607
23 Foreign Language Support	611

23.1	Unicode.....	612
23.1.1	UTF-8 encoding	612
23.1.2	Unicode characters	612
23.1.3	UTF-8 strings	613
23.1.3.1	Using U2C.exe to convert UTF-8 text into "C"-code	613
23.1.4	Unicode API	614
23.1.4.1	UTF-8 functions	614
	GUI_UC_ConvertUC2UTF8()	614
	GUI_UC_ConvertUTF82UC()	614
	GUI_UC_Encode()	615
	GUI_UC_GetCharCode()	615
	GUI_UC_GetCharSize()	615
	GUI_UC_SetEncodeNone()	616
	GUI_UC_SetEncodeUTF8()	616
23.1.4.2	Double byte functions	616
	GUI_UC_DispString()	616
23.2	Arabic language support.....	618
23.2.1	Basics	618
23.2.2	Notation forms	618
23.2.3	Ligatures	619
23.2.4	How to enable Arabic support.....	619
23.2.5	Text alignment.....	620
23.2.6	Font files used with Arabic text.....	620
23.3	Shift JIS support	621
23.3.1	Creating Shift JIS fonts	621
23.3.2	Example	621
24	Display drivers	625
24.1	Available drivers and supported display controllers.....	626
24.2	CPU / Display controller interface.....	629
24.2.1	Full bus interface	629
24.2.2	Simple bus interface	630
24.2.3	4 pin SPI interface	630
24.2.4	3 pin SPI interface	631
24.2.5	I2C bus interface	631
24.3	Detailed display driver descriptions	633
24.3.1	LCDColorOnMono driver.....	633
	Supported hardware	633
	Display data RAM organization	633
	RAM requirements of the driver	633
	Additional driver functions	634
	Hardware configuration	634
	Additional configuration switches	634
	Special requirements	634
24.3.2	LCDFujitsu driver.....	635
	Supported hardware	635
	Display data RAM organization	635
	RAM requirements of the driver	635
	Additional driver functions	635
	Hardware configuration	635
	Additional configuration switches	636
	Special requirements	636
24.3.3	LCDLin driver (8 and 16 bit access).....	637
	Supported hardware	637
	Display data RAM organization	638
	Additional RAM requirements of the driver	638
	Additional driver functions	638
	Hardware configuration	638
	Additional configuration switches	639
	Additional info for S1D13A03, S1D13A04 and S1D13A05	639
	Additional info for S1D13806, S1D13A03, S1D13A04 and S1D13A05	639

24.3.4	LCDLin driver (32 bit access)	640
	Supported hardware	640
	Display data RAM organization	641
	Additional RAM requirements of the driver	641
	Additional driver functions	641
	Hardware configuration	642
	Additional configuration switches	642
	LCD_LIN_SWAP()	642
	LCD_FILL_RECT().....	643
	How to migrate from LCDLin to LCDLin32	643
24.3.5	LCDMem driver	644
	Using the CPU as LCD controller	644
	How to connect the CPU to the row/column drivers	644
	CPU load	644
	Supported hardware	645
	Display data RAM organization	645
	RAM requirements of the driver	645
	Additional driver functions	645
	Hardware configuration	645
	Additional configuration switches	646
24.3.6	LCDMemC driver	647
	Supported hardware	647
	Display data RAM organization	648
	RAM requirements of the driver	649
	Additional driver functions	649
	Hardware configuration	649
	Additional configuration switches	649
24.3.7	LCDNoritake display driver	650
	Supported hardware	650
	Display data RAM organization	650
	Additional RAM-requirement	650
	Additional driver functions	650
	Hardware configuration	650
24.3.8	LCDPage1bpp driver	651
	Supported hardware	651
	Display data RAM organization	651
	Additional RAM requirements of the driver	652
	Additional driver functions	652
	Hardware configuration	652
	Additional configuration switches	652
	Special requirements for certain LCD controllers	653
24.3.9	LCDPage4bpp driver	654
	Supported hardware	654
	Display data RAM organization	654
	Additional RAM requirements of the driver	654
	Additional driver functions	655
	Hardware configuration	655
	Additional configuration switches	655
24.3.10	LCDSLin driver	656
	Supported hardware	656
	Display data RAM organization	656
	Additional RAM-requirement	656
	Additional driver functions	656
	Hardware configuration	657
	Additional configuration switches for SED 1330, SED 1335	657
	Additional info for SED 1330, SED 1335	657
24.3.11	LCDVesa driver	658
	Supported hardware	658
	Display data RAM organization	658
	RAM requirements of the driver	658
	Additional driver functions	658

	Hardware configuration	658
	Additional configuration switches	658
	Special requirements	658
24.3.12	LCDXylon driver	659
	Supported hardware	659
	Display data RAM organization	659
	Additional RAM requirements of the driver	659
	Additional driver functions	659
	Hardware configuration	659
	Additional configuration switches	660
24.3.13	LCD0323 driver	661
	Supported hardware	661
	Display data RAM organization	661
	Additional RAM requirements of the driver	661
	Additional driver functions	661
	Hardware configuration	661
	Additional configuration switches	662
	Special requirements for certain LCD controllers	662
24.3.14	LCD07X1 driver	663
	Supported hardware	663
	Display data RAM organization	663
	Additional RAM requirements of the driver	663
	Additional driver functions	664
	Hardware configuration	664
	Additional configuration switches	664
	Special requirements for certain LCD controllers	664
24.3.15	LCD1200 driver	665
	Supported hardware	665
	Display data RAM organization	665
	Additional RAM requirements	665
	Additional driver functions	665
	Hardware configuration	666
	Additional configuration switches	666
	Special requirements	666
24.3.16	LCD13701 driver	667
	Supported hardware	667
	Display data RAM organization	667
	Additional RAM requirements of the driver	668
	Additional driver functions	668
	Hardware configuration	668
	Additional configuration switches	668
	Special requirements for certain LCD controllers	668
24.3.17	LCD159A driver	669
	Supported hardware	669
	Display data RAM organization	669
	Additional RAM requirements	669
	Additional driver functions	669
	Hardware configuration	669
	Additional configuration switches	669
	Special requirements for certain LCD controllers	669
24.3.18	LCD15E05 driver	670
	Supported hardware	670
	Display data RAM organization	670
	Additional RAM requirements of the driver	670
	Additional driver functions	670
	Hardware configuration	671
	Additional configuration switches	671
	Special requirements for certain LCD controllers	671
24.3.19	LCD1611 driver	672
	Supported hardware	672
	Display data RAM organization	672

	Additional RAM requirements of the driver	672
	Additional driver functions	673
	Hardware configuration	673
	Additional configuration switches	673
	Special requirements for certain LCD controllers	673
24.3.20	LCD161620 driver	674
	Supported hardware	674
	Display data RAM organization	674
	Additional RAM requirements of the driver	674
	Additional driver functions	674
	Hardware configuration	674
	Additional configuration switches	675
	Special requirements	675
24.3.21	LCD1781 driver	676
	Supported hardware	676
	Display data RAM organization	676
	Additional RAM requirements	676
	Additional driver functions	676
	Hardware configuration	677
	Additional configuration switches	677
	Special requirements	677
24.3.22	LCD501 driver	678
	Supported hardware	678
	Display data RAM organization	678
	Additional RAM requirements	678
	Additional driver functions	678
	Hardware configuration	678
	Additional configuration switches	678
	Special requirements	678
24.3.23	LCD6331 driver	680
	Supported hardware	680
	Display data RAM organization	680
	Additional RAM requirements	680
	Additional driver functions	680
	Hardware configuration	681
	Additional configuration switches	681
	Special requirements	681
24.3.24	LCD6642X driver	682
	Supported hardware	682
	Display data RAM organization	682
	Additional RAM requirements of the driver	682
	Additional driver functions	682
	Hardware configuration	682
	Additional configuration switches	683
	Special requirements	683
24.3.25	LCD667XX driver	684
	Supported hardware	684
	Display data RAM organization	684
	Additional RAM requirements	684
	Additional driver functions	684
	Hardware configuration	685
	Additional configuration switches	685
	Special requirements	685
24.3.26	LCD66750 driver	686
	Supported hardware	686
	Display data RAM organization	686
	Additional RAM requirements of the driver	686
	Additional driver functions	686
	Hardware configuration	686
	Additional configuration switches	687
	Special requirements for certain LCD controllers	687

24.3.27	LCDTemplate driver	688
	Adapting the template driver.....	688
24.3.28	LCDNull driver.....	689
	Using this driver.....	689
24.4	LCD layer and display driver API.....	690
24.4.1	Display driver API.....	690
24.4.2	Init & display control group.....	691
	LCD_L0_Init()	691
	LCD_L0_ReInit().....	692
	LCD_L0_Off(), LCD_L0_On()	692
24.4.3	Drawing group	692
	LCD_L0_DrawBitmap().....	692
	LCD_L0_DrawHLine()	693
	LCD_L0_DrawPixel()	693
	LCD_L0_DrawVLine()	693
	LCD_L0_FillRect()	693
	LCD_L0_SetPixelIndex()	694
	LCD_L0_XorPixel()	694
24.4.4	"Get" group	694
	LCD_L0_GetPixelIndex().....	694
24.4.5	Lookup table (LUT) group	695
	LCD_L0_SetLUTEntry().....	695
24.4.6	Miscellaneous group.....	695
	LCD_L0_ControlCache()	695
	LCD_GetBitsPerPixel()	696
	LCD_GetBitsPerPixelEx().....	696
	LCD_GetFixedPalette()	696
	LCD_GetFixedPaletteEx()	696
	LCD_GetNumColors()	697
	LCD_GetNumColorsEx().....	697
	LCD_GetVXSize(), LCD_GetVYSize()	697
	LCD_GetVXSizeEx(), LCD_GetVYSizeEx()	697
	LCD_GetXMag(), LCD_GetYMag()	698
	LCD_GetXMagEx(), LCD_GetYMagEx()	698
	LCD_GetXSize(), LCD_GetYSize()	698
	LCD_GetXSizeEx(), LCD_GetYSizeEx()	698
25	VNC server	699
25.1	Introduction	700
25.1.1	Requirements.....	700
25.1.2	Notes on this implementation	700
25.2	The VNC viewer.....	701
25.2.1	Starting the VNC viewer	701
25.3	emWin VNC server.....	702
25.3.1	Starting the emWin VNC server	702
25.3.2	How the server starts	702
25.3.3	Integration of the VNC server on the target	702
25.4	Required resources	703
25.5	Configuration options	703
25.6	VNC API	703
	GUI_VNC_AttachToLayer()	703
	GUI_VNC_Process()	704
	GUI_VNC_X_StartServer()	705
26	Timing and Execution-Related Functions	707
26.1	Timing and execution API	708
	GUI_Delay()	708
	GUI_Exec().....	708
	GUI_Exec1()	708
	GUI_GetTime().....	709

27	Low-Level Configuration (LCDConf.h)	711
27.1	Available configuration macros	712
27.2	General (required) configuration	714
	LCD_CONTROLLER	714
	LCD_BITSPERPIXEL	714
	LCD_FIXEDPALETTE	714
	LCD_XSIZE, LCD_YSIZE	714
27.3	Initialisation of the controller	716
	LCD_INIT_CONTROLLER	716
27.4	Display orientation	717
	LCD_MIRROR_X	718
	LCD_MIRROR_Y	718
	LCD_SWAP_XY	719
27.5	Color configuration	720
	LCD_MAX_LOG_COLORS	720
	LCD_PHYSCOLORS	720
	LCD_PHYSCOLORS_IN_RAM	720
	LCD_REVERSE	720
	LCD_SET_LUT_ENTRY	720
	LCD_SWAP_RB	721
27.6	Magnifying the LCD	721
	LCD_XMAG	721
	LCD_YMAG	721
27.7	Simple bus interface configuration	722
27.7.1	Macros used by a simple bus interface	722
	LCD_READ_A0	722
	LCD_READ_A1	722
	LCD_WRITE_A0	722
	LCD_WRITE_A1	722
	LCD_WRITEITEM_A1	723
27.7.2	Example of memory mapped interface	723
27.7.3	Sample routines for connection to I/O pins	723
27.8	3 pin SPI configuration	725
27.8.1	Macros used by a 3 pin SPI interface	725
	LCD_WRITE	725
	LCD_WRITEITEM	725
27.8.2	Sample routines for connection to I/O pins	725
27.9	4 pin SPI configuration	726
27.9.1	Macros used by a 4 pin SPI interface	726
	LCD_WRITE_A0	726
	LCD_WRITE_A1	726
	LCD_WRITEITEM_A1	726
27.9.2	Sample routines for connection to I/O pins	726
27.10	I2C bus interface configuration	728
27.10.1	Macros used by a I2C bus interface	728
	LCD_READ_A0	728
	LCD_READ_A1	728
	LCD_WRITE_A0	728
	LCD_WRITE_A1	728
	LCD_WRITEITEM_A1	729
27.10.2	Sample routines for connection to I/O pins	729
27.11	Full bus interface configuration	730
27.11.1	Macros used by a full bus interface	730
	LCD_READ_MEM	730
	LCD_READ_REG	730
	LCD_WRITE_MEM	731
	LCD_WRITE_REG	731
	LCD_BUSWIDTH	731
	LCD_ENABLE_REG_ACCESS	732
	LCD_ENABLE_MEM_ACCESS	732
	LCD_SWAP_BYTE_ORDER	732

27.11.2	Configuration example	732
27.12	Hardware access for multiple LCD controllers.....	734
27.13	Virtual display support.....	735
	LCD_VXSIZE, LCD_VYSIZE	735
	LCD_XORG<n>, LCD_YORG<n>.....	735
27.14	LCD controller configuration: common/segment lines	736
	LCD_XORG<n>, LCD_YORG<n>.....	736
	LCD_FIRSTSEG<n>	736
	LCD_LASTSEG<n>	736
	LCD_FIRSTCOM<n>	736
	LCD_LASTCOM<n>	736
27.14.1	Single LCD controller configuration.....	737
27.14.2	Configuring additional LCD controllers	737
27.15	COM/SEG lookup tables	740
	LCD_LUT_COM	740
	LCD_LUT_SEG.....	740
27.16	Miscellaneous.....	741
	LCD_NUM_CONTROLLERS.....	741
	LCD_CACHE	741
	LCD_USE_BITBLT	741
	LCD_SUPPORT_CACHECONTROL.....	741
	LCD_TIMERINIT0.....	742
	LCD_TIMERINIT1.....	742
	LCD_ON	742
	LCD_OFF	742
28	High-Level Configuration (GUIConf.h)	743
28.1	General notes	744
28.2	How to configure the GUI	744
28.2.1	Sample configuration	744
28.3	Available GUI configuration macros	744
28.3.1	GUI_ALLOC_SIZE	745
28.3.2	GUI_MAXBLOCKS	746
28.3.3	GUI_TRIAL_VERSION	746
28.4	GUI_X routine reference	746
28.5	Init routine	747
	GUI_X_Init().....	747
28.6	Timing routines	747
	GUI_X_Delay().....	747
	GUI_X_ExecIdle().....	747
	GUI_X_GetTime()	747
28.7	Kernel interface routines.....	748
28.8	Debugging.....	748
	GUI_X_ErrorOut(), GUI_X_Warn(), GUI_X_Log()	748
28.9	Dynamic memory	748
	GUI_ALLOC_ALLOC.....	749
	GUI_ALLOC_FREE.....	749
	GUI_ALLOC_GETMAXSIZE.....	749
	GUI_ALLOC_H2P	749
	GUI_HMEM	750
	Example	750
28.10	Special considerations for certain Compilers/CPUs.....	750
28.10.1	AVR with IAR-Compiler.....	750
	GUI_UNI_PTR	750
	GUI_CONST_STORAGE.....	751
28.10.2	8051 Keil compiler and other 8-bit CPU compilers	751
	GUI_COMPILER_SUPPORTS_FP.....	751
29	Performance and Resource Usage.....	753
29.1	Performance benchmark	754

29.2	Configuration and performance table	754
29.3	Memory requirements of the GUI on ARM7	756
29.3.1	Memory requirements of typical applications	757
29.3.1.1	Minimum project ("Hello world").....	757
29.3.1.2	Configuration.....	757
29.3.1.2	System	757
29.3.1.3	Memory requirements.....	757
29.3.2	Window application.....	758
29.3.2.1	Configuration.....	758
29.3.2.2	System	758
29.3.2.3	Memory requirements.....	758
30	Support	759
30.1	Problems with tool chain (compiler, linker)	760
30.1.1	Compiler crash.....	760
30.1.2	Compiler warnings.....	760
30.1.3	Compiler errors.....	760
30.1.4	Linker problems	761
30.2	Problems with hardware/driver	761
30.3	Problems with API functions.....	761
30.4	Problems with the performance.....	762
30.5	Contacting support	762
30.6	FAQ's	763

Chapter 1

Introduction to emWin

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.

1.2 Assumptions

This guide assumes that you already possess a solid knowledge of the "C" programming language. If you feel that your knowledge of "C" is not sufficient, we recommend *The "C" Programming Language* by Kernighan and Richie, which describes the programming standard and, in newer editions, also covers the ANSI "C" standard. Knowledge of assembly programming is not required.

1.3 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.

1.3.1 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 LCD (any type and any resolution)

The 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 apply to typical systems.

Small systems (no window manager)

- RAM: 100 bytes
- Stack: 500 bytes
- ROM: 10-25 kb (depending on the functionality used)

Big systems (including window manager and widgets)

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

Note that ROM requirements will increase if your application uses many fonts. All values are rough estimates and cannot be guaranteed.

1.3.2 Development environment (compiler)

The CPU used is of no importance; only an ANSI-compliant "C" compiler covering the international standard ISO/IEC 9899:1990, ISO/IEC 9899:1999 or ANSI/ISO 9899:1990 is required. If your compiler has some limitations, please 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.

1.4 Features

emWin is designed to provide an efficient, processor- and LCD controller-independent graphical user interface for any application that operates with a graphical LCD. 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 LCD controller and CPU. Its features include the following:

General

- Any 8/16/32-bit CPU; only an ANSI "C" compiler covering the international standard ISO/IEC 9899:1990, ISO/IEC 9899:1999 or ANSI/ISO 9899:1990 is required.
- Any (monochrome, grayscale or color) LCD with any controller supported (if the right driver is available).
- May work without LCD controller on smaller displays.
- Any interface supported using configuration macros.
- Display-size configurable.
- Characters and bitmaps may be written at any point on the LCD, not just on

- even-numbered byte addresses.
- Routines are optimized for both size and speed.
- Compile time switches allow for different optimizations.
- For slower LCD controllers, LCD 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 30: "Standard 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.
- Fonts are fully scalable, separately in X and Y.
- Font converter available; any font available on your host system (i.e. Microsoft Windows) can be converted.

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. 20 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 plus viewer.
- Bitmap converter.
- Font converter.

1.5 Samples 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 `Sample\EXE`. The source of the sample programs is located in the folder `Sample`. The folder `Sample\GUIDemo` contains an application program showing most of emWin's features. All samples are also available at www.segger.com.

1.6 Starter kits

Complete starter kits including a demoboard with LCD, a "C" compiler and a sample project are available. They have been designed primarily to test and demonstrate emWin, and they can be used to get familiar with the software.

Starter kit "Andy"

- Starterboard with Mitsubishi M30624 CPU and Samsung KS0713 LCD controller (including schematic and documentation).
- Monochrome LCD (128*64 pixel).



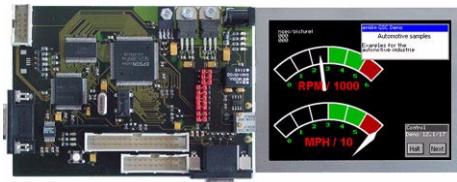
Starter kit "Armin"

- Starterboard with Mitsubishi M30803 CPU and SED 13705 LCD controller (including schematic and documentation).
- LCD (320*240 pixel) either monochrome, 1/4 VGA color display or TFT.



Starter kit "Axel"

- Starterboard with NEC V850SB1 CPU and SED 13806 LCD controller (including schematic and documentation).
- LCD (320*240 pixel) either monochrome, 1/4 VGA color display, TFT or none (VGA output).



Starter kit "Berni"

- Starterboard with Mitsubishi M30624 CPU (including schematic and documentation).
- **No LCD controller** -- intended for use with the LCDMem driver.
- Monochrome, yellow backlit LCD (160*100 pixel).



For more details, please take a look at our website at www.segger.com.

1.7 How to use this manual

This manual explains how to install, configure and use emWin. It describes the internal structure of the software and all the functions that emWin offers (the Application Program Interface, or API). 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 then recommended:

- Copy the emWin files to your computer.
- Go through Chapter 2: "Getting Started".
- Use the simulator in order to become more familiar with what the software can

- do (refer to Chapter 3: "Simulator").
- Expand your program using the rest of the manual for reference.

1.8 Typographic conventions for syntax

This manual uses the following typographic conventions:

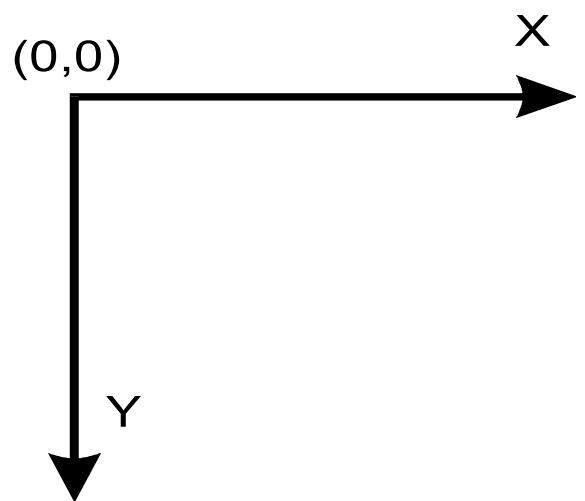
Style	Used for
Body	Body text.
Keyword	Text that you enter at the command-prompt or that appears on the display (i.e. system functions, file- or pathnames).
Parameter	Parameters in API functions.
Sample	Sample code in program examples.
New Sample	Sample code that has been added to an existing program example.
Warning	Important cautions or reminders.

1.9 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.



1.10 How to connect the LCD to the microcontroller

emWin handles all access to the LCD. Virtually any LCD controller can be supported, independently of how it is accessed. For details, please refer to Chapter 27: "Low-Level Configuration". Also, please get in contact with us if your LCD controller is not supported. We are currently writing drivers for all LCD controllers available on the market and may already have a proven driver for the LCD controller that you intend to use. It is usually very simple to write the routines (or macros) used to access the LCD in your application. SEGGER Microcontroller Systeme GmbH offers the service of making these customizations for you, if necessary with your target hardware.

It does not really matter how the LCD 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 explained in Chapter 24: "LCD Drivers". The most common ways to access the LCD are described as follows. If you simply want to understand how to use emWin, you may skip this section.

LCD with memory-mapped LCD controller:

The LCD 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 LCD controller and is most recommended. The LCD addresses are defined to the segment LCDSEG, and in order to be able to access the LCD 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 LCD controllers.

LCD with LCD controller connected to port / buffer

For slower LCD controllers used on fast processors, the use of port-lines may be the only solution. This method of accessing the LCD has the disadvantage of being somewhat slower than direct bus-interface but, particularly with a cache that minimizes the accesses to the LCD, the LCD 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 LCD is connected to. This type of interface is also supported by different drivers for the different LCD controllers.

Proprietary solutions: LCD without LCD controller

The LCD can also be connected without an LCD controller. In this case, the LCD 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 LCD driver because emWin simply places all the display data into the LCD cache. You yourself must write the hardware-dependent portion that periodically transfers the data in the cache memory to your LCD.

Sample code for transferring the video image into the display is available in both "C" and optimized assembler for M16C and M16C/80.

1.11 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	Explanation
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

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 configuration file `LCDCConf.h`.

Chapter 2

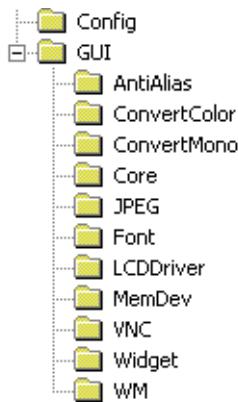
Getting Started

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 directory 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.



2.1.1 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\JPEG	JPEG support (Independent JPEG Group's free JPEG software)
GUI\Font	Font files
GUI\LCDDriver	LCD driver
GUI\MemDev	Memory device support *
GUI\VNC	VNC support *
GUI\Widget	Widget library *
GUI\WM	Window manager *

(* = optional)

2.1.2 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\Widget (if using 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 (or at least renaming) the `GUI\` directories prior to updating.

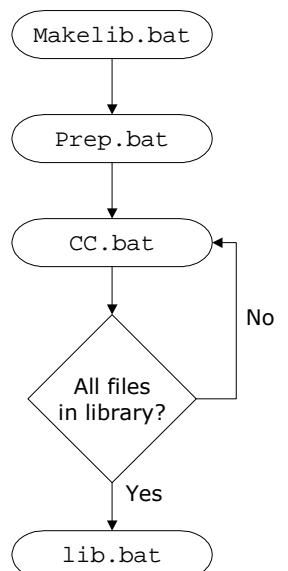
2.2 Adding emWin to the target program

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 sample projects available to help you get started.

2.3 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 root directory. 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 another way.



File	Explanation
<code>Makelib.bat</code>	Main batch file. No modification required.
<code>Prep.bat</code>	Called by <code>Makelib.bat</code> to prepare environment for the tool chain to be used,
<code>CC.bat</code>	Called by <code>Makelib.bat</code> 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 <code>lib.bat</code> file.
<code>lib.bat</code>	Called by <code>Makelib.bat</code> to put the object files listed by <code>CC.bat</code> into a library.

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`.

2.3.1 Adapting the library batch files to a different system

The following will show how to adapt the files by a sample 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	<code>MakeLib.bat</code> uses this folder to copy all source and header files used for the build process. Will be deleted after the build process.

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 a sample 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
-fFFRAM : 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 -fFFRAM -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 a sample 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

```

2.4 "C" files to include in the project

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

- All "C" files of the folder GUI\Core
- The fonts you plan to use (located in GUI\Font)
- LCD driver: All "C" files of the folder GUI\LCDDriver.
- All "C" files of the folder GUI\JPEG (only if you need JPEG support)

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

- GUI_X.c. A sample file is available as Sample\GUI_X. This file contains the hardware-dependent part of emWin and should be modified as described in Chapter 28: "High-Level Configuration".

Be sure that you include GUI.h in all of your source files that access emWin.

2.5 Configuring emWin

The Config folder should contain the configuration files matching your order. The file `LCDConf.h` normally contains all the definitions necessary to adopt emWin to your LCD, which is the main task when configuring emWin. For details, please see Chapter 27: "Low-Level Configuration".

If emWin is not configured correctly, because you did not select the right display resolution or chose the wrong LCD controller, it will probably not display anything at all or display something that does not resemble what you expected. So take care to tailor `LCDConf.h` to your needs. If you do not wish to deal with this process, SEGGER Microcontroller Systeme GmbH can do it for you, as well as test emWin in your application with your hardware and your LCD.

The following types of configuration macros exist:

Binary switches "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 values "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 an LCD.

Selection switches "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 LCD controller used, where the number selected denotes which source code (in which LCD driver) is used to generate object code.

Alias "A"

A macro which operates like a simple text substitute. An example would be the define `U8`, in which the preprocessor would replace with `unsigned char`.

Function replacements "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 an LCD) which is highly hardware-dependent. This type of macro is always declared using brackets (and optional parameters).

2.6 Initializing emWin

The routine `GUI_Init()` initializes the LCD and the internal data structures of emWin, and must be called before any other emWin function. This is done by placing the following line into the init sequence of your program:

```
GUI_Init();
```

If this call is left out, the entire graphics system will not be initialized and will therefore not be ready.

2.7 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: Customizing emWin

The first step is usually to customize emWin by modifying the header file `LCDConf.h`. You must define the basic data types (`U8`, `U16`, etc.) and mandatory configuration switches regarding resolution of the display and the LCD controller used.

Step 2: Defining access addresses or access routines

For memory-mapped LCDs, the access addresses of the LCD simply need to be defined in `LCDConf.h`. For port/buffer-accessed LCDs, interface routines must be defined. Samples of the required routines are available under `Samples\LCD_X` or on our website in the download area.

Step 3: Compiling, linking and testing the sample code

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

Step 4: Modifying the sample program

Make simple modifications to the sample 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 sample adaptations, please refer to Chapter 28: "High-Level Configuration".

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.

2.8 The "Hello world" sample program

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. A "Hello world" program with emWin, called `HELLO.c`, is shown below and is available as `BASIC_HelloWorld.c` in the sample 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 LCD and the GUI must first be initialized. emWin is initialized by a call to `GUI_Init()` at the start of the program, as described previously. In this example, we assume that the hardware of your application is already initialized.

The "Hello world" program looks as follows:

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample code
*
*****
```

```
-----
File      : BASIC_HelloWorld.c
Purpose   : Simple demo drawing "Hello world"
-----
*/
```

```
#include "GUI.H"

*****
*
*          main
*
*****
```

```
void MainTask(void) {
/*
 ToDo: Make sure hardware is initialized first!!
*/
    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 LCD 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.

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample code
*
*****
```

```
-----
File      : BASIC_Hello1.c
Purpose   : Simple demo drawing "Hello world"
-----
*/
```

```
#include "GUI.H"

*****
*
*          main
*
*****
```

```
void MainTask(void) {
    int i=0;
/*
 ToDo: Make sure hardware is initialized first!!
*/
    GUI_Init();
    GUI_DispString("Hello world!");
    while(1) {
        GUI_DispDecAt( i++, 20,20,4);
        if (i>9999) i=0;
    }
}
```

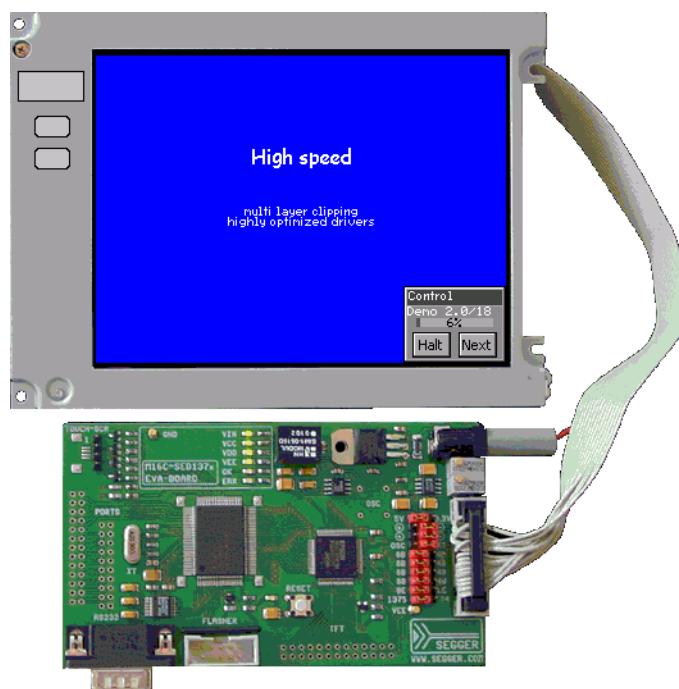
Chapter 3

Simulator

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 need for hardware!).
- Debugging of your user interface program.
- Creation of demos of your application, which can be used to discuss the user interface.

The resulting executable can be easily sent via email.



3.1 Using the simulator

The emWin simulator uses 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 will have 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 LCD driver. Instead of using the actual LCD 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.

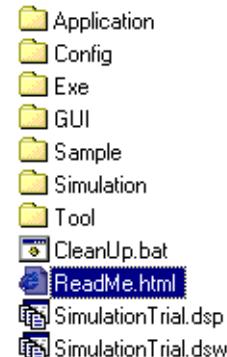
3.1.1 Using the simulator in the trial emWin version

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 change any configuration settings or view the source code, but you will still be able to become familiar with what emWin can do.

3.1.1.1 Directory structure

The directory structure of the simulator in the trial version will appear as pictured to the right. The table below explains the contents of the folders:

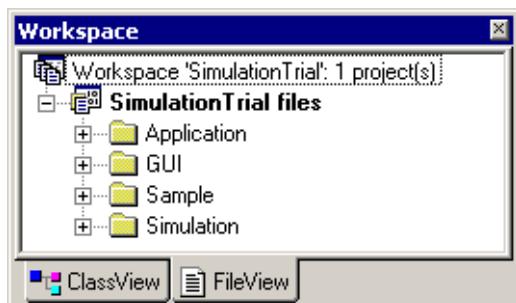
Directory	Contents
Application	Source of the demo program.
Config	configuration files used to build the library. Do not make any changes to these files!
Exe	Ready-to-use demo program.
GUI	Library files and include files needed to use the library.
Sample	Simulation samples and their sources.
Simulation	Files needed for the simulation.
Tool	The emWin viewer, a demo version of the bitmap converter and a demo version of the font converter.



3.1.1.2 Visual C++ workspace

The root directory shown above includes the Microsoft Visual C++ workspace (SimulationTrial.dsw) and project file (SimulationTrial.dsp). 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.



3.1.1.3 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 need only to rebuild and start it. Please note that to rebuild the executable, you will need to have Microsoft Visual C++ (version 6.00 or later) installed.

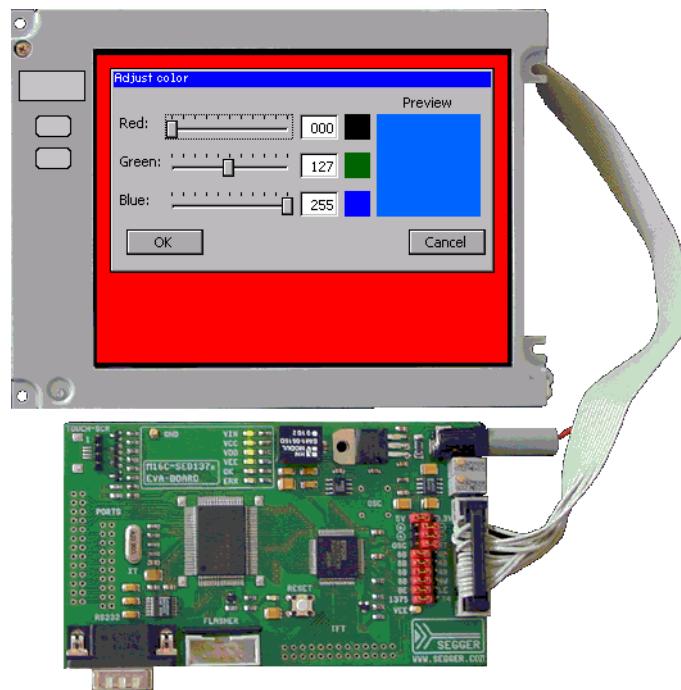
- Step 1: Open the Visual C++ workspace by double-clicking on Simulation-Trial.dsw.
- Step 2: Rebuild the project by choosing Build/Rebuild All from the menu (or by pressing F7).
- Step 3: Start the simulation by choosing Build/Start Debug/Go from the menu (or by pressing F5).

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

3.1.1.4 Compiling the samples

The Sample directory contains ready-to-go samples 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 sample which should be used by right-clicking on it and deselecting 'Settings\General\Exclude from build'. The screenshot below shows the sample DIALOG_SliderColor.c.
- Step 3: Rebuild the sample by choosing Build/Rebuild All from the menu (or by pressing F7).
- Step 4: Start the simulation by choosing Build/Start Debug/Go from the menu (or by pressing F5). The result of the sample selected above is pictured below:

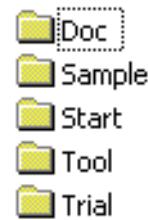


3.1.2 Using the simulator with the emWin source

3.1.2.1 Directory structure

The root directory of the simulator can be anywhere on your PC, e.g. 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 <\$chapttext: "Getting Started" for more information).

The following table shows the contents of the folders:



Directory	Contents
Doc	Contains emWin-Documentation.
Sample	Code samples, described later in this documentation.
Start	All you need to create a new project with emWin
Tool	Tools shipped with emWin
Trial	Complete trial version

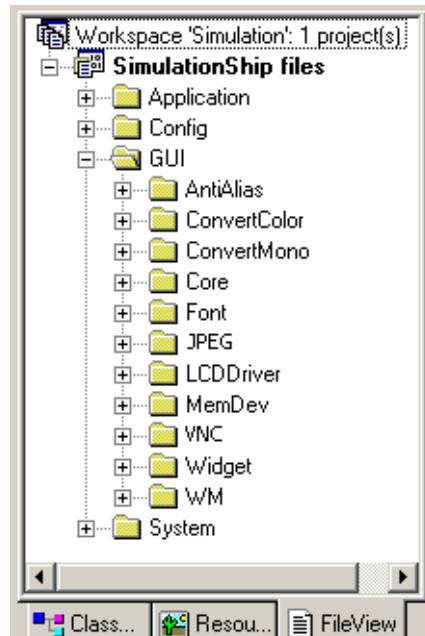
If you want to start a new project you should make a copy of the Start-folder. It contains all you need for a new project. The sub-directories containing emWin program files are in the Start\GUI folder and should contain the exact same files as the directories of the same names which you are using for your target (cross) compiler. You should not make any changes to the GUI subdirectories, 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).



3.1.2.2 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. Please 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 LCDDriver are part of the basic emWin package and will always appear in the workspace directory.



3.1.2.3 Compiling the application

The demo simulation contains one or more application "C" files (located in the Application directory), which can be modified. You may also add files to or remove files from the project. Typically you would want to at least change the bitmap to your own company logo or image of choice. 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 simulator 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.

3.2 Device simulation

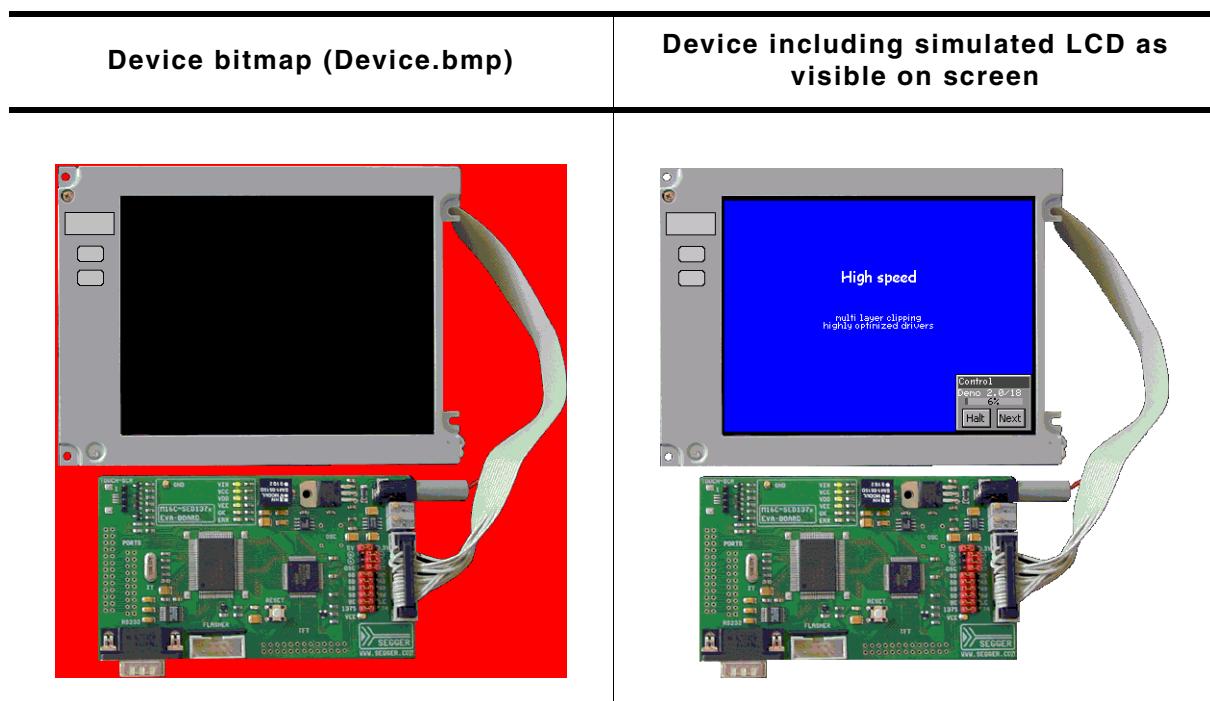
The simulator can show the simulated LCD in a bitmap of your choice, typically your target device. The bitmap can be dragged over the screen and may, in certain applications, be used to simulate the behavior of the entire target device.

In order to simulate the appearance of the device, a bitmap is required. This bitmap is usually a photo (top view) of the device, and must 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 by including the following line in the resource file (extension `.rc`):

```
145 BITMAP DISCARDABLE "Device.bmp"
```

For more information, please refer to the Win32 documentation.

The size of the bitmap should be such that the size of the area in which the LCD will be shown equals the resolution of the simulated LCD. This is best seen in the following example:



The red area is automatically made transparent. The transparent 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 (0xFF0000) 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_SetTransColor()`.

3.2.1 Device simulator API

All of the device simulator API functions must be called in the setup phase. The calls should ideally be done from within the routine `SIM_X_Init()`, which is located in the file `SIM_X.c`. The example below calls `SIM_SetLCDPos()` in the setup:

```
#include <windows.h>
#include <stdio.h>

#include "SIM.h"

void SIM_X_Init() {
    SIM_SetLCDPos(0,0);      // Define the position of the LCD in the bitmap
}
```

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

Routine	Explanation
<code>SIM_GUI_SetLCDColorBlack()</code>	Set the color to be used as black (color monochrome displays).
<code>SIM_GUI_SetLCDColorWhite()</code>	Set the color to be used as white (color monochrome displays).
<code>SIM_GUI_SetLCDPos()</code>	Set the position for the simulated LCD within the target device bitmap.
<code>SIM_GUI_SetMag()</code>	Set magnification factors for X and/or Y axis.
<code>SIM_GUI_SetTransColor()</code>	Set the color to be used for transparent areas (default: 0xFF0000).

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	Meaning
<code>DisplayIndex</code>	Reserved for future use; must be 0.
<code>Color</code>	RGB value of the color.

Add. 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 0xFFFFFFF.

Example using default settings

```
void SIM_X_Init() {
    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, 0xFFFFFFFF); // Define the color used as white
    // (used for colored monochrome displays)
}
```



Example using yellow instead of white

```
void SIM_X_Init() {
    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	Meaning
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). Composite view (Layer 0 +

Layer 0

Add. information

The X- and Y-positions are relative to the target device bitmap, therefore Layer 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. If the use of the device bitmaps should be disabled, omit the call of this function in `SIM_X_Init()`.

SIM_GUI_SetMag()

Layer 1

Description

Sets magnification factors for X and/or Y axis.

Prototype

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

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

Add. 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	Meaning
Color	RGB value of the color in the format 00000000RRRRRRRGGGGGGGBBBBBBB.

Add. 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.

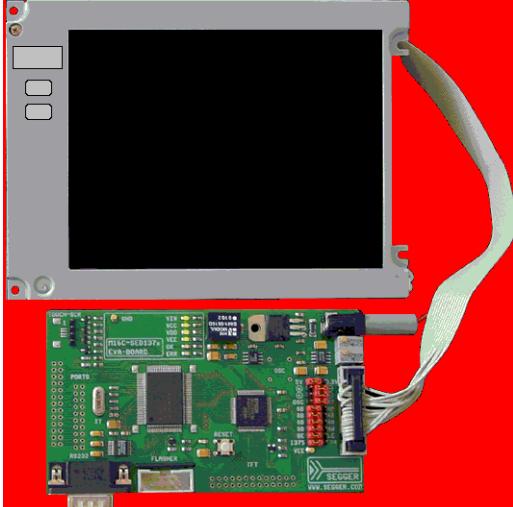
3.2.2 Hardkey simulation

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 "unpresses" 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). This bitmap can again be in a separate file in the directory, or included as a resource in the executable. The filename needs to be `Device1.bmp`, and the following lines would typically be included in the resource file (extension `.rc`):

```
145 BITMAP DISCARDABLE "Device.bmp"
146 BITMAP DISCARDABLE "Device1.bmp"
```

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)
	

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.

3.2.2.1 Hardkey simulator 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	Explanation
<code>SIM_HARDKEY_GetNum()</code>	Return the number of available hardkeys.
<code>SIM_HARDKEY_GetState()</code>	Return the state of a specified hardkey (0: unpressed, 1: pressed).
<code>SIM_HARDKEY_SetCallback()</code>	Set a callback routine to be executed when the state of a specified hardkey changes.
<code>SIM_HARDKEY_SetMode()</code>	Set the behavior for a specified hardkey (default = 0: no toggle).
<code>SIM_HARDKEY_SetState()</code>	Set the state for a specified hardkey (0: unpressed, 1: pressed).

SIM_HARDKEY_GetNum()

Description

Returns the number of available hardkeys.

Prototype

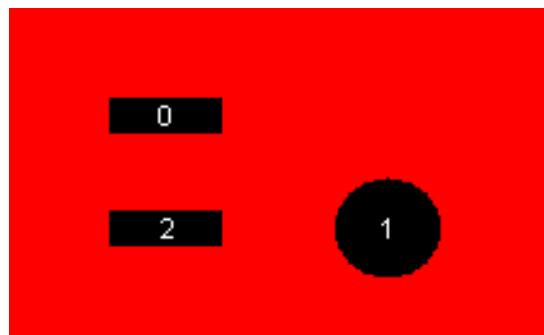
```
int SIM_HARDKEY_GetNum(void);
```

Return value

The number of available hardkeys found in the bitmap.

Add. 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	Meaning
<code>KeyIndex</code>	Index of hardkey (0 = index of first key).

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

Return value

Pointer to the previous callback routine.

Add. information

The callback routine must have the following prototype:

Prototype

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

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

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	Meaning
KeyIndex	Index of hardkey (0 = index of first key).
Mode	Behavior mode (see table below).

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

Add. 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	Meaning
KeyIndex	Index of hardkey (0 = index of first key).
State	State of the specified hardkey (see table below).

Permitted values for parameter State	
0	Unpressed.
1	Pressed.

Add. information

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

3.3 Integrating the emWin simulation into an existing simulation

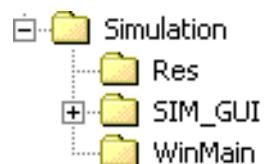
In order to integrate the emWin simulation into an existing simulation, the source code of the simulation is not required. The source code of the simulation is not normally shipped with emWin. It is a separate (optional) software item and is not included in the emWin basic package.

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.

3.3.1 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	Contents
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.

3.3.2 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`.

3.3.2.1 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 this order as shown in the following application code sample):

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

3.3.2.2 Sample 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);
    /* 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 a 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();
}

```

3.3.3 Integration into the embOS Simulation

3.3.3.1 WinMain

The following code sample 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);
    /* 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;
}
```

3.3.3.2 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 "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) {
    GUI_COLOR aColor[] = {GUI_RED, GUI_YELLOW};
    GUI_Init();
    while (1) {
        int i;
        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
*
*****/
#include <windows.h>
void main(void) {
    OS_IncDI(); /* Initially disable interrupts */
    OS_InitKern(); /* initialize OS */
    OS_InitHW(); /* initialize Hardware for OS */
    /* You need to create at least one task 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

3.3.4 GUI simulation API

The table below lists the available routines for user defined simulation programmes 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	Explanation
<code>SIM_GUI_CreateLCDInfoWindow()</code>	Creates a window which shows the available colors of the given layer with the given size and position.
<code>SIM_GUI_CreateLCDWindow()</code>	Creates a LCD window with the given size and position.
<code>SIM_GUI_Exit()</code>	Stops the GUI simulation.
<code>SIM_GUI_Init()</code>	Initializes the GUI simulation.
<code>SIM_GUI_SetLCDWindowHook()</code>	Sets a hook function to be called if the LCD window receives a message.

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	Meaning
<code>hParent</code>	Handle of the parent window.
<code>x</code>	X position in parent coordinates.
<code>y</code>	Y position in parent coordinates.
<code>xSize</code>	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.
<code>ySize</code>	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.
<code>LayerIndex</code>	Index of layer to be shown.

Add. 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	Meaning
<code>hParent</code>	Handle of the parent window.
<code>x</code>	X position in parent coordinates.
<code>y</code>	Y position in parent coordinates.
<code>xSize</code>	X size in pixel of the new window.
<code>ySize</code>	Y size in pixel of the new window.
<code>LayerIndex</code>	Index of layer to be shown.

Add. 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.h`.

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

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	Meaning
<code>hInst</code>	Handle to current instance passed to WinMain.
<code>hWndMain</code>	Handle of the simulations main window.
<code>pCmdLine</code>	Pointer to command line passed to WinMain
<code>sAppName</code>	Pointer to a string that contains the application name.

Add. information

The parameters `hWndMain` and `sAppName` are used if 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	Meaning
<code>pfHook</code>	Pointer to hook function.

Prototype of hook function

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

Parameter	Meaning
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.

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 simulator 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:

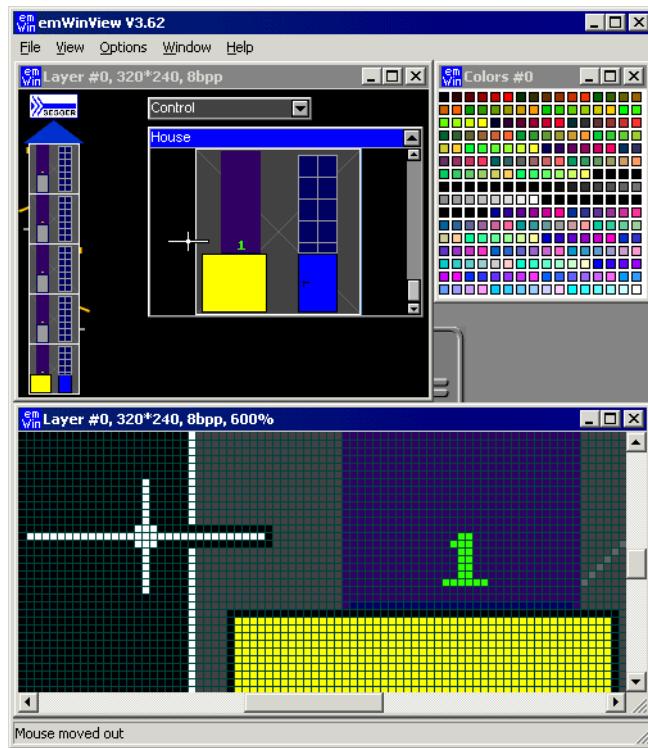
- Multiple windows for each layer
- Watching the whole virtual layer in one window
- Magnification of each layer window
- Composite view if using multiple layers

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 multi layer configuration, a composite view window will also be visible.



4.1.1 Using the simulator and the viewer

If you use the simulator 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.

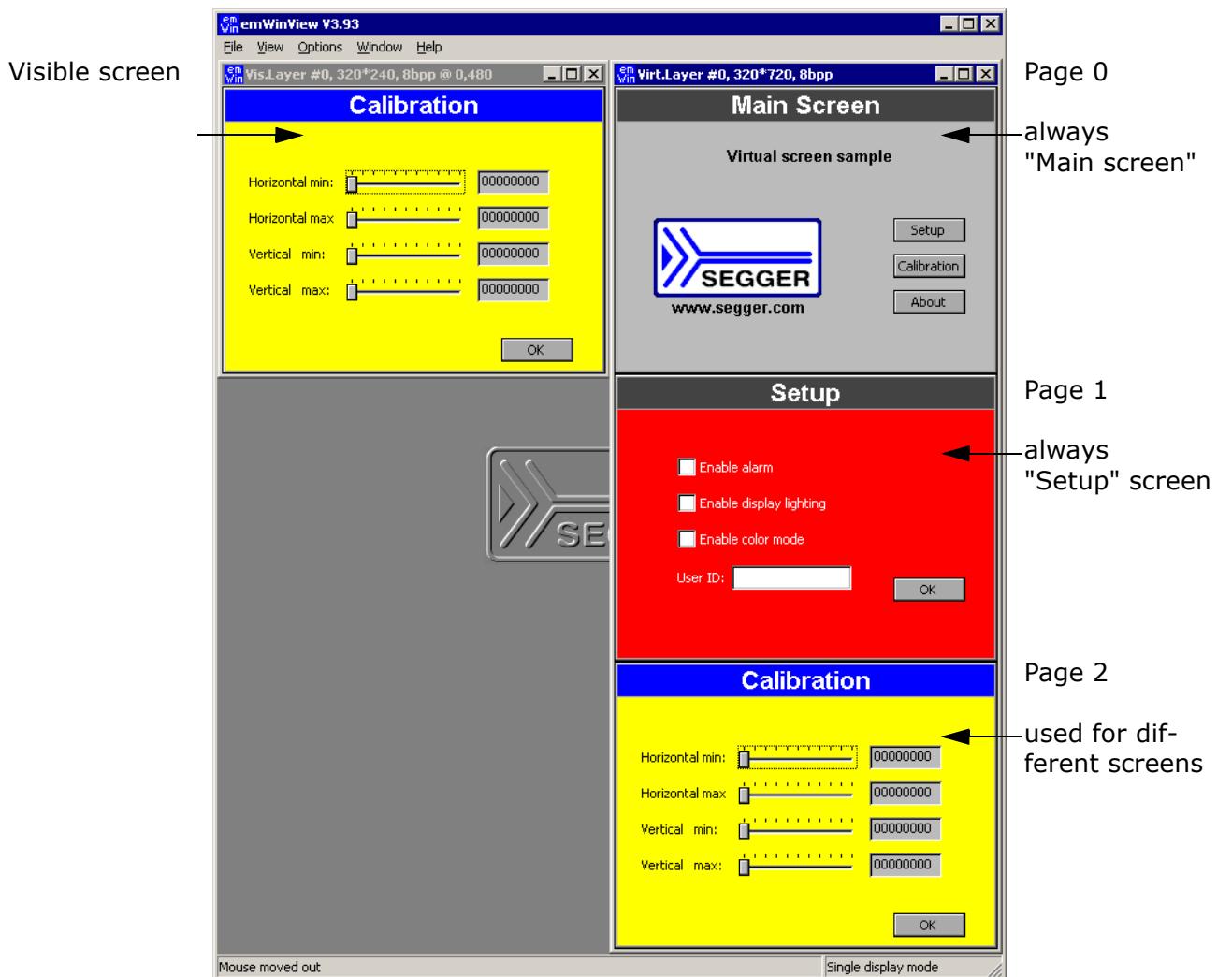
The emWin viewer solves this problem by showing the display window and the color window of your simulation in a separate process. It is your choice if you want to start the viewer before debugging your application or while you are debugging. Our suggestion:

- 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 please refer to chapter 'Virtual Screens'.

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

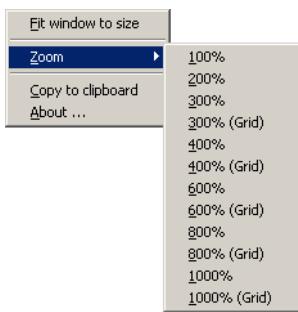
If you want to show a magnified area of the LCD output or the composite view of a multi layer configuration it could be useful to open more than one output window. You can do this by `View/Visible Layer/Layer (1...4)`, `View/Virtual Layer/Layer (1...4)` or `View/Composite`.

4.1.5 Zooming

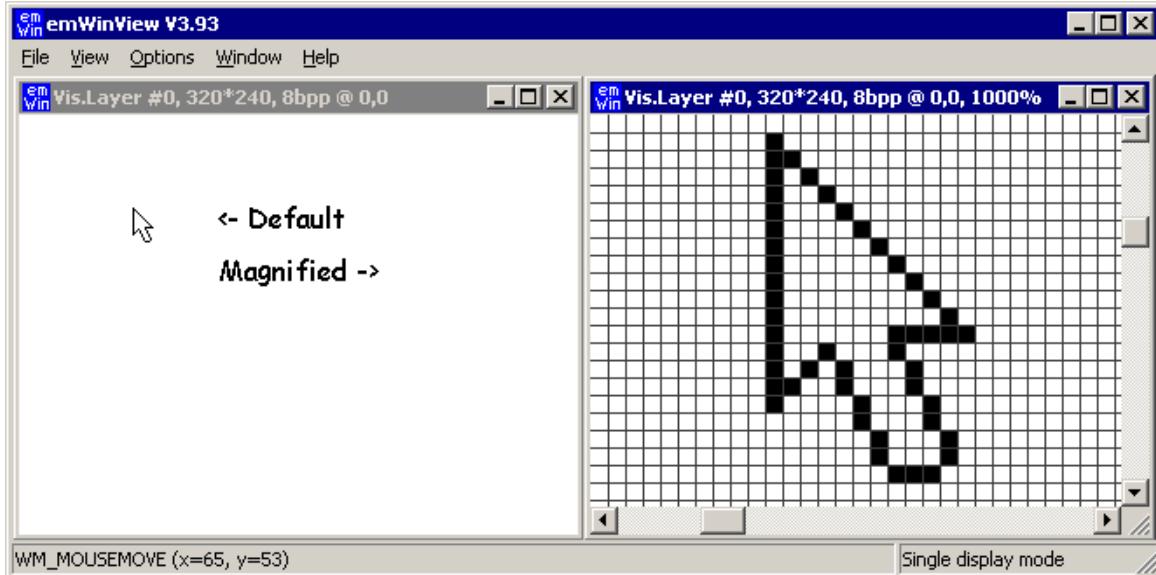
Zooming in or out is easy:

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 $\geq 300\%$, 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

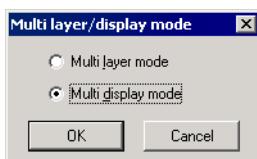
If you want to adapt the size of the window to the magnification choose Fit window to size from the first popup menu.

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 mspaint application.

4.1.7 Using the viewer with multiple displays

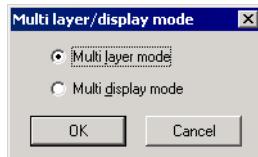
If you are working with multiple displays you should set the viewer into 'Multi display mode' by using the command Options/Multi layer/display.



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

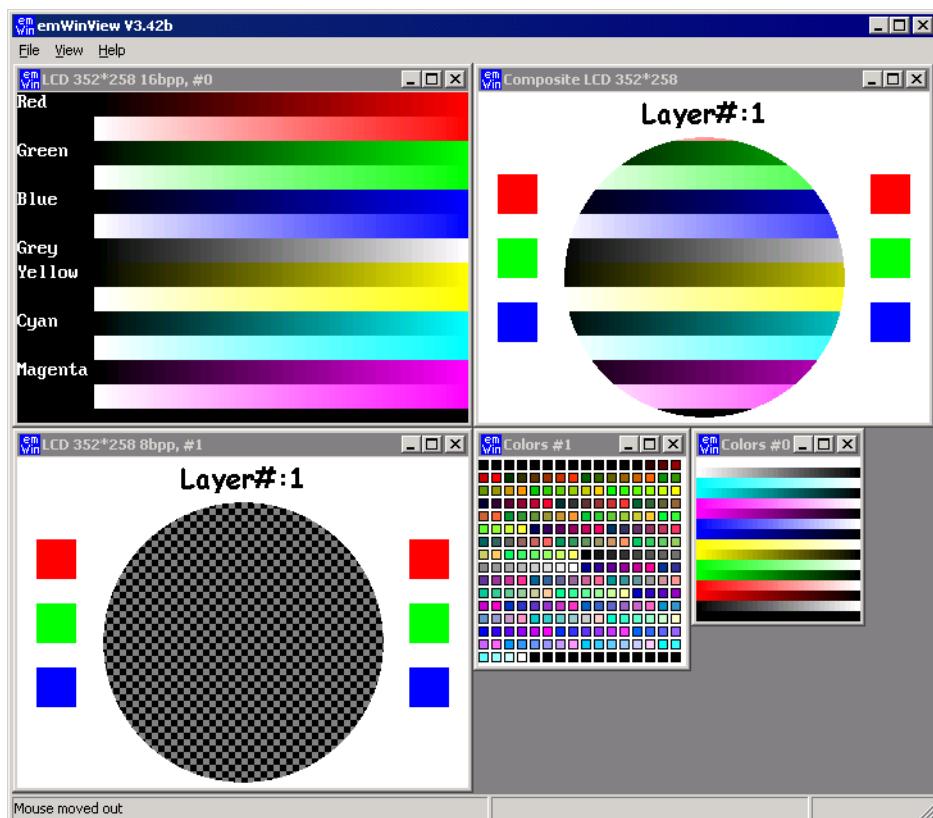
If you are working with multiple layers you should set the viewer into 'Multi layer mode' by using the command Options/Multi layer/display.



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.

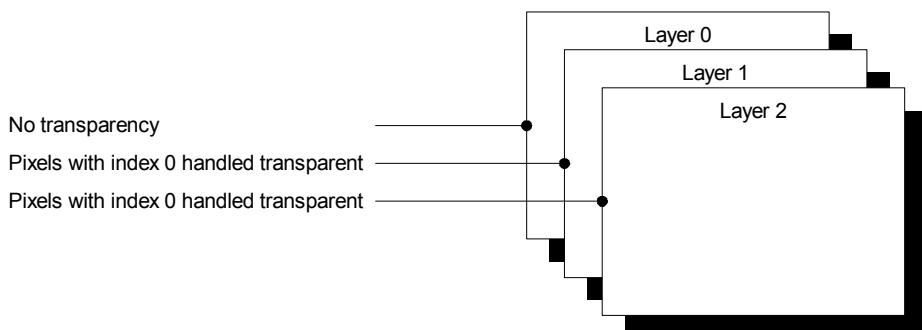
Sample

The sample 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 viewer treats pixels with index 0 in a layer above layer 0 as transparent. Therefore in the composite view the pixels of the layers below are visible. The composite window shows the layers one above the other:



Transparent pixels will be shown in layer windows tiled

Chapter 5

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 explanations of the individual routines that are available.

5.1 Basic routines

In order to display text on the LCD, simply call the routine `GUI_DispString()` with the text you want to display as parameters. For example:

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

The above code will display the text "Hello world" at the current text position. However, as you will see, there are routines to display text in a different font or in a certain position. In addition, it is possible to write not only strings but also decimal, hexadecimal and binary values to the display. Even though the graphic displays are usually byte-oriented, the text can be positioned at any pixel of the display, not only at byte positions.

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 for the following:

Char. Code	ASCII code	"C"	Meaning
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 <code>GUI_GetFontDistY()</code>).
13	CR	\r	Carriage return. The current text position is changed to the beginning of the current line. Per default, this is: X = 0.

Usage of the control character `LF` can be very convenient in strings. A line feed can be made part of a string so that a string spanning multiple lines can be displayed with a single routine call.

Positioning text at a selected position

This may be done by using the routine `GUI_GotoXY()` as shown in the following example:

```
GUI_GotoXY(10,10); // Set text position (in pixels)
GUI_DispString("Hello world!"); // Show text
```

5.2 Text API

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

Routine	Explanation
Routines to display text	
<code>GUI_DispChar()</code>	Displays single character at current position.
<code>GUI_DispCharAt()</code>	Displays single character at specified position.
<code>GUI_DispChars()</code>	Displays character a specified number of times.
<code>GUI_DispNextLine()</code>	Moves the cursor to the beginning of the next line.
<code>GUI_DispString()</code>	Displays string at current position.
<code>GUI_DispStringAt()</code>	Displays string at specified position.
<code>GUI_DispStringAtCEOL()</code>	Displays string at specified position, then clear to end of line.
<code>GUI_DispStringHCenterAt()</code>	Displays string centered horizontally at the given position.
<code>GUI_DispStringInRect()</code>	Displays string in specified rectangle.
<code>GUI_DispStringInRectEx()</code>	Displays string in specified rectangle and optionally rotates it.

Routine	Explanation
<code>GUI_DisplStringLen()</code>	Display string at current position with specified number of characters.
	Selecting text drawing modes
<code>GUI_GetTextMode()</code>	Returns the current text mode
<code>GUI_SetTextMode()</code>	Sets text drawing mode.
<code>GUI_SetTextStyle()</code>	Sets the text style to be used.
	Selecting text alignment
<code>GUI_GetTextAlign()</code>	Return current text alignment mode.
<code>GUI_SetLBorder()</code>	Set left border after line feed.
<code>GUI_Set.TextAlign()</code>	Set text alignment mode.
	Setting the current text position
<code>GUI_GotoX()</code>	Set current X-position.
<code>GUI_GotoXY()</code>	Set current (X,Y) position.
<code>GUI_GotoY()</code>	Set current Y-position.
	Retrieving the current text position
<code>GUI_GetDispPosX()</code>	Return current X-position.
<code>GUI_GetDispPosY()</code>	Return current Y-position.
	Routines to clear a window or parts of it
<code>GUI_Clear()</code>	Clear active window (or entire display if background is the active window).
<code>GUI_DisplCEOL()</code>	Clear display from current text position to end of line.

5.3 Routines to display text

`GUI_DisplChar()`

Description

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

Prototype

```
void GUI_DisplChar(U16 c);
```

Parameter	Meaning
<code>c</code>	Character to display.

Add. information

This is the basic routine for displaying a single character. All other display routines (`GUI_DisplCharAt()`, `GUI_DisplString()`, 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_DisplChar('A');
```

Related topics

```
GUI_DisplChars(), GUI_DisplCharAt()
```

GUI_DispatchAt()

Description

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

Prototype

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

Parameter	Meaning
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.

Add information

Displays the character with its upper left corner at the specified (X,Y) position.

Writes the character using the routine `GUI_Dispatch()`.

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

Related topics

```
GUI_Dispatch(), GUI_DispatchChars()
```

GUI_DispatchChars()

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_DispatchChars(U16 c, int Cnt);
```

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

Add. information

Writes the character using the routine `GUI_Dispatch()`.

If the character is not available in the current font, nothing is displayed.

Example

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

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

Related topics

```
GUI_Dispatch(), GUI_DispatchAt()
```

GUI_DispatchNextLine()

Description

Moves the cursor to the beginning of the next line.

Prototype

```
void GUI_DispatchNextLine(void);
```

Related topics

`GUI_SetLBorder()`

GUI_DisppString()

Description

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

Prototype

```
void GUI_DisppString(const char GUI_FAR *s);
```

Parameter	Meaning
<code>s</code>	String to display.

Add. 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_DisppString("Hello world");// Disp text
GUI_DisppString("\nNext line");// Disp text
```

Related topics

`GUI_DisppStringAt()`, `GUI_DisppStringAtCEOL()`, `GUI_DisppStringLen()`,

GUI_DisppStringAt()

Description

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

Prototype

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

Parameter	Meaning
<code>s</code>	String to display.
<code>x</code>	X-position to write to in pixels of the client window.
<code>y</code>	Y-position to write to in pixels of the client window.

Example

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

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

Related topics

`GUI_DisppString()`, `GUI_DisppStringAtCEOL()`, `GUI_DisppStringLen()`,

GUI_DisppStringAtCEOL()

Description

This routine uses the exact same parameters as `GUI_DisppStringAt()`. 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_DisppCEOL()`. 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_DispatchStringHCenterAt()

Description

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

Prototype

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

Parameter	Meaning
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.

GUI_DispatchStringInRect()

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_DispatchStringInRect(const char GUI_FAR *s, const GUI_RECT *pRect,
int Align);
```

Parameter	Meaning
s	String to display.
pRect	Rectangle to write to in pixels of the client window.
Align	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.

Example

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

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

Add. information

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

Related topics

[GUI_DispatchString\(\)](#), [GUI_DispatchStringAtCEOL\(\)](#), [GUI_DispatchStringLen\(\)](#),

GUI_DispatchStringInRectEx()

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_DispatchStringInRectEx(const char * s,
                                GUI_RECT * pRect,
                                int TextAlign,
                                int MaxLen,
```

```
const GUI_ROTATION * pLCD_Api);
```

Parameter	Meaning
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)

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_DisppStringInRectEx(acText,
                        &Rect,
                        GUI_TA_HCENTER | GUI_TA_VCENTER,
                        strlen(acText),
                        GUI_ROTATE_CCW);
```

Screenshot of above example



Add. 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_DisppStringLen()

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_DisppStringLen(const char GUI_FAR *s, int Len);
```

Parameter	Meaning
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.

Add. 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_DisppString()`, `GUI_DisppStringAt()`, `GUI_DisppStringAtCEOL()`,

5.4 Selecting text drawing modes

Normally, text is written into the selected window at the current text position using the selected font in normal text. Normal text means that the text overwrites whatever is already displayed where the bits set in the character mask are set on the display. In this mode, active bits are written using the foreground color, while inactive bits are written with the background color. However, in some situations it may be desirable to change this default behavior. emWin offers four flags for this purpose (one default plus three modifiers), which may be combined:

Normal text

Text can be displayed normally by specifying `GUI_TEXTMODE_NORMAL` or 0.

Reverse text

Text can be displayed in reverse by specifying `GUI_TEXTMODE_REVERSE`. What is usually displayed as white on black will be displayed as black on white.

Transparent text

Transparent text means that the text is written on top of whatever is already visible on the display. The difference is that whatever was previously on the screen can still be seen, whereas with normal text the background is erased.

Text can be displayed transparently by specifying `GUI_TEXTMODE_TRANS`.

XOR text

What usually is drawn white (the actual character) is inverted. The effect is identical to that of the default mode (normal text) if the background is black. If the background is white, the output is identical to reverse text.

If you use colors, an inverted pixel is calculated as follows:

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

Transparent reversed text

As with transparent text, it does not overwrite the background, and as with reverse text, the text is displayed in reverse.

Text can be displayed in reverse transparently by specifying `GUI_TEXTMODE_TRANS | GUI_TEXTMODE_REV`.

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_DisppStringHCenterAt("GUI_TM_NORMAL" , 160, 10);
GUI_SetTextMode(GUI_TM_REV);
GUI_DisppStringHCenterAt("GUI_TM_REV" , 160, 26);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DisppStringHCenterAt("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);

```

Screen shot of above example



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

Permitted values for parameter <code>TextMode</code> (OR-combinable)	
GUI_TEXTMODE_NORMAL	Sets normal text. This is the default setting; the value is identical to 0.
GUI_TEXTMODE_REVERSE	Sets reverse text.
GUI_TEXTMODE_TRANSPARENT	Sets transparent text.
GUI_TEXTMODE_XOR	Text will be inverted on the display.

Return value

The previous selected text mode.

Example

Shows "The value is" at position 0,0 on the display, shows a value in reverse text, then sets the text mode back to normal:

```

int i = 20;
GUI_DispStringAt("The value is", 0, 0);
GUI_SetTextMode(GUI_TEXTMODE_REVERSE);
GUI_DispDec(20, 3);
GUI_SetTextMode(GUI_TEXTMODE_NORMAL);

```

GUI_SetTextStyle()

Description

Sets the text style to the parameter specified.

Prototype

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

Parameter	Meaning
Style	Text style to set (see table below).

Permitted values for parameter <i>Style</i>	
GUI_TS_NORMAL	Renders text normal (default).
GUI_TS_UNDERLINE	Renders text underlined.
GUI_TS_STRIKETHRU	Renders text in strikethrough type.
GUI_TS_OVERLINE	Renders text in overline type.

Return value

The previous selected text style.

5.5 Selecting text alignment

GUI_Get TextAlign()

Description

Returns the current text alignment mode.

Prototype

```
int GUI_Get TextAlign(void);
```

GUI_SetLBorder()

Description

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

Prototype

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

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

GUI_Set TextAlign()

Description

Sets the text alignment mode for string output in the current window.

Prototype

```
int GUI_Set TextAlign(int TextAlign);
```

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

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

Return value

The selected text alignment mode.

Add. information

`GUI_SetTextAlign()` does not affect the character output routines beginning with `GUI_DispChar()`.

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);
```

5.6 Setting the current text position

Every task has a current text position. This is the position relative to the origin of the window (usually (0,0)) where the next character will be written if a text output routine is called. Initially, this position is (0,0), which is the upper left corner of the current window. There are 3 functions which can be used to set the current text position.

`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	Meaning
<code>x</code>	New X-position (in pixels, 0 is left border).
<code>y</code>	New Y-position (in pixels, 0 is top border).

Return value

Usually 0.

If a value != 0 is returned, then the current text position is outside of the window (to the right or below), so a following write operation can be omitted.

Add. information

`GUI_GotoXY()` sets both the X- and Y-components of the current text position.
`GUI_GotoX()` sets the X-component of the current text position; the Y-component remains unchanged.
`GUI_GotoY()` sets the Y-component of the current text position; the X-component remains unchanged.

Example

```
Shows "(20,20)" at position 20,20 on the display:  
GUI_GotoXY(20,20)  
GUI_DisppString("The value is");
```

5.7 Retrieving the current text position

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);
```

5.8 Routines to clear a window or parts of it

GUI_Clear()

Description

Clears the current window.

Prototype

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

Add. 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_DisppString("Hello world", 0, 0); // Disp text  
GUI_Delay(1000); // Wait 1 second (not part of emWin)  
GUI_Clear(); // Clear screen
```

GUI_DisppCEOL()

Description

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

Prototype

```
void GUI_DisppCEOL(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 );// Disp text
Delay (1000);
GUI_DispStringAt( "Hi ", 0, 0 );
GUI_DispCEOL();
```


Chapter 6

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.

6.1 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	Explanation
Displaying decimal values	
GUI_DisplDec()	Display value in decimal form at current position with specified number of characters.
GUI_DisplDecAt()	Display value in decimal form at specified position with specified number of characters.
GUI_DisplDecMin()	Display value in decimal form at current position with minimum number of characters.
GUI_DisplDecShift()	Display long value in decimal form with decimal point at current position with specified number of characters.
GUI_DisplDecSpace()	Display value in decimal form at current position with specified number of characters, replace leading zeros with spaces.
GUI_DisplSDec()	Display value in decimal form at current position with specified number of characters and sign.
GUI_DisplSDecShift()	Display long value in decimal form with decimal point at current position with specified number of characters and sign.
Displaying floating-point values	
GUI_DisplFloat()	Display floating-point value with specified number of characters.
GUI_DisplFloatFix()	Display floating-point value with fixed no. of digits to the right of decimal point.
GUI_DisplFloatMin()	Display floating-point value with minimum number of characters.
GUI_DisplSFloatFix()	Display floating-point value with fixed no. of digits to the right of decimal point and sign.
GUI_DisplSFloatMin()	Display floating-point value with minimum number of characters and sign.
Displaying binary values	
GUI_DisplBin()	Display value in binary form at current position.
GUI_DisplBinAt()	Display value in binary form at specified position.
Displaying hexadecimal values	
GUI_DisplHex()	Display value in hexadecimal form at current position.
GUI_DisplHexAt()	Display value in hexadecimal form at specified position.
Version of emWin	
GUI_GetVersionString()	Return the current version of emWin.

6.2 Displaying decimal values

GUI_DisplDec()

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_DisplDec(I32 v, U8 Len);
```

Parameter	Meaning
v	Value to display. Minimum -2147483648 (= -2^31). Maximum 2147483647 (= 2^31 -1).
Len	No. of digits to display (max. 9).

Add. 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_DispatchString("Min:");
GUI_DispatchDec(Min,2);
GUI_DispatchString(" Sec:");
GUI_DispatchDec(Sec,2);
```

Related topics

`GUI_DispatchSDec()`, `GUI_DispatchDecAt()`, `GUI_DispatchDecMin()`, `GUI_DispatchDecSpace()`

GUI_DispatchDecAt()

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

Parameter	Meaning
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	No. of digits to display (max. 9).

Add. information

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

Example

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

Related topics

`GUI_DispatchDec()`, `GUI_DispatchSDec()`, `GUI_DispatchDecMin()`, `GUI_DispatchDecSpace()`

GUI_DispatchDecMin()

Description

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

Prototype

```
void GUI_DispatchDecMin(I32 v);
```

Parameter	Meaning
v	Value to display. Minimum: -2147483648 (= -2 ³¹); maximum 2147483647 (= 2 ³¹ -1). Maximum no. of digits displayed is 9.

Add. information

If values have to be aligned but differ in the number of digits, this function is not a good choice. Try one of the functions that specify the number of digits.

Example

```
// Show result
GUI_DisppString("The result is :");
GUI_DisppDecMin(Result);
```

Related topics

[GUI_DisppDec\(\)](#), [GUI_DisppDecAt\(\)](#), [GUI_DisppSDec\(\)](#), [GUI_DisppDecSpace\(\)](#)

GUI_DisppDecShift()

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_DisppDecShift(I32 v, U8 Len, U8 Shift);
```

Parameter	Meaning
<code>v</code>	Value to display. Minimum: -2147483648 (= -2 ³¹); maximum: 2147483647 (= 2 ³¹ -1).
<code>Len</code>	No. of digits to display (max. 9).
<code>Shift</code>	No. of digits to show to right of decimal point.

Add. information

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

GUI_DisppDecSpace()

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	Meaning
<code>v</code>	Value to display. Minimum: -2147483648 (= -2 ³¹); maximum: 2147483647 (= 2 ³¹ -1).
<code>MaxDigits</code>	No. of digits to display, including leading spaces. Maximum no. of digits displayed is 9 (excluding leading spaces).

Add. information

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

Example

```
// Show result
GUI_DisppString("The result is :");
GUI_DisppDecSpace(Result, 200);
```

Related topics

[GUI_DisppDec\(\)](#), [GUI_DisppDecAt\(\)](#), [GUI_DisppSDec\(\)](#), [GUI_DisppDecMin\(\)](#)

GUI_DisppSDec()

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_DispatchSDec(I32 v, U8 Len);
```

Parameter	Meaning
v	Value to display. Minimum: -2147483648 (= -2^31); maximum: 2147483647 (= 2^31 -1).
Len	No. of digits to display (max. 9).

Add. information

Leading zeros are not suppressed.

This function is similar to `GUI_DispatchDec`, but a sign is always shown in front of the value, even if the value is positive.

Related topics

`GUI_DispatchDec()`, `GUI_DispatchDecAt()`, `GUI_DispatchDecMin()`, `GUI_DispatchDecSpace()`

GUI_DispatchSDecShift()

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_DispatchSDecShift(I32 v, U8 Len, U8 Shift);
```

Parameter	Meaning
v	Value to display. Minimum: -2147483648 (= -2^31); maximum: 2147483647 (= 2^31 -1).
Len	No. of digits to display (max. 9).
Shift	No. of digits to show to right of decimal point.

Add. information

A sign is always shown in front of the value.

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

Example

```
void DemoDec(void) {
    long l = 12345;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x8);
    GUI_DispatchStringAt("GUI_DispatchSDecShift:\n", 0, 0);
    GUI_DispatchSDecShift(l, 7, 3);
    GUI_SetFont(&GUI_Font6x8);
    GUI_DispatchStringAt("Press any key", 0, GUI_VYSIZE-8);
    WaitKey();
}
```

Screen shot of above example



6.3 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	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	No. of digits to display (max. 9).

Add. information

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

Example

```
/*      Shows all features for displaying floating point values      */
void DemoFloat(void) {
    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);
    GUI_SetFont(&GUI_Font6x8);
    GUI_DispStringAt("Press any key", 0, GUI_VYSIZE-8);
    WaitKey();
}
```

Screen shot of above example

```

GUI_DispFloat:
123.45678 -123.4568
GUI_DispFloatFix:
000123.46 -00123.46
GUI_DispSFloatFix:
+00123.46 -00123.46
GUI_DispFloatMin:
123.457 -123.457
GUI_DispSFloatMin:
+123.457 -123.457

Press any key

```

GUI_DispFloat()

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_DispFloat (float v, char Len, char Decs);
```

Parameter	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	No. of digits to display (max. 9).
Decs	No. of digits to show to right of decimal point.

Add. 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 f, char Fract);
```

Parameter	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Fract	Minimum no. of characters to display.

Add. information

Leading zeros are suppressed.
If the value is negative, a minus sign is shown.
The length need not be specified; the minimum length will automatically be used. If values have to be aligned but differ in the number of digits, this function is not a good choice. Try one of the functions that specify the number of digits.

GUI_DisplSFloatFix()

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_DisplSFloatFix(float v, char Len, char Decs);
```

Parameter	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	No. of digits to display (max. 9).
Decs	No. of digits to show to right of decimal point.

Add. information

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

GUI_DisplSFloatMin()

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_DisplSFloatMin(float f, char Fract);
```

Parameter	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Fract	Minimum no. of digits to display.

Add. information

Leading zeros are suppressed.
A sign is always shown in front of the value.
The length need not be specified; the minimum length will automatically be used. If values have to be aligned but differ in the number of digits, this function is not a good choice. Try one of the functions that specify the number of digits.

6.4 Displaying binary values

GUI_DisplBin()

Description

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

Prototype

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

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

Add. 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_DispatchAt\(\)](#)

GUI_DispatchAt()

Description

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

Prototype

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

Parameter	Meaning
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).

Add. information

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

Example

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

Related topics

[GUI_Dispatch\(\)](#), [GUI_DispatchHex\(\)](#)

6.5 Displaying hexadecimal values

GUI_DispatchHex()

Description

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

Prototype

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

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

Add. 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	Meaning
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.

Add. 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()`

6.6 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_DispatchString(GUI_GetVersionString());
```

Chapter 7

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 (please refer to Chapter 14: "The Window Manager") and are based on fast and efficient algorithms. Currently, only the `DrawArc()` function requires floating-point calculations.

7.1 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	Explanation
Drawing modes	
<code>GUI_GetDrawMode()</code>	Returns the current drawing mode
<code>GUI_SetDrawMode()</code>	Sets the drawing mode.
Query current client rectangle	
<code>GUI_GetClientRect()</code>	Returns the current available drawing area
Basic drawing routines	
<code>GUI_ClearRect()</code>	Fills a rectangular area with the background color.
<code>GUI_DrawPixel()</code>	Draws a single pixel.
<code>GUI_DrawPoint()</code>	Draws a point.
<code>GUI_DrawRect()</code>	Draws a rectangle.
<code>GUI_DrawRectEx()</code>	Draws a rectangle.
<code>GUI_FillRect()</code>	Draws a filled rectangle.
<code>GUI_FillRectEx()</code>	Draws a filled rectangle.
<code>GUI_InvertRect()</code>	Invert a rectangular area.
Drawing bitmaps	
<code>GUI_DrawBitmap()</code>	Draws a bitmap.
<code>GUI_DrawBitmapEx()</code>	Draws a scaled bitmap.
<code>GUI_DrawBitmapExp()</code>	Draws a bitmap using additional parameters.
<code>GUI_DrawBitmapMag()</code>	Draws a magnified bitmap.
Drawing lines	
<code>GUI_DrawHLine()</code>	Draws a horizontal line.
<code>GUI_DrawLine()</code>	Draws a line from a specified startpoint to a specified endpoint (absolute coordinates).
<code>GUI_DrawLineRel()</code>	Draws a line from the current position to an endpoint specified by X- and Y-distances (relative coordinates).
<code>GUI_DrawLineTo()</code>	Draws a line from the current position to a specified endpoint.
<code>GUI_DrawPolyLine()</code>	Draws a polyline.
<code>GUI_DrawVLine()</code>	Draws a vertical line.
<code>GUI_GetLineStyle()</code>	Returns the current line style.
<code>GUI_MoveRel()</code>	Moves the line pointer relative to its current position
<code>GUI_MoveTo()</code>	Moves the line pointer to the given position
<code>GUI_SetLineStyle()</code>	Sets the current line style.
Drawing polygons	
<code>GUI_DrawPolygon()</code>	Draws the outline of a polygon.
<code>GUI_EnlargePolygon()</code>	Enlarges a polygon.
<code>GUI_FillPolygon()</code>	Draws a filled polygon.
<code>GUI_MagnifyPolygon()</code>	Magnifies a polygon.
<code>GUI_RotatePolygon()</code>	Rotates a polygon by a specified angle.
Drawing circles	
<code>GUI_DrawCircle()</code>	Draws the outline of a circle.
<code>GUI_FillCircle()</code>	Draws a filled circle.
Drawing ellipses	
<code>GUI_DrawEllipse()</code>	Draws the outline of an ellipse.
<code>GUI_FillEllipse()</code>	Draws a filled ellipse.
Drawing arcs	
<code>GUI_DrawArc()</code>	Draws an arc.
Drawing a graph	

Routine	Explanation
<code>GUI_DrawGraph()</code>	Draws a graph.
	Saving and restoring the GUI-context
<code>GUI_RestoreContext()</code>	Restores the GUI-context.
<code>GUI_SaveContext()</code>	Saves the GUI-context.
	Clipping
<code>GUI_SetClipRect()</code>	Sets the rectangle used for clipping

7.2 Drawing modes

emWin can draw in NORMAL mode or in XOR mode. The default is NORMAL mode, in which the content of the display is overdrawn by the graphic. In XOR mode, the content of the display is inverted when it is overdrawn.

Restrictions associated with `GUI_DRAWMODE_XOR`

- XOR mode is only useful when using two displayed colors inside the active window or screen.
- Some drawing functions of emWin do not work precisely with this drawing mode. Generally, this mode works only with a pen size of one pixel. That means before using functions like `GUI_DrawLine()`, `GUI_DrawCircle()`, `GUI_DrawRect()` and so on, you must make sure that the pen size is set to 1 when you are working in XOR mode.
- 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.

`GUI_GetDrawMode()`

Description

Returns the current drawing mode.

Prototype

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

Return value

The currently selected drawing mode.

Add. information

For details about drawing modes please refer to the function `GUI_SetDrawMode()`.

`GUI_SetDrawMode()`

Description

Selects the specified drawing mode.

Prototype

```
GUI_DRAWMODE GUI_SetDrawMode(GUI_DRAWMODE mode);
```

Parameter	Meaning
<code>mode</code>	Drawing mode to set. May be a value returned by any routine which sets the drawing mode or one of the constants below.

Permitted values for parameter mode	
NORMAL	Default: Draws points, lines, areas, bitmaps.
XOR	Inverts points, lines, areas when overwriting the color of another object on the display.

Return value

The selected drawing mode.

Add. information

In addition to setting the drawing mode, this routine may also be used to restore a drawing mode that has previously been changed.

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);
```

Screen shot of above example



7.3 Query current client rectangle

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	Meaning
pRect	Pointer to GUI_RECT-structure to store result.

7.4 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_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	Meaning
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.

Related topics

[GUI_InvertRect\(\)](#), [GUI_FillRect\(\)](#)

GUI_DrawPixel()

Description

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

Prototype

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

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

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	Meaning
x	X-position of point.
y	Y-position of point.

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

GUI_DrawRectEx()

Description

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

Prototype

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

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

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

Add. 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 rectangular area at a specified position in the current window.

Prototype

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

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

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

Related topics

[GUI_FillRect\(\)](#), [GUI_ClearRect\(\)](#)

7.5 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 64Kb. If larger bitmaps should be displayed with a 16 bit CPU please refer to chapter "High-Level Configuration\Available GUI configuration macros".

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	Meaning
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.

Add. information

The picture data is interpreted as bitstream starting with the most significant bit (msb) of the first byte.

A new line always starts at an even byte address, as the nth 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. For more information, please refer to Chapter 10: "Bitmap Converter".

Example

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

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

Screen shot of above example



GUI_DrawBitmapExp()

Description

Same function as `GUI_DrawBitmap()`, but with additional parameters.

Prototype

```
void GUI_DrawBitmapExp(int x0, int y0,
                      int XSize, int YSize,
                      int XMul, int YMul,
                      int BitsPerPixel,
                      int BytesPerLine,
                      const U8* pData,
                      const GUI_LOGPALETTE* pPal);
```

Parameter	Meaning
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.
Xsize	Number of pixels in horizontal direction. Valid range: 1 ... 255.
Ysize	Number of pixels in vertical direction. Valid range: 1 ... 255.
XMUL	Scale factor of X-direction.
YMUL	Scale factor of Y-direction.
BitsPerPixel	Number of bits per pixel.
BytesPerLine	Number of bytes per line of the image.
pData	Pointer to the actual image, the data that defines what the bitmap looks like.
pPal	Pointer to a <code>GUI_LOGPALETTE</code> structure.

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	Meaning
pBM	Pointer to the bitmap to display.
x0	X-position of the anker point in the display.
y0	Y-position of the anker point in the display.
xCenter	X-position of the anker point in the bitmap.
yCenter	Y-position of the anker point in the bitmap.
xMag	Scale factor of X-direction.
yMag	Scale factor of Y-direction.

Add. information

A negative value of the `xMag`-parameter would mirror the bitmap in the X-axis and a negative value of the `yMag`-parameter would mirror the bitmap in the Y-axis. The unit of `xMag`- and `yMag` are thousandth. The position given by the parameter `xCenter` and `yCenter` specifies the pixel of the bitmap which should be displayed at the display at position `x0/y0` independent of scaling or mirroring.

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

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	Meaning
<code>pBM</code>	Pointer to the bitmap to display.
<code>x0</code>	X-position of the upper left corner of the bitmap in the display.
<code>y0</code>	Y-position of the upper left corner of the bitmap in the display.
<code>XMul</code>	Magnification factor of X-direction.
<code>YMul</code>	Magnification factor of Y-direction.

GUI_DrawStreamedBitmap()

Description

Draws a bitmap from a data bitmap data stream.

Prototype

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

Parameter	Meaning
<code>pBMH</code>	Pointer to the data stream.
<code>x</code>	X-position of the upper left corner of the bitmap in the display.
<code>y</code>	Y-position of the upper left corner of the bitmap in the display.

Add. information

You can use the bitmap converter (Chapter 10) to create bitmap data streams. The format of these streams is not the same as the format of a .bmp file.

7.6 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	Meaning
<code>y</code>	Y-position.
<code>x0</code>	X-starting position.
<code>x1</code>	X-end position.

Add. 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	Meaning
<code>x0</code>	X-starting position.
<code>y0</code>	Y-starting position.
<code>x1</code>	X-end position.
<code>y1</code>	Y-end position.

Add. 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	Meaning
<code>dx</code>	Distance in X-direction to end of line to draw.
<code>dy</code>	Distance in Y-direction end of line to draw.

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	Meaning
x	X-end position.
y	Y-end position.

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	Meaning
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.

Add. 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 endpoint in the current window.

Prototype

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

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

Add. 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	Meaning
dx	Distance to move in X.
dy	Distance to move in Y.

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	Meaning
x	New position in X.
y	New position in Y.

GUI_SetLineStyle()

Description

Sets the current line style used by the function `GUI_DrawLine`.

Prototype

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

Parameter	Meaning
LineStyle	New line style to be used (see table below).

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

Add. information

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

7.7 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	Meaning
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.

Add. 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	Meaning
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.

Add. information

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

Example

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

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

GUI_POINT aEnlargedPoints[countof(aPoints)];

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

Screen shot 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	Meaning
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.

Add. information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. The endpoint must not touch the outline of the polygon.

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	Meaning
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.

Add. 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. Whereas setting the parameter `Len` to 1 will enlarge the polygon by one pixel on all sides, setting the parameter `Mag` to 1 will have no effect.

Example

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

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

```

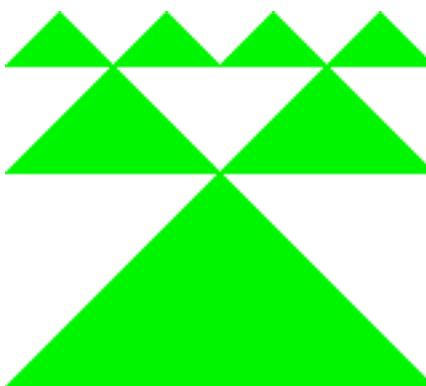
    { 40, 20},
    { 20, 0}
};

GUI_POINT aMagnifiedPoints[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, countof(aPoints), Mag);
        for (i = Count; i > 0; i--, x += 40 * Mag) {
            GUI_FillPolygon(aMagnifiedPoints, countof(aPoints), x, y);
        }
        y += 20 * Mag;
    }
}

```

Screen shot 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	Meaning
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.

Add. 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 samples shipped with emWin.

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*          emWin sample code
*****
*****
```

```
-----
File      : 2DGL_DrawPolygon.c
Purpose   : Example for drawing a polygon
-----
*/
#include "gui.h"

/********************* The points of the arrow *****/
/*
*           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 *****/
/*
*           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 *****/
/*
*           main
*
*****
*/
void main(void) {
    GUI_Init();
    DrawPolygon();
    while(1)
        GUI_Delay(100);
}
```

Screen shot of above example



7.8 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	Meaning
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). Minimum: 0 (will result in a point); maximum: 180.

Add. information

This routine cannot handle a radius in excess of 180 because it uses integer calculations that would otherwise produce an overflow. However, for most embedded applications this is not a problem since a circle with diameter 360 is larger than the display anyhow.

Example

```
// Draw concentric circles
void ShowCircles(void) {
    int i;
    for (i=10; i<50; i++)
        GUI_DrawCircle(120,60,i);
}
```

Screen shot 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	Meaning
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). Minimum: 0 (will result in a point); maximum: 180.

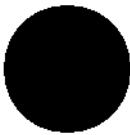
Add. information

This routine cannot handle a radius in excess of 180.

Example

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

Screen shot of above example



7.9 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	Meaning
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). Minimum: 0; maximum: 180.
ry	Y-radius of the ellipse (half the diameter). Minimum: 0; maximum: 180.

Add. information

This routine cannot handle rx/ry parameters in excess of 180 because it uses integer calculations that would otherwise produce an overflow.

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	Meaning
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). Minimum: 0; maximum: 180.
ry	Y-radius of the ellipse (half the diameter). Minimum: 0; maximum: 180.

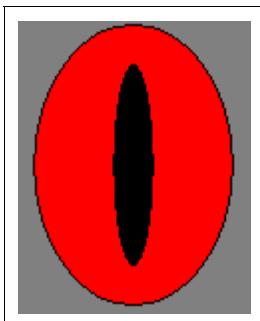
Add. information

This routine cannot handle a rx/ry parameters in excess of 180.

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);
```

Screen shot of above example



7.10 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 GL_DrawArc (int xCenter, int yCenter, int rx, int ry, int a0, int a1);
```

Parameter	Meaning
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).

Limitations

Currently the ry parameter is not used. The rx parameter is used instead.

Add. information

GUI_DrawArc() uses the floating-point library. It cannot handle rx/ry parameters in excess of 180 because it uses integer calculations that would otherwise produce an overflow.

Example

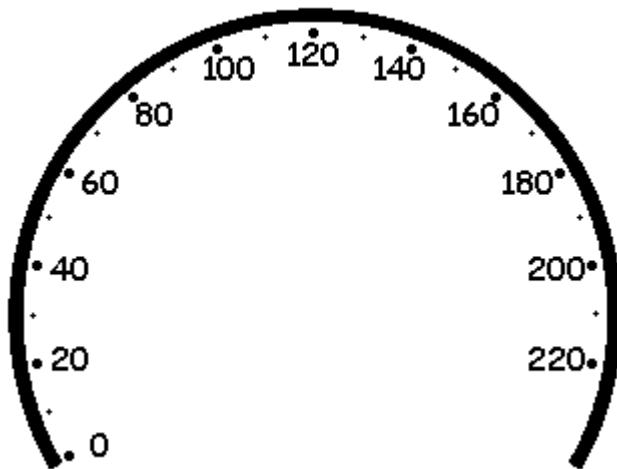
```
void DrawArcScale(void) {
    int x0 = 160;
    int y0 = 180;
    int i;
    char ac[4];
    GUI_SetBkColor(GUI_WHITE);
    GUI_Clear();
```

```

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);
    }
}
}

```

Screen shot of above example



7.11 Drawing graphs

GUI_DrawGraph()

Description

Draws a graph at once.

Prototype

```
void GUI_DrawGraph(I16 *paY, int NumPoints, int x0, int y0);
```

Parameter	Meaning
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.

Add. 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.

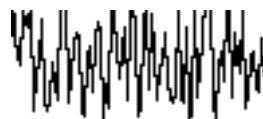
Example

```
#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);
}
```

Screen shot of above example



7.12 Saving and restoring the GUI-context

GUI_RestoreContext()

Description

The function restores the GUI-context.

Prototype

```
void GUI_RestoreContext(const GUI_CONTEXT* pContext);
```

Parameter	Meaning
<code>pContext</code>	Pointer to a <code>GUI_CONTEXT</code> structure containing the new context.

Add. 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 and 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	Meaning
<code>pContext</code>	Pointer to a <code>GUI_CONTEXT</code> structure for saving the current context.

7.13 Clipping

GUI_SetClipRect()

Description

Sets the clipping rectangle used for limiting the output.

Prototype

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

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

Add. information

The clipping area is per default limited to the configured (virtual) display size. 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.

Sample

The following sample shows how to use the function:

```
GUI_RECT Rect = {10, 10, 100, 100};  
GUI_SetClipRect(&Rect);  
. /* Use the clipping area ... */  
.  
GUI_SetClipRect(NULL);
```

Chapter 8

Displaying bitmap files

The most recommended and most efficient way to display a bitmap is to convert it into a GUI_BITMAP structure using the bitmap converter and to add the source to your project. If your application needs to display images not known at compile time the file support routines could be used.
emWin currently supports the formats described in this chapter.

8.1 BMP file support

Although bitmaps which can be used with emWin are normally defined as `GUI_BITMAP` structures in "C", 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 (i.e. a company logo) it is much more efficient to define them as `GUI_BITMAP` structures that can be used directly by emWin. This may be easily done with the Bitmap Converter.

8.1.1 BMP file API

The table below lists the available BMP file related routines in alphabetical order. Detailed descriptions follows:

Routine	Explanation
<code>GUI_BMP_Draw()</code>	Draws a bitmap file which has been loaded into memory.
<code>GUI_BMP_GetXSize()</code>	Returns the X-size of a bitmap loaded into memory.
<code>GUI_BMP_GetYSize()</code>	Returns the Y-size of a bitmap loaded into memory.
<code>GUI_BMP_Serialize()</code>	Creates a BMP-file.
<code>GUI_BMP_SerializeEx()</code>	Creates a BMP-file from the given rectangle.

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* pBMP, int x0, int y0);
```

Parameter	Meaning
<code>pBMP</code>	Pointer to the start of the memory area in which the .bmp file resides.
<code>x0</code>	X-position of the upper left corner of the bitmap in the display.
<code>y0</code>	Y-position of the upper left corner of the bitmap in the display.

Add. information

There are different types of .bmp files. Supported types are 1/4/8/24 bpp files. These are the most common file types; other types and compression are not supported.

The samples shipped with emWin contain an example that uses this function to show all files in the Windows system directory.

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* pBMP);
```

Parameter	Meaning
<code>pBMP</code>	Pointer to the start of the memory area in which the .bmp file resides.

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* pBMP);
```

Parameter	Meaning
pBMP	Pointer to the start of the memory area in which the .bmp file resides.

Return value

Y-size of the bitmap.

GUI_BMP_Serialize()

Description

The function creates a BMP-file containing the complete contents of the LCD.

Prototype

```
void GUI_BMP_Serialize(GUI_CALLBACK_VOID_U8_P * pfSerialize, void * p);
```

Parameter	Meaning
pfSerialize	Pointer to serialization function
p	Pointer to user defined data passed to serialization function

Add. information

The following sample will show 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.

Prototype

```
void GUI_BMP_SerializeEx(GUI_CALLBACK_VOID_U8_P * pfSerialize,
    int x0, int y0,
```

```
int xSize, int ySize, void * p);
```

Parameter	Meaning
pfSerialize	Pointer to serialization function.
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.

Add. information

(See GUI_BMP_Serialize)

8.2 JPEG file support

The JPEG file support of emWin is based on the Independent JPEG Group's free JPEG software. It can be found in the GUI\JPEG subdirectory. For copyright information please take a look to the file GUI\JPEG\DOC\README. The library contains C software to implement JPEG image decompression. 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.

Please note, that the JPEG file support of emWin is only part of the color version. The JPEG package is not shipped with monochrome versions.

8.2.1 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 to the free JPEG software. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.) So far as we are aware, there are no patent restrictions on the remaining code.

The library does not contain provision for supporting the hierarchical or lossless processes defined in the standard.

8.2.2 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 sub-folder. 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.

Sample

The following sample 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);
    ...
}
```

8.2.3 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 please take a look to the memory device chapter.

8.2.4 Memory usage

The JPEG library needs malloc/free to allocate about 16Kb work area for decompression.

8.2.5 JPEG file API

The table below lists the available JPEG file related routines in alphabetical order. Detailed descriptions follows:

Routine	Explanation
GUI_JPEG_Draw()	Draws a JPEG file which has been loaded into memory.
GUI_JPEG_GetInfo()	Fills a GUI_JPEG_INFO structure.

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 * pImageData, int DataSize, int x0, int y0);
```

Parameter	Meaning
pImageData	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.

Return value

Zero if success, nonzero if the function fails. (The current implementation always returns 0)

Add. information

The sample folder contains the sample `2DGL_DrawJPEG.c` which shows how to use the function.

GUI_JPEG_GetInfo()

Description

Fills a GUI_JPEG_INFO structure with information about the given image.

Prototype

```
int GUI_JPEG_GetInfo(const void * pImageData,
                      int DataSize,
                      GUI_JPEG_INFO* pInfo);
```

Parameter	Meaning
pImageData	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.

Return value

Zero if success, nonzero if the function fails. (The current implementation always returns 0)

Elements of GUI_JPEG_INFO

Data type	Element	Meaning
int	XSize	Pixel size in X of the image.
int	YSize	Pixel size in Y of the image.

Add. information

The sample folder contains the sample `2DGL_DrawJPG.c` which shows how to use the function.

8.3 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. The emWin GIF file support is limited to only one image per file. Unsupported data like raw text or application specific data will be ignored.

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.

8.3.1 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'.

8.3.2 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 please take a look to the memory device chapter.

8.3.3 Memory usage

The GIF decompression routine of emWin needs about 16Kb static work area for decompression.

8.3.4 GIF file API

The table below lists the available GIF file related routines in alphabetical order. Detailed descriptions follows:

Routine	Explanation
GUI_GIF_Draw()	Draws a GIF file which has been loaded into memory.
GUI_GIF_GetXSize()	Returns the X-size of a bitmap loaded into memory.
GUI_GIF_GetYSize()	Returns the Y-size of a bitmap loaded into memory.

GUI_GIF_Draw()

Description

Draws 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	Meaning
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.

Return value

Zero if success, nonzero if the function fails.

Add. information

The GIF support of emWin is limited to draw only one image per .gif file. If the file contains more than one image, the function shows only the first image of the file. Transparency and interlaced GIF images are supported.

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	Meaning
pGIF	Pointer to the start of the memory area in which the .gif file resides.

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	Meaning
pGIF	Pointer to the start of the memory area in which the .bmp file resides.

Return value

Y-size of the GIF image.

Chapter 9

Fonts

The most common fonts are shipped with emWin as standard fonts. In fact, you will probably find that these fonts are fully sufficient for your application. For detailed information on the individual fonts, please refer to the subchapter "Standard Fonts", which describes all fonts included with emWin and shows all characters as they appear on the display.

emWin supports ASCII, ISO 8859-1 and Unicode. Usually, 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 characters that are available depends on the selected font.

9.1 Supported font types

The current version of emWin supports 4 types of fonts: monospaced bitmap fonts, proportional bitmap fonts, proportional bitmap fonts with 2 bpp built-in antialiasing information and antialiased fonts with 4 bpp built-in antialiasing information. (For more information on antialiased fonts, see Chapter 22: "Antialiasing".)

All fonts linked with your application and declared (in `GUIConf.h`) may be selected. 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 (described in a separate manual) may be used to create additional fonts.

In order to be able to use a font in your application, the following requirements must be met:

- The font is in a form compatible with emWin as either "C" file, object file or library.
- The font file is linked with your application.
- The font declaration is contained in `GUIConf.h` (this is necessary in order to avoid compiler warnings due to undeclared externals).

9.2 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 sections that follow.

Routine	Explanation
Selection of a font	
<code>GUI_GetFont()</code>	Returns a pointer to the currently selected font.
<code>GUI_SetFont()</code>	Sets the current font.
<code>GUI_SIF_CreateFont()</code>	Creates and selects a font by passing a pointer to system independent font data.
<code>GUI_SIF_DeleteFont()</code>	Deletes a font created by <code>GUI_SIF_CreateFont()</code>
Font-related functions	
<code>GUI_GetCharDistX()</code>	Returns the width in pixels (X-size) of a specified character in the current font.
<code>GUI_GetFontDistY()</code>	Returns the Y-spacing of the current font.
<code>GUI_GetFontInfo()</code>	Returns a structure containing font information.
<code>GUI_GetFontSizeY()</code>	Returns the height in pixels (Y-size) of the current font.
<code>GUI_GetStringDistX()</code>	Returns the X-size of a text using the current font.
<code>GUI_GetTextExtend</code>	Evaluates the size of a text using the current font
<code>GUI_GetYDistOfFont()</code>	Returns the Y-spacing of a particular font.
<code>GUI_GetYSizeOfFont()</code>	Returns the Y-size of a particular font.
<code>GUI_IsInFont()</code>	Evaluates whether a specified character is in a particular font.

9.3 Selection of a font

emWin offers different fonts, one of which is always selected. This selection can be changed by calling the function `GUI_SetFont()`, which selects 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.

GUI_GetFont()

Description

Returns a pointer to the currently selected font.

Prototype

```
const GUI_FONT * GUI_GetFont(void)
```

GUI_SetFont()

Description

Sets the font to be used for text output.

Prototype

```
const GUI_FONT * GUI_SetFont(const GUI_FONT * pNewFont);
```

Parameter	Meaning
pFont	Pointer to the font to be selected and used.

Return value

Returns a pointer to the previously selected font so that it may be restored at a later point.

Examples

Displays sample text in 3 different sizes, restoring the former font afterwards:

```
void DispText(void) {
    const GUI_FONT GUI_FLASH* OldFont=GUI_SetFont(&GUI_Font8x16);
    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 font
}
```

Screen shot of above example:

This text is 8 by 16 pixels
This text is 6 by 8 pixels
This text is proportional

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
```

Screen shot of above example:

The result is: 42

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	Meaning
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.

Permitted values for element pFontType	
GUI_SIF_TYPE_PROP	Should be used if the parameter pFont points to a non antialiased proportional 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.

Add. 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 please refer to the font converter documentation.

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 */

void MainTask(void) {
    GUI_Init();
    GUI_SIF_CreateFont(_DownloadedFont, &_Font, GUI_SIF_TYPE_PROP);
    GUI_DispString("Hello World!");
    while (1) {
        GUI_Exec();
    }
}
```

GUI_SIF_DeleteFont()

Description

Deletes a font pointed by the parameter pFont.

Prototype

```
void GUI_SIF_DeleteFont(GUI_FONT * pFont);
```

Parameter	Meaning
pFont	Pointer to the font to be deleted.

Add. Information

After using a font created with GUI_SIF_CreateFont() the font should be deleted if not used anymore.

Example

```
GUI_FONT _Font; /* Font structure in RAM */
GUI_SIF_CreateFont(_DownloadedFont, &_Font, GUI_SIF_TYPE_PROP);
/*
   Use the font
```

```
*/  
GUI_SIF_DeleteFont(&_Font);
```

9.4 Font-related functions

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	Meaning
c	Character to calculate width from.

GUI_GetFontDistY()

Description

Returns the Y-spacing of the currently selected font.

Prototype

```
int GUI_GetFontDistY(void);
```

Add. 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	Meaning
pFont	Pointer to the font.
pfi	Pointer to a GUI_FONTINFO structure.

Add. 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);
```

Add. 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.

GUIGetStringDistX()

Description

Returns the X-size used to display a specified string in the currently selected font.

Prototype

```
int GUIGetStringDistX(const char GUI_FAR *s);
```

Parameter	Meaning
<code>s</code>	Pointer to the string.

GUIGetTextExtend()

Description

Calculates the size of a given string using the current font.

Prototype

```
void GUIGetTextExtend(GUI_RECT* pRect, const char* s, int Len);
```

Parameter	Meaning
<code>pRect</code>	Pointer to <code>GUI_RECT</code> -structure to store result.
<code>s</code>	Pointer to the string.
<code>Len</code>	Number of characters of the string.

GUIGetYDistOfFont()

Description

Returns the Y-spacing of a particular font.

Prototype

```
int GUIGetYDistOfFont(const GUI_FONT* pFont);
```

Parameter	Meaning
<code>pFont</code>	Pointer to the font.

Add. information

(see `GUI_GetFontDistY()`)

GUIGetYSizeOfFont()

Description

Returns the Y-size of a particular font.

Prototype

```
int GUI_GetYSizeOfFont(const GUI_FONT* pFont);
```

Parameter	Meaning
pFont	Pointer to the font.

Add. information

(see `GUI_GetFontSizeY()`)

GUI_IsInFont()

Description

Evaluates whether or not a particular font contains a specified character.

Prototype

```
char GUI_IsInFont(const GUI_FONT*pFont, U16 c);
```

Parameter	Meaning
pFont	Pointer to the font.
c	Character to be searched for.

Add. 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_DispatchString("GUI_FontD32 does not contain 'X'");
}
```

9.5 Character sets

9.5.1 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	{		}	~	

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.

9.5.2 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	ˊ
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

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

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

9.5.3 Unicode

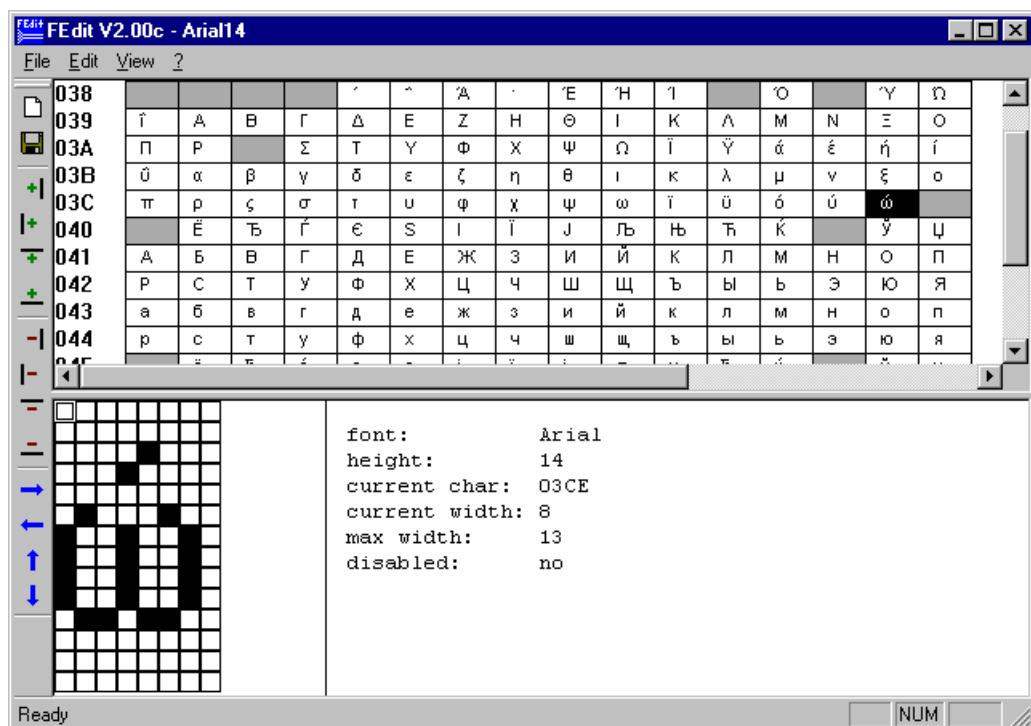
Unicode is the ultimate in character coding. It is an international standard based on ASCII and ISO 8859-1. Contrary to ASCII, UNICODE requires 16-bit characters because all characters have their own code. Currently, more than 30,000 different characters are defined. However, not all of the character images are defined in emWin. It is the responsibility of the user to define these additional characters. Please contact SEGGER Microcontroller Systeme GmbH or your distributor, as we may already have the character set that you need.

9.6 Font converter

Fonts which can be used with emWin must be defined as `GUI_FONT` structures in "C". The structures -- or rather the font data which is referenced by these structures -- can be rather large. It is very time-consuming and inefficient to generate these fonts manually. We therefore recommend using the font converter, which automatically generates "C" files from fonts.

The font converter is a simple Windows program. You need only to load an installed Windows font into the program, edit it if you want or have to, and save it as a "C" file. The "C" file may then be compiled, allowing the font to be shown on your display with emWin on demand.

The character codes 0x00 - 0x1F and 0x80 - 0x9F are disabled by default. The following is a sample screen shot of the font converter with a font loaded:



The font converter is described in a separate documentation which can be obtained by request from SEGGER Microcontroller Systeme GmbH (info@segger.com).

9.6.1 Adding fonts

Once you have created a font file and linked it to the project, declare the linked font as `extern const GUI_FONT`, as shown in the example below:

Example

```
extern const GUI_FONT GUI_FontNew;

int main(void) {
    GUI_Init();
    GUI_Clear();
    GUI_SetFont(&GUI_FontNew);
    GUI_DispString("Hello world\n");
    return 0;
}
```

9.7 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.

This chapter provides an overview of the standard emWin fonts.

9.7.1 Font identifier naming convention

All standard fonts are named as follows. The elements of the naming convention are then explained in the table:

`GUI_Font[<style>][<width>x]<height>[x<MagX>x<MagY>][H][B][_<characterset>]`

Element	Meaning
<code>GUI_Font</code>	Standard prefix for all fonts shipped with emWin.
<code><style></code>	Specifies a non-standard font style. Example: Comic style in <code>GUI_FontComic18B_ASCII</code> .
<code><width></code>	Width of characters, contained only in monspaced fonts.
<code><height></code>	Height of the font in pixels.
<code><MagX></code>	Factor of magnification in X, contained only in magnified fonts.
<code><MagY></code>	Factor of magnification in Y, contained only in magnified fonts.
<code>H</code>	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.
<code>B</code>	Abbreviation for "bold". Used in bold fonts.
<code><characterset></code>	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.

Example 1

`GUI_Font16_ASCII`

Element	Meaning
<code>GUI_Font</code>	Standard font prefix.
<code>16</code>	Height in pixels.
<code>ASCII</code>	Font contains ASCII characters only.

Example 2

`GUI_Font8x15B_ASCII`

Element	Meaning
<code>GUI_Font</code>	Standard font prefix.
<code>8</code>	Width of characters.
<code>x15</code>	Height in pixels.
<code>B</code>	Bold font.
<code>ASCII</code>	Font contains ASCII characters only.

Example 3

GUI_Font8x16x1x2

Element	Meaning
GUI_Font	Standard font prefix.
8	Width of characters.
x16	Height in pixels.
x1	Magnification factor in X.
x2	Magnification factor in Y.

9.7.2 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][<charerset>]

Element	Meaning
F	Standard prefix for all fonts files shipped with emWin.
<width>	Width of characters, contained only in monspaced 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.
<charerset>	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.

9.7.3 Measurement, ROM-size and character set of fonts

The following pages 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	Meaning
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".

9.7.4 Proportional fonts

GUI_Font8_ASCII

F: 08
B: 07
C: 07
L: 05
U: 01



!#\$%&?()*, -./0123456789;:<=>?@ABCDEFGHIJKLMNPQRSTUUVWX
YZ[.]^_ abcdefghijklmnopqrstuvwxyz({})~

ROM Size	1562 bytes
Used files	F08_ASCII.c

GUI_Font8 1

F: 08
B: 07
C: 07
L: 05
U: 01



ROM Size	1586 bytes + size of (GUI_Font8_ASCII)
Used files	F08_ASCII.c, F08_1.c

GUI Font10 ASCII

F: 10
B: 09
C: 08
L: 06
U: 01



!#%\$&'^+,-./0123456789;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ
WXYZYI^`abcdefghijklmnopqrstuvwxyz{}`

ROM Size	1800 bytes
Used files	F10 ASCII.c

GUI_Font10_1

F: 10
B: 09
C: 08
L: 06
U: 01



ROM Size	2456 bytes + size of (GUI_Font10_ASCII)
Used files	F10_1.c, F10_ASCII.c

GUI Font10S ASCII

F: 10
B: 08
C: 06
L: 04
U: 02



"#%&()*,./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[!`_`abcde
fghijklmnoqrstuvwxyz!`"

ROM Size	1760 bytes
Used files	F10 ASCII.c

GUI Font10S 1

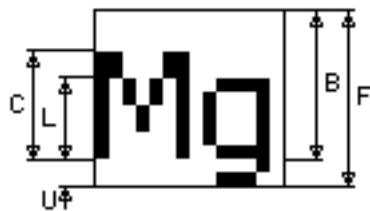
F: 10
B: 08
C: 06
L: 04
U: 02



ROM Size	1770 bytes + size of (GUI_Font10S_ASCII)
Used files	F10S_1.c F10_ASCII.c

GUI_Font13_ASCII

F: 13
B: 11
C: 08
L: 06
U: 02



!"#\$%&()'*+,.-/0123456789;:<=>?@ABCDEFGHIJKLMNPQRS
TUUVWXYZ[{}]^_`abcdefghijklmnopqrstuvwxyz{{}}~

ROM Size	2076 bytes
Used files	F13_ASCII.c

GUI_Font13_1

F: 13
B: 11
C: 08
L: 06
U: 02



!#\$%&()'*)+,.-./0123456789;:<=>?@ABCDEFGHIJKLMNPQRS
TUVWXYZ[{}]^_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£×¥¦§¨©³«¬
-®°±²³¹µ¶·¹⁰¼¼½¾ÀÁÂÂÃÂÆŒÉÊÉÍÍÍÐÑÒÓÔÔÔ×ØÙÙÙ
ÜÝþßáâââââæœéêéííñññòóôôô÷øùùùùýþ

ROM Size	2149 bytes + size of (GUI_Font13_ASCII)
Used files	F13_1.c, F13_ASCII.c

GUI Font13H ASCII

F: 13
B: 11
C: 09
L: 07
U: 02



!#\$%&'^)*+,-./0123456789;,<=>?@ABCDEFGHIJKL
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{ }~

ROM Size	2232 bytes
Used files	F13H ASCII.c

GUI_Font13H_1

F: 13
B: 11
C: 09
L: 07
U: 02



ROM Size	2291bytes + size of (GUI_Font13H_ASCII)
Used files	F13H_1.c, F13H_ASCII.c

GUI Font13HB ASCII

F: 13
B: 11
C: 09
L: 07
U: 02



!#\$%&'()*+,-./0123456789;:<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

ROM Size	2690 bytes
Used files	F13HB ASCII.c

GUI Font13HB 1

F: 13
B: 11
C: 09
L: 07
U: 02



ROM Size	2806 bytes + size of (GUI_Font13HB_ASCII)
Used files	F13HB_1.c, F13HB_ASCII.c

GUI Font16 ASCII

F: 16
B: 13
C: 10
L: 07
U: 03



!"#\$%&'()*+,./0123456789;:<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}{|}

ROM Size	2714 bytes
Used files	F16_ASCII.c

GUI Font16 1

F: 16
B: 13
C: 10
L: 07
U: 03



ROM Size	3850 bytes + size of (GUI_Font16_ASCII)
Used files	F16_1.c, F16_ASCII.c

GUI_Font16_HK

ああいいうええおおかがきぎくぐけげこご
さざしじすずせぜそぞただぢぢっつづでと
どなにぬねのはばばひびびふぶぶへべべほぼ
ぼまみむめもややゅゆよよらりるれろわわゐ
ゑをんアアイイウウェエオオカガキギクグケ
ゲコゴサザシジスズセゼソゾタダチヂツツ
テデトドナニヌネノハババヒビビフブブヘベ
ペホボボマミムメモヤヤュヨヨラリルレロ
ワワヰエヲンヴカケ

ROM Size	6950 bytes
Used files	F16.HK.c

GUI Font16 1HK

F: 16
B: 13
C: 10
L: 07
U: 03



ROM Size	120 bytes + size of (GUI_Font16_HK) + size of (GUI_Font16_1)
Used files	F16_1HK.c, F16_HK.c, F16_1.c, F16_ASCII.c

GUI_Font16B_ASCII

F: 16
B: 13
C: 10
L: 07
U: 03



!%\$%^&'^0^+,-./0123456789;:<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[N]^_`abcdefghijklmnopqrstuvwxyz{}`

ROM Size	2690 bytes
Used files	F16B_ASCII.c

GUI Font16B 1

F: 16
B: 13
C: 10
L: 07
U: 03



ROM Size	2790 bytes + size of (GUI_Font16B_ASCII)
Used files	F16B_1.c, F16B_ASCII.c

GUI FontComic18B ASCII

F: 18
B: 15
C: 12
L: 09
U: 03



!"#\$%&'0*+,./0123456789:;<=>?@ABCD
EFGHIJKLMNOPQRSTUVWXYZ[\]^_`ab
cdefghijklmnopqrstuvwxyz[{}]{}`€

ROM Size	3572 bytes
Used files	FComic18B_ASCII.c

GUI FontComic18B 1

F: 18
B: 15
C: 12
L: 09
U: 03



ROM Size	4334 bytes + size of(GUI_FontComic18b_ASCII)
Used files	FComic18B_1c, FComic18B_ASCII.c

GUI Font24 ASCII

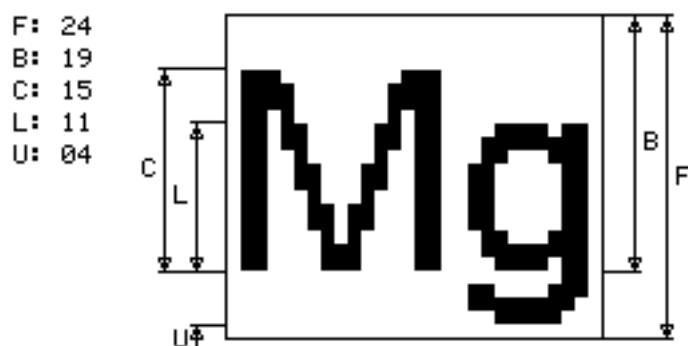
F: 24
B: 19
C: 15
L: 11
U: 04



!'"#\$%&'()^+,-./0123456789;:<=>
?@ABCDEFGHIJKLMNPQRST
UVWXYZ[\]^_`abcdefghijklmnopqr
stuvwxyz[|]~

ROM Size	4786 bytes
Used files	F24_ASCII.c

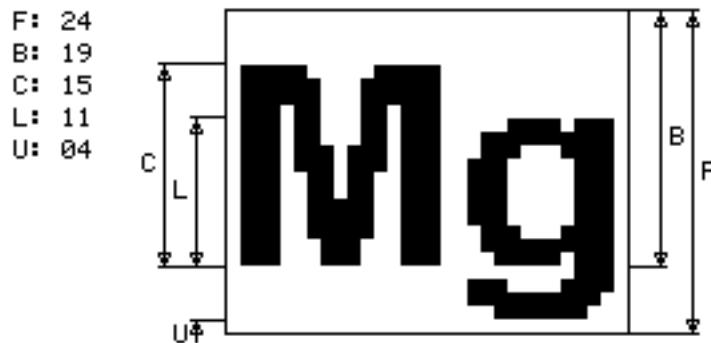
GUI Font24 1



!#\$%&'()*+,./0123456789;:<=>
?@ABCDEFGHIJKLMNOPQRST
UVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{||}~ ¡¢£¤¥¡§¤©ª«¬-®¬°±²
³ºµ¶. º¹»¼¼³¼¿ÀÁÂÃÄÅÆÇÈÉÊË
ÍÏÏÐÑÒÓÔÔÖÖ×ØÙÚÛÜÝþÞàáâãä
åæçèéêëíïðñòóôôöö÷øùúûüýþ

ROM Size	5022 bytes + size of(GUI_Font24_ASCII)
Used files	F24_ASCII.c, F24_1.c

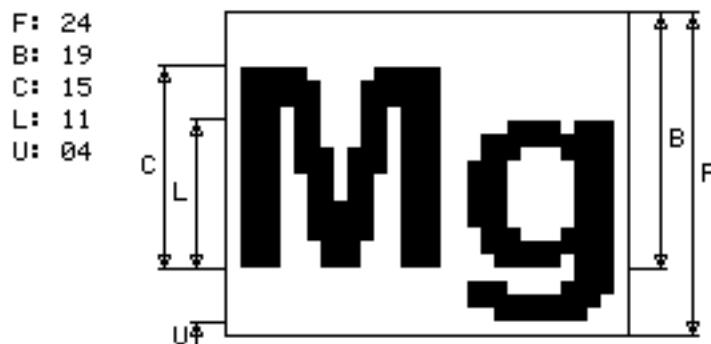
GUI_Font24B_ASCII



!'"#\$%&'()>*+,./0123456789:;=>
?@ABCDEFGHIJKLMNPQRST
UVWXYZ[]^_`abcdefghijklmnp
qrstuvwxyz{}~-

ROM Size	4858 bytes
Used files	F24B_ASCII.c

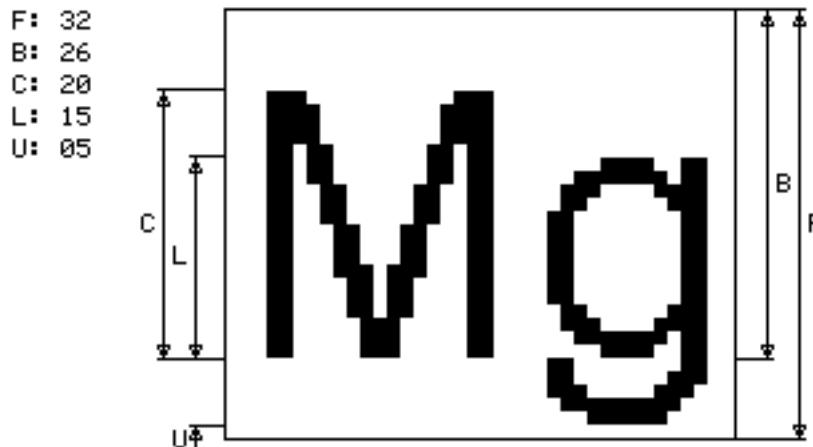
GUI_Font24B_1



!"#\$%&'()*+,./0123456789:;<=>
?@ABCDEFGHIJKLMNPQRST
UVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{[]}`íç£¤¥!§©ª«¬-®
°±²³°µ¶·¹º»¼¼¾¿ÀÁÃÄÅÆÇÈ
ÉÊËÏÍÐÑÒÓÔÖÖ×ØÙÚÛÜÝÞßà
åâãäåæçèéêëïíðñòóôöö÷øùúû
üýþ

ROM Size	5022 bytes + size of (GUI_Font24B_ASCII)
Used files	F24B_ASCII.c, F24B_1.c

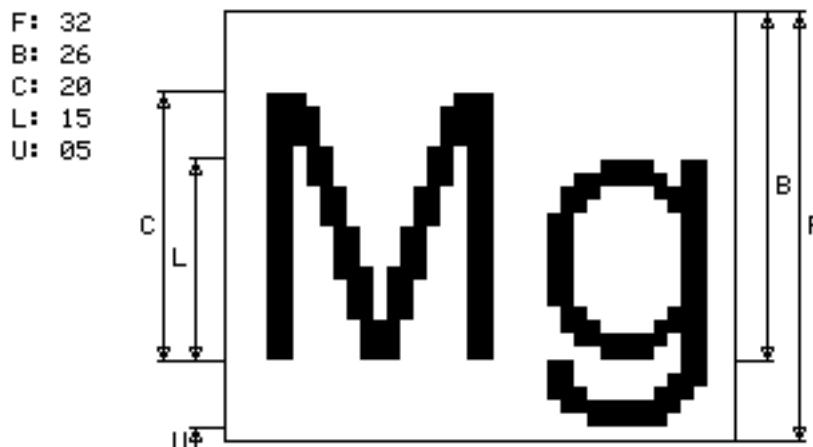
GUI_Font32_ASCII



!"#\$%&'()*+,-./012345678
9.;<=>?@ABCDEFGHIJK
LMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnpqrstuvwxyz
vwxyz{}~

ROM Size	7234 bytes
Used files	F32_ASCII.c

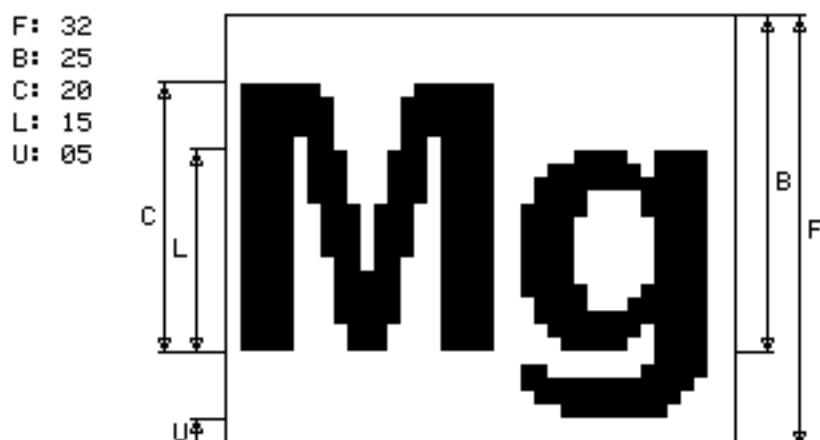
GUI_Font32_1



!"#\$%&'()*+,./012345678
 9.;<=>?@ABCDEFGHIJK
 LMNOPQRSTUVWXYZ[\]
 ^_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¤¥!§®«¬-®
 °±²³'µ¶,¹º»¼¼¾¿ÀÁÂÃ
 ÄÅÆÇÉÉÉÉÍÍÍÐÑÒÓÔÕ
 Ö×ØÙÙÙÙÝÞÞàáâãäåæç
 èéêëíññòóôõö÷øùúûüýþ

ROM Size	7734 bytes + size of (GUI_Font32_ASCII)
Used files	F32_ASCII.c, F32_1.c

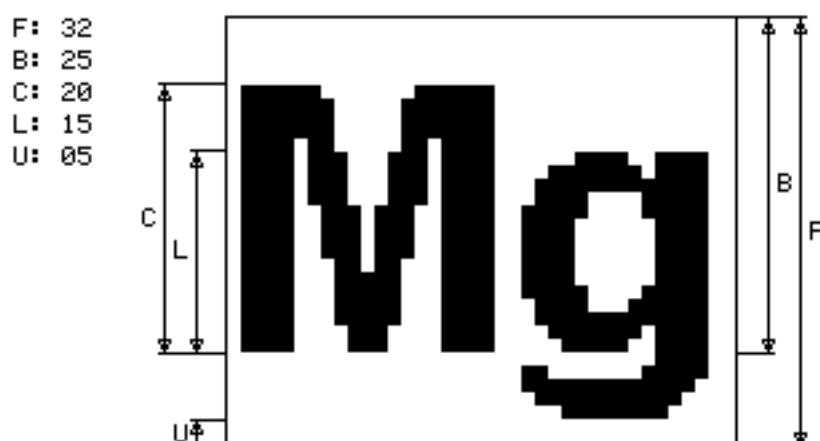
GUI_Font32B_ASCII



!"#\$%&'()*+,-./01234567
89::;<=>?@ABCDEFGHI
JKLMNOPQRSTUVWXYZ
Z[\]^_`abcdefghijklmn
opqrstuvwxyz{|}~

ROM Size	7842 bytes
Used files	F32B_ASCII.c

GUI_Font32B_1



!"#\$%&'()*+,./01234567
 89:;<=>?@ABCDEFGHI
JKLMNOPQRSTUVWXYZ
Z[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~ ¡¢£¤¥!§
 $\text{°}\text{©}^{\text{a}}\text{«}\text{¬}\text{‐}\text{®}\text{‐}\text{°}\text{±}^{23}\text{‐}\text{μ}\text{¶}\text{‐}\text{‐}^{10}\text{»}\text{‐}\frac{1}{4}\text{‐}\frac{1}{2}\text{‐}\frac{3}{4}$
ÀÁÂÃÄÅÆÇÈÉÊËÍÏÐÑ
ÒÓÔÕÖ×ØÙÚÛÜÝÞßàá
âääåæçèéêëíïðñòóôõö
÷øùúûüýþ

ROM Size	8118 bytes + size of (GUI_Font32B_ASCII)
Used files	F32B_ASCII.c, F32B_1.c

GUI_FontComic24B_ASCII

F: 24
 B: 20
 C: 17
 L: 13
 U: 04



!"#\$%&'0*+,./0123456789
 :;<=>?@ABCDEFGHIJKLMNP
QRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~

ROM Size	6146 bytes
Used files	FComic24B_ASCII.c

GUI_FontComic24B_1

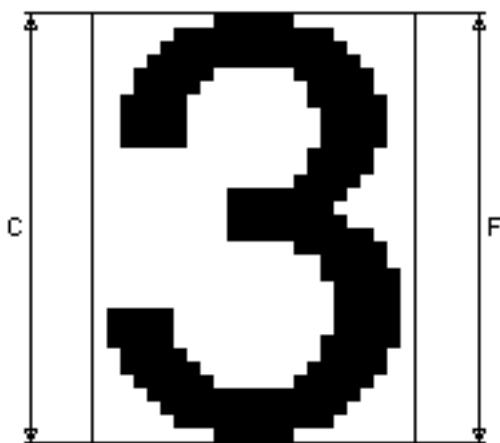
F: 24
B: 20
C: 17
L: 13
U: 04



ROM Size	5598 bytes + size of (GUI_Font24_ASCII)
Used files	FComic24B_1c, FComic24B_ASCII.c

GUI_FontD32

F: 32
C: 32

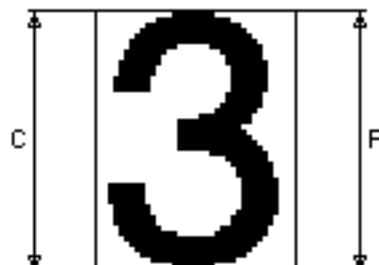


+-.012345678
9:

ROM Size	1574 bytes
Used files	FD32.c

GUI_FontD48

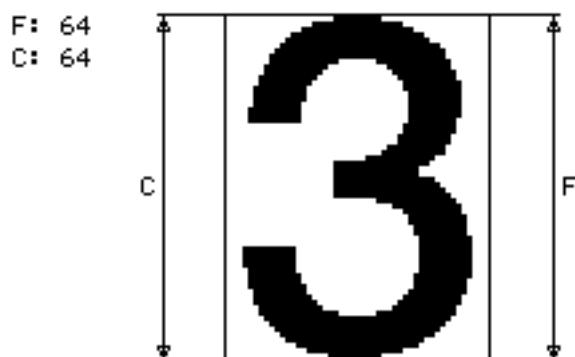
F: 48
C: 48



+-.01234
56789:

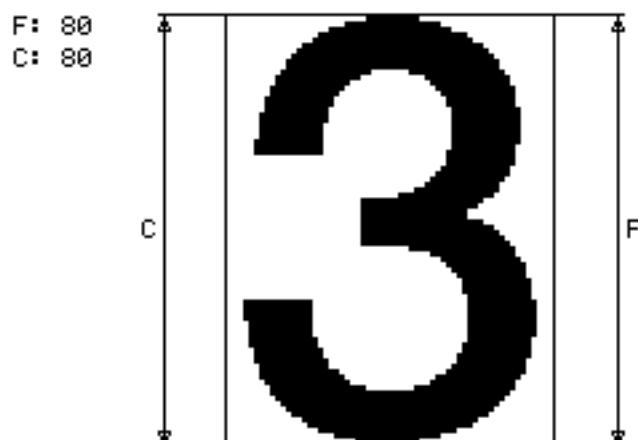
ROM Size	3512 bytes
Used files	FD48.c

GUI_FontD64



+-.012
345678
9:

ROM Size	5384 bytes
Used files	FD64.c

GUI_FontD80

+ - . 0 1
2 3 4 5 6
7 8 9 :

ROM Size	8840 bytes
Used files	FD80.c

9.7.5 Monospaced fonts

GUI_Font4x6

F: 06
B: 05
C: 05
L: 04
U: 01



```
!"#$%&'()**+,.-./0123456789:;=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ^_`ab<defghi,jklmn
apqrStuvwxyz)<<"
```

ROM Size	620 bytes
Used files	F4x6.c

GUI_Font6x8

F: 08
B: 07
C: 07
L: 05
U: 01



```
!"#$%&'()**+,.-./0123456789:;=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ^_`ab<defghi,jklmn
apqrStuvwxyz)<<"
```

ROM Size	1840 bytes
Used files	F6x8.c

GUI_Font6x9

F: 09
B: 07
C: 07
L: 05
U: 01



```
!"#$%&'()**+,.-./0123456789:;=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ^_`ab<defghi,jklmn
apqrStuvwxyz)<<"
```

ROM Size	This font uses the same ROM location as GUI_Font6x8.
Used files	F6x8.c

GUI_Font8x8

F: 08
B: 07
C: 07
L: 05
U: 01



ROM Size	1840 bytes
Used files	F8x8.c

GUI_Font8x9

F: 09
B: 07
C: 07
L: 05
U: 01



ROM Size	This font uses the same ROM location as GUI_Font8x8.
Used files	F8x8.c

GUI_Font8x10_ASCII

F: 10
B: 09
C: 09
L: 07
U: 01



! "#\$%& <>**+ - /0123456789: ;<=>?@ABCDEF
GHIJKLMNOPQRSTUVWXYZ[\]^_ abcdefghijklm
nopqrstuvwxyz{ }~^`

ROM Size	1770 bytes
Used files	F8x10_ASCII.c

GUI Font8x12 ASCII

F: 12
B: 10
C: 09
L: 06
U: 02



! "#\$%&'(>)*+, -./0123456789:;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[\]^_` abcdefghijklm nopqrstuvwxyz{ }~^△

ROM Size	1962 bytes
Used files	F8x12_ASCII.c

GUI Font8x13 ASCII

F: 13
B: 11
C: 09
L: 06
U: 02



!"#\$%&'()!*,-./0123456789:;,<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~|

ROM Size	2058 bytes
Used files	F8x13_ASCII.c

GUI Font8x13 1

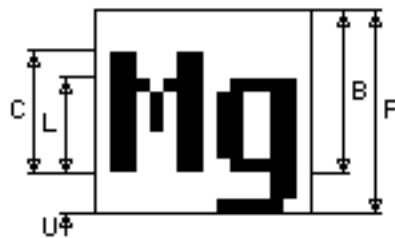
F: 13
B: 11
C: 09
L: 06
U: 02



ROM Size	2070 bytes + size of (GUI_Font8x13_ASCII)
Used files	F8x13_1.c, F8x13_ASCII.c

GUI_Font8x15B_ASCII

F: 15
B: 12
C: 09
L: 07
U: 03

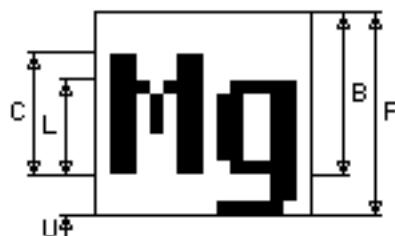


!\"#\$%&'()**+, -./0123456789:;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[\`]^_`abcdefghijklmnopqrstuvwxyz{|}~■

ROM Size	2250 bytes
Used files	F8x15B_ASCII.c

GUI_Font8x15B_1

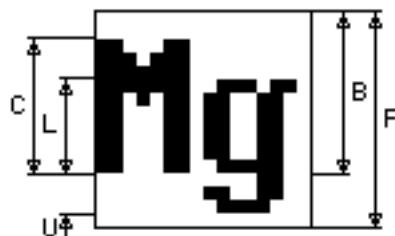
F: 15
B: 12
C: 09
L: 07
U: 03



ROM Size	2262 bytes + size of (GUI_Font8x15B_ASCII)
Used files	F8x15B_1.c, F8x15B_ASCII.c

GUI Font8x16

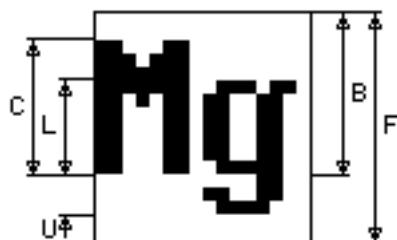
F: 16
B: 12
C: 10
L: 07
U: 03



ROM Size	3304 bytes
Used files	F8x16.c

GUI Font8x17

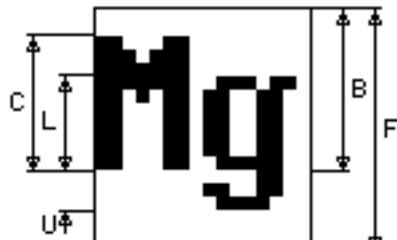
F: 17
B: 12
C: 10
L: 07
U: 03



ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI_Font8x18

F: 18
B: 12
C: 10
L: 07
U: 03



ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI Font8x16x1x2

Measurement	F:32 B:24 C:20 L:14 U:6
ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI Font8x16x2x2

! "#\$%&' ()**+, -./012
3456789: ;<=>?@ABCDE
FGHIJKLMNOPQRSTUVWXYZ
YZ[\]^_`abcdefghijklmnopqrstuvwxyz{ | }~

Not all characters are displayed. The character set is the same as that of GUI Font8x16.

Measurement	F:32 B:24 C:20 L:14 U: 6
ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI_Font8x16x3x3

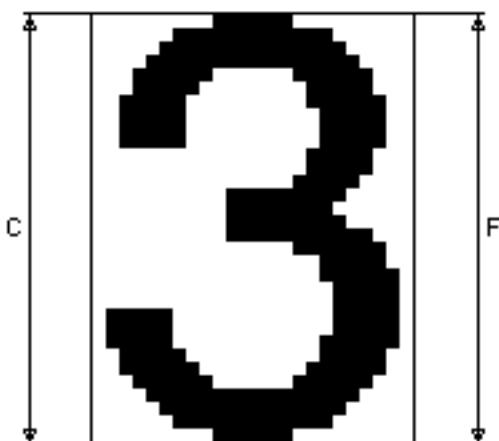
! " # \$ % & ' () * + ,
- . / 0 1 2 3 4 5 6 7 8 9
: ; < = > ? @ A B C D E F
G H I J K L M N O P Q R S

Not all characters are displayed. The character set is the same as that of GUI_Font8x16.

Measurement	F:48 B:36 C:30 L:21 U: 9
ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI_FontD24x32

F: 32
C: 32



+-.01234567
89:

ROM Size	1606 bytes
Used files	FD24x32.c

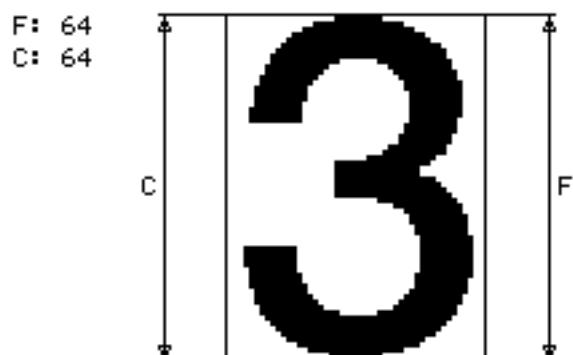
GUI_FontD36x48

F: 48
C: 48



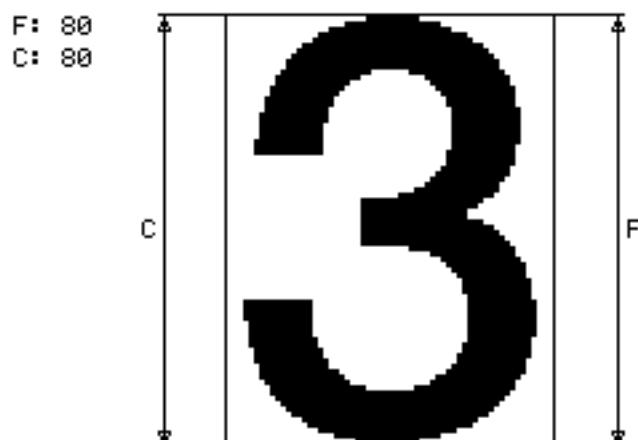
+-.0123
456789:

ROM Size	3800 bytes
Used files	FD36x48.c

GUI_FontD48x64

+ - . 0 1
2 3 4 5 6 7
8 9 :
:

ROM Size	5960 bytes
Used files	FD48x64.c

GUI_FontD60x80

+ - . 0
1 2 3 4 5
6 7 8 9 :
:

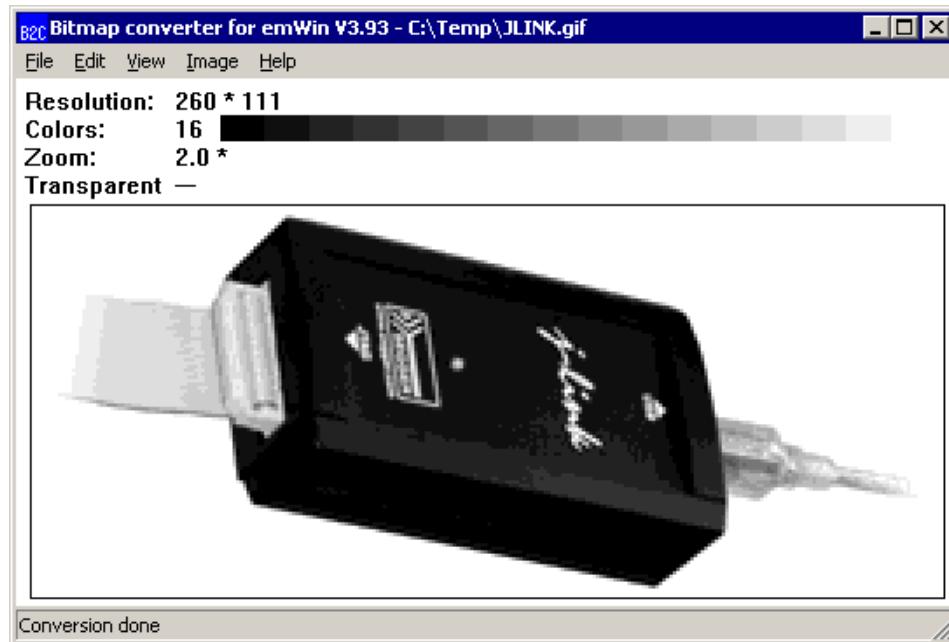
ROM Size	9800 bytes
Used files	FD60x80.c

Chapter 10

Bitmap Converter

The bitmap converter is a Windows program which is easy to use. Simply load a bitmap (in the form of a .bmp or a .gif file) into the application. Convert the color format if you want or have to, and convert it into a "C" file by saving it in the appropriate format. The "C" file may then be compiled, allowing the image to be shown on your display with emWin.

Screenshot of the Bitmap Converter



10.1 What it does

The bitmap converter is 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, especially if you are dealing with images of considerable size and with multiple shades of gray or colors. We therefore recommend using the bitmap converter, which automatically generates "C" files from bitmaps.

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 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 file formats

The bitmap converter basically supports 2 file formats: Windows bitmap files (*.bmp) and "Graphic Interchange Format" (*.gif):

Windows Bitmap Files

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

The bitmap converter supports reading of one image per GIF file. If the file for example contains a movie consisting of more than one image, the converter reads only the first image.

Transparency and interlaced GIF images are supported by the converter.

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 (i.e. .jpg, .jpeg, .png, .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.

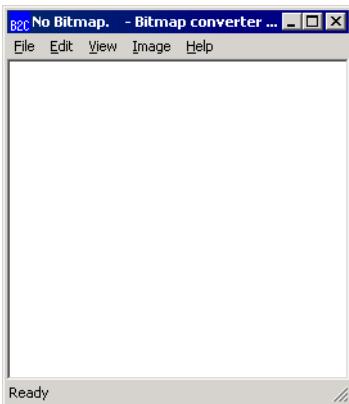
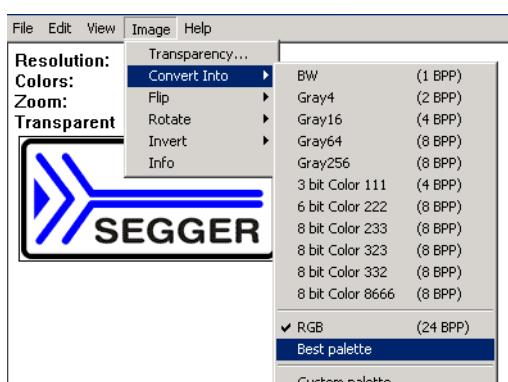
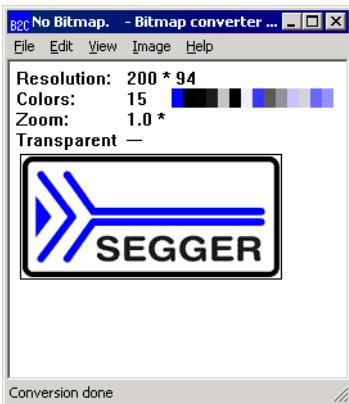
10.3 Generating "C" files from bitmaps

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. When using full-color bitmaps, it will be necessary to convert the image into a palette format, as the bitmap converter cannot generate "C" files from bitmaps in full-color mode.

The bitmap may be saved as a `.bmp` 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 displays 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.

The basic procedure for using the bitmap converter is illustrated below:

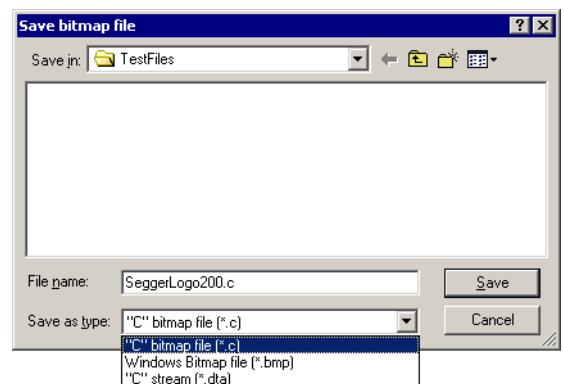
<p>Step 1: Start the application.</p> <p>The bitmap converter is opened showing an empty window.</p>	
<p>Step 2: Load a bitmap into the bitmap converter.</p> <p>Choose File/Open.</p> <p>Locate the document you want to open and click Open (must be a .bmp file).</p> <p>In this example, the file SeggerLogo200.bmp is chosen.</p>  <p>The bitmap converter displays the loaded bitmap.</p>	 <p>In this example, the loaded bitmap is in full-color mode. It must be converted to a palette format before a "C" file can be generated.</p>
<p>Step 3: Convert the image if necessary.</p> <p>Choose Image/Convert Into.</p> <p>Select the desired palette.</p> <p>In this example, the option Best palette is chosen.</p>  <p>The bitmap converter displays the converted bitmap.</p>	 <p>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.</p>

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."

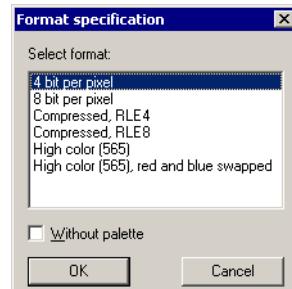
Click Save.



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.



10.4 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.

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.

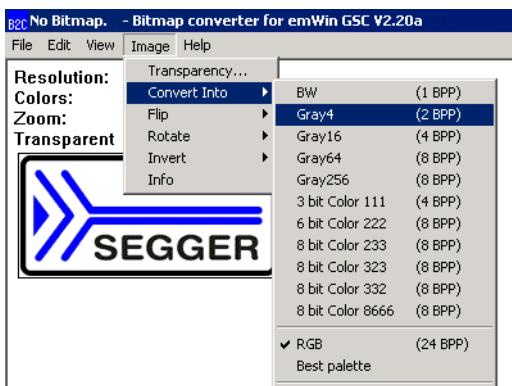
The procedure for conversion would be as follows:

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.



Choose Image/Convert Into/Gray4.



The bitmap converter displays the converted bitmap.



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.

10.5 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.

If you want to save a bitmap using RLE compression, you can do so by selecting one of the compressed 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; it works in the same way as displaying uncompressed bitmaps.

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 $9,400/3,803 = 2.47$.

10.6 Using a custom palette

Under certain circumstances it may be desirable to use a custom palette for conversions. In these cases (usually only if you have a color display using a special custom palette and you would like to use the same palette for bitmaps) a custom palette may be used. In the menu, you would select `Image/Convert Into/Custom palette`.

File format for custom palette

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: $16 + (\text{NumColors} * 4)$ 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.

Custom palette sample file

This sample 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 sample.

10.7 BmpCvt.exe: 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.

10.7.1 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 -converttobestpalette -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.

10.7.2 Valid 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	Explanation
-converttobw	Convert to BW.
-converttograyscale	Convert to Gray4.
-converttograyscale16	Convert to Gray16.
-converttograyscale64	Convert to Gray64.
-converttograyscale256	Convert to Gray256.
-converttoto111	Convert to 111.

Command	Explanation
<code>-convertinto222</code>	Convert to 222.
<code>-convertinto233</code>	Convert to 233.
<code>-convertinto323</code>	Convert to 323.
<code>-convertinto332</code>	Convert to 332.
<code>-convertinto8666</code>	Convert to 8666.
<code>-convertintorgb</code>	Convert to RGB.
<code>-convertintobestpalette</code>	Convert to best palette.
<code>-convertintocustompalette<filename></code>	Convert to a custom palette. <code><filename></code> User-specified filename of desired custom palette.
<code>-fliph</code>	Flip image horizontally.
<code>-flipv</code>	Flip image vertically.
<code>-rotate90cw</code>	Rotate image by 90 degrees clockwise.
<code>-rotate90cc</code>	Rotate image by 90 degrees counter-clockwise.
<code>-rotate180</code>	Rotate image by 180 degrees.
<code>-invertindices</code>	Invert indices.
<code>-saveas<filename>[,<type>[,<fmt>[,<noplt>]]]</code>	Save file as filename. <code><filename></code> User-specified file name including the file extension. <code><type></code> Must be an integer from 1 to 6 as follows: 1: "C" with palette (.c file) 2: Windows Bitmap file (.bmp file) 3: "C" stream (.dta file)
<code><fmt></code>	Specifies the bitmap format (only if type == 1): 1: 1 bit per pixel 2: 2 bits per pixel 3: 4 bits per pixel 4: 8 bits per pixel 5: RLE4 compression 6: RLE8 compression 7: High color (565) 8: High color (565), 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)
<code><noplt></code>	Saves the bitmap with or without palette (only if type == 1) 0: Save bitmap with palette (default) 1: Save bitmap without palette
<code>-exit</code>	Terminate PC program automatically.
<code>-help</code>	Display this box.
<code>-?</code>	Display this box.

10.8 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 sample 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 bitmap converter)

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*          www.segger.com
*****
*
* C-file generated by
*
*      Bitmap converter for emWin V3.76.
*      Compiled Feb 10 2004, 09:26:47
*      (C) 1998 - 2004 Segger Microcontroller Systeme GmbH
*
*****
*
* Source file: test
* Dimensions: 200 * 94
* NumColors: 15
*
*****
*/
#include "stdlib.h"
#include "GUI.h"

#ifndef GUI_CONST_STORAGE
#define GUI_CONST_STORAGE const
#endif

/* Palette
The following are the entries of the palette table.
Every entry is a 32-bit value (of which 24 bits are actually used)
the lower 8 bits represent the Red component,
the middle 8 bits represent the Green component,
the highest 8 bits (of the 24 bits used) represent the Blue component
as follows: 0xBBGRR
*/
static GUI_CONST_STORAGE GUI_COLOR ColorsLogo[] = {
    0xFFFFFFFF, 0xFF0000, 0x000000, 0x0F0F0F
    , 0x1C1F23, 0xC3C3C3, 0x020202, 0xFBEBEF
    , 0xFF3B3B, 0x5A5B5E, 0x909294, 0xFFC1C1
    , 0xD0D1D1, 0xFF6868, 0xFF9393
};

static GUI_CONST_STORAGE GUI_LOGPALETTE PalLogo = {
    15, /* number of entries */
    0, /* No transparency */
    &ColorsLogo[0]
};

static GUI_CONST_STORAGE unsigned char acLogo[] = {
    0x00, 0x00, 0xC9, 0x43, ... , 0x00, /* Not all data is shown */
    0x00, 0x0A, 0x32, 0x22, ... , 0x00, /* in this example */
    0x00, 0x92, 0x22, 0x22, ... , 0x00,
    0xA0, 0x22, 0x22, 0x22, ... , 0xA0,
    0xC6, 0x22, 0x23, 0x95, ... , 0x6C,
    .
    .
    .
    0xC6, 0x22, 0x23, 0xA5, ... , 0x6C,
```

```

0x0A, 0x22, 0x22, 0x22, ..., 0x90,
0x07, 0x92, 0x22, 0x22, ..., 0x70,
0x00, 0x7A, 0x32, 0x22, ..., 0x00,
0x00, 0x00, 0xCA, 0x44, ..., 0x00
};

GUI_CONST_STORAGE GUI_BITMAP bmLogo = {
  200, /* XSize */
  94, /* YSize */
  100, /* BytesPerLine */
  4, /* BitsPerPixel */
  acLogo, /* Pointer to picture data (indices) */
  &PalLogo /* 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 3,803 bytes for 18,800 pixels.

Resulting compressed "C" code (generated by bitmap converter)

```

*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*          www.segger.com
*****
*
* C-file generated by
*
*      Bitmap converter for emWin V3.76.
*      Compiled Feb 10 2004, 09:26:47
*      (C) 1998 - 2004 Segger Microcontroller Systeme GmbH
*
*****
*
* Source file: test
* Dimensions: 200 * 94
* NumColors: 15
*
*****
*/
#include "stdlib.h"
#include "GUI.h"

#ifndef GUI_CONST_STORAGE
#define GUI_CONST_STORAGE const
#endif

/* Palette
The following are the entries of the palette table.
Every entry is a 32-bit value (of which 24 bits are actually used)
the lower 8 bits represent the Red component,
the middle 8 bits represent the Green component,
the highest 8 bits (of the 24 bits used) represent the Blue component
as follows: 0xBBGRR
*/
static GUI_CONST_STORAGE GUI_COLOR ColorsLogoCompressed[ ] = {
  0xFFFFFFF, 0xFF0000, 0x000000, 0x0F0F0F
, 0x1C1F23, 0xC3C3C3, 0x020202, 0xFBEFEF
, 0xFF3B3B, 0x5A5B5E, 0x909294, 0xFFC1C1
, 0xD0D1D1, 0xFF6868, 0xFF9393
};

static GUI_CONST_STORAGE GUI_LOGPALETTE PalLogoCompressed = {
  15, /* number of entries */
  0, /* No transparency */
  &ColorsLogoCompressed[0]
};

```

```
static GUI_CONST_STORAGE unsigned char acLogoCompressed[ ] = {  
    /* RLE: 004 Pixels @ 000,000*/ 4, 0x00,  
    /* ABS: 003 Pixels @ 004,000*/ 0, 3, 0xC9, 0x40,  
    /* RLE: 186 Pixels @ 007,000*/ 186, 0x03,  
    /* ABS: 003 Pixels @ 193,000*/ 0, 3, 0x49, 0xC0,  
    /* RLE: 007 Pixels @ 196,000*/ 7, 0x00,  
    .  
    .  
    /* RLE: 006 Pixels @ 198,092*/ 6, 0x00,  
    /* ABS: 004 Pixels @ 004,093*/ 0, 4, 0xCA, 0x44,  
    /* RLE: 184 Pixels @ 008,093*/ 184, 0x03,  
    /* ABS: 004 Pixels @ 192,093*/ 0, 4, 0x44, 0xA5,  
    /* RLE: 004 Pixels @ 196,093*/ 4, 0x00,  
}; /* 3803 for 18800 pixels */  
  
GUI_CONST_STORAGE GUI_BITMAP bmLogoCompressed = {  
    200, /* XSize */  
    94, /* YSize */  
    100, /* BytesPerLine */  
    GUI_COMPRESS_RLE4, /* BitsPerPixel */  
    acLogoCompressed, /* Pointer to picture data (indices) */  
    &PalLogoCompressed /* Pointer to palette */  
    ,GUI_DRAW_RLE4  
};  
  
/* *** End of file *** */
```


Chapter 11

Colors

emWin supports black/white, grayscale (monochrome with different intensities) and color displays. The same user program can be used with any display; only the LCD-configuration needs to be changed. The color management tries to find the closest match for any color that should be displayed.

Logical colors are the colors the application deals with. A logical colors is always defined as an RGB value. This is a 24-bit value containing 8 bits per color as follows: 0xBBGGRR. Therefore, white would be 0xFFFFF, black would be 0x000000, bright red 0xFF.

Physical colors are the colors which can actually be displayed by the display. They are specified in the same 24-bit RGB format as logical colors. At run-time, logical colors are mapped to physical colors.

For displays with few colors (such as monochrome displays or 8/16-color LCDs), emWin converts them by using an optimized version of the "least-square deviation search". It compares the color to display (the logical color) with all the available colors that the LCD can actually show (the physical colors) and uses the one that the LCD-metric considers closest.

11.1 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		0xFFFFFFFF
GUI_LIGHTGRAY		0xD3D3D3
GUI_GRAY		0x808080
GUI_DARKGRAY		0x404040
GUI_BLACK		0x000000
GUI_BROWN		0x2A2AA5

Example

```
/* Set background color to magenta */
GUI_SetBkColor(GUI_MAGENTA);
GUI_Clear();
```

11.2 The color bar test routine

The color bar sample 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 screen shots 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 samples shipped with emWin.

11.3 Fixed palette modes

The following table lists the available fixed palette color modes and the necessary #defines which need to be made in the file `LCDConf.h` in order to select them. Detailed descriptions follow.

LCD_FIXEDPALETTE (Color Mode)	No. available colors	LCD_SWAP_RB	Mask
1	2 (black and white)	x	0x01
2	4 (grayscale)	x	0x03
4	16 (grayscale)	x	0x0F
111	8	0	BGR
111	8	1	RGB
222	64	0	BBGGRR
222	64	1	RRGGBB
233	256	0	BBGGGRRR
233	256	1	RRGGGBBB
323	256	0	BBBGGRRR
323	256	1	RRRGGBBB
332	256	0	BBBGGGRR
332	256	1	RRRGGBBB
44412	4096	0	0000BBBBGGGGRRRR
44412	4096	1	0000RRRRGGGGBBBB
444121	4096	0	BBBBGGGGRRRR0000
44416	4096	0	0BBB0GGGG0RRRR0
44416	4096	1	0RRR0GGGG0BBBB0
555	32768	0	0BBBBBGGGGGRRRRR
555	32768	1	0RRRRRGGGGGBBBBB
556	65536	0	BBBBBGGGGGRRRRR
556	65536	1	RRRRRGGGGGGBBBBBB
565	65536	0	BBBBBGGGGGRRRRR
565	65536	1	RRRRRGGGGGGGBBBBB
655	65536	0	BBBBBBGGGGGRRRRR
655	65536	1	RRRRRRGGGGGGBBBBB
8666	232	x	0xFF
86661	233 (232 + transparency)	x	0xFF

1 mode: 1 bpp (black and white)

Use of this mode is necessary for monochrome displays with 1 bit per pixel.

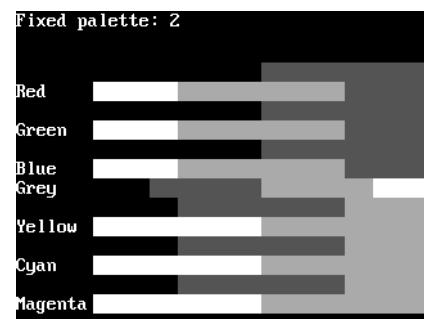
Available colors: 2:



2 mode: 2 bpp (4 grayscales)

Use of this mode is necessary for monochrome displays with 2 bits per pixel.

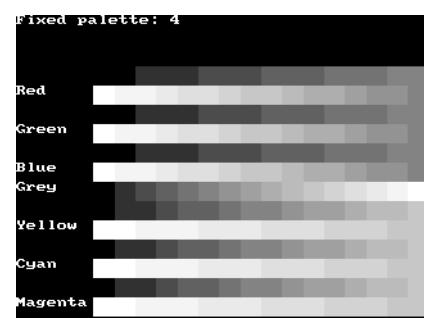
Available colors: $2 \times 2 = 4$:



4 mode: 4 bpp (16 grayscales)

Use of this mode is necessary for monochrome displays with 4 bits per pixel.

Available colors: $2 \times 2 \times 2 \times 2 = 16$:

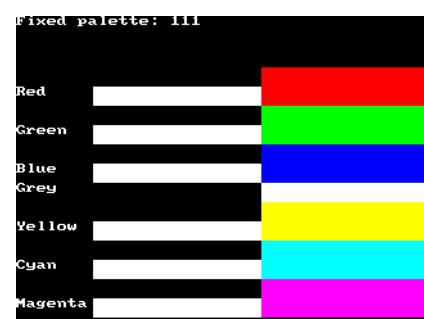


111 mode: 3 bpp (2 levels per color)

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: $2 \times 2 \times 2 = 8$:



111 mode: 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: $2 \times 2 \times 2 = 8$:



222 mode: 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: $4 \times 4 \times 4 = 64$:



222 mode: 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

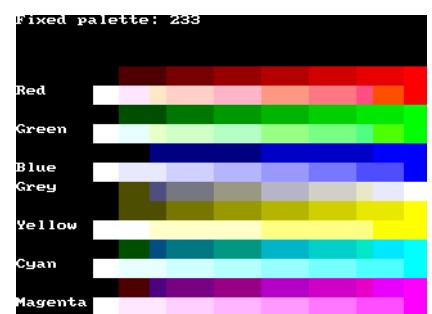
Available colors: $4 \times 4 \times 4 = 64$:



233 mode: 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



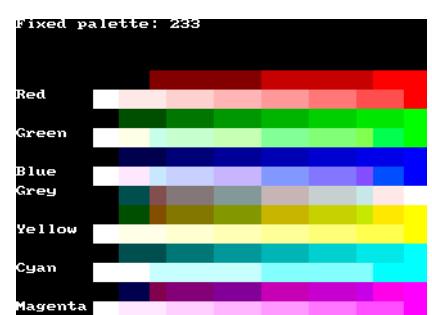
Available colors: $4 \times 8 \times 8 = 256$:



233 mode: 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: RRGGBBBB



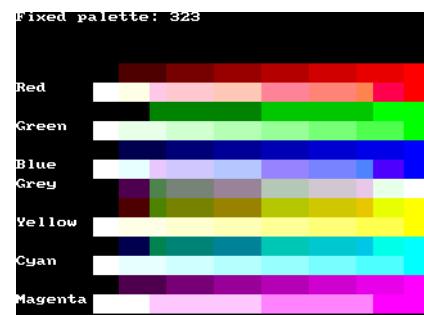
Available colors: $4 \times 8 \times 8 = 256$:



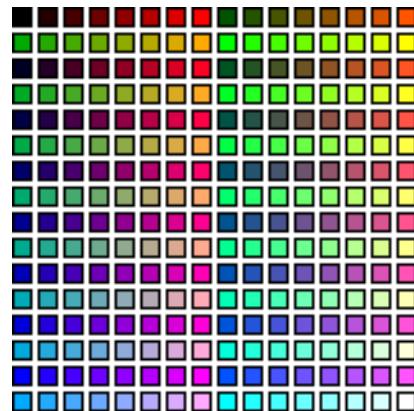
323 mode: 8 bpp

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: $8 \times 4 \times 8 = 256$:



323 mode: 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

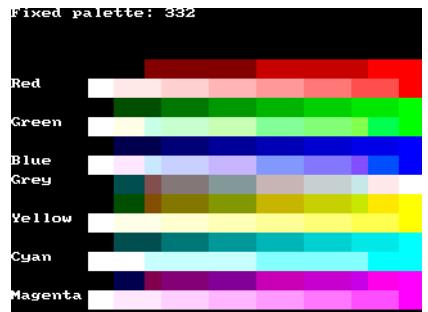
Available colors: $8 \times 4 \times 8 = 256$:



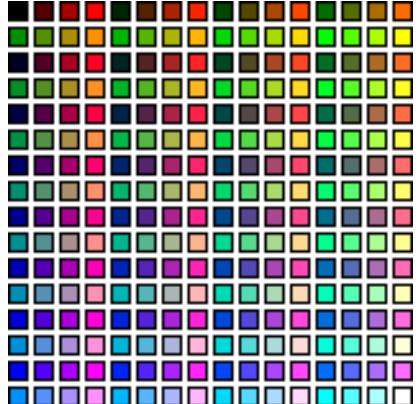
332 mode: 8 bpp

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



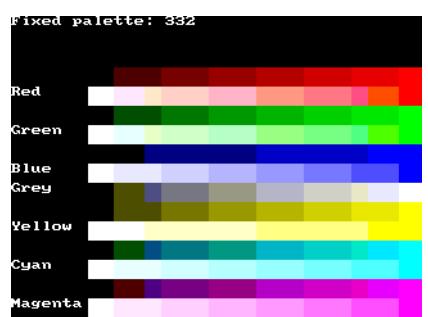
Available colors: $8 \times 8 \times 4 = 256$:



332 mode: 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: $8 \times 8 \times 4 = 256$:



44412 mode:

The red, green and blue components are each 4 bits.

Color mask: 0000BBBBGGGGRRRR

Available colors: $16 \times 16 \times 16 = 4096$.



44416 mode:

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 44412 mode.

Color mask: 0BBBB0GGGG0RRR0

Available colors: $16 \times 16 \times 16 = 4096$.

44412 mode: red and blue swapped

The red, green and blue components are each 4 bits. The available colors are the same as those in 44412 mode.

Available colors: $16 \times 16 \times 16 = 4096$.

Color mask: RRRRGGGGBBBB

44416 mode: 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 44412 mode.

Color mask: 0RRRR0GGGG0BBBB0

Available colors: $16 \times 16 \times 16 = 4096$.

444121 mode:

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 44412 mode.

Color mask: BBBBGGGGRRRR0000

Available colors: $16 \times 16 \times 16 = 4096$.

555 mode: 15 bpp

Use of this mode is necessary for an LCD controller that supports RGB colors with a color-depth of 15 bpp (such as SED1356 or SED13806). The red, green and blue components are each 5 bits.

Color mask: BBBBBGGGGGRRRR

Available colors: $32 \times 32 \times 32 = 32768$.



555 mode: 15 bpp, red and blue swapped

Use of this mode is necessary for an LCD 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: RRRRRGGGGGGBBBB

Available colors: $32 \times 32 \times 32 = 32768$.

565 mode: 16 bpp

Use of this mode is necessary for an LCD 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: BBBBBGGGGGGRRRR

Available colors: $32 \times 64 \times 32 = 65536$.



565 mode: 16 bpp, red and blue swapped

Use of this mode is necessary for an LCD 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: RRRRRGGGGGGGBBBB

Available colors: $32 \times 64 \times 32 = 65536$.

556 mode: 16 bpp

Use of this mode is necessary for an LCD 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: BBBBGGGGRRRR

Available colors: $32 \times 32 \times 64 = 65536$.

556 mode: 16 bpp, red and blue swapped

Use of this mode is necessary for an LCD 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: RRRRGGGGGGBBBBB

Available colors: $32 \times 32 \times 64 = 65536$.

8666 mode: 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 screen shot 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: $6 \times 6 \times 6 + 16 = 232$:



86661 mode: 8bpp, 6 levels per color + 16 grayscales + transparency

This mode is most frequently used with multi layer 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: $6 \times 6 \times 6 + 16 = 232$.



11.4 Custom palette modes

emWin can handle a custom hardware 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 your LCD controller/display combination is able to display, emWin will be able to simulate them in the PC simulation and handle these colors correctly in your target system. Working with a custom palette requires a color depth <= 8 bpp.

In order to define a custom palette, you should do so in the configuration file `LCDConf.h`.

Example

The following example (part of `LCDConf.h`) would define a custom palette with 4 colors, all of which are shades of gray:

```
#define LCD_FIXEDPALETTE 0
#define LCD_PHYSCOLORS 0xffffffff, 0xaaaaaaaa, 0x55555555, 0x00000000
```

11.5 Modifying the color lookup table at run time

The color information at each pixel is stored either in RGB mode (in which the red, green and blue components are kept for each pixel) or in color-index mode (in which a single number called the color index is stored for each pixel). Each color index corresponds to an entry in a lookup table, or color map, that defines a specific set of R, G and B values.

If your LCD controller features a color lookup table (LUT), it is properly initialized by emWin during the initialization phase (`GUI_Init() -> LCD_Init() -> LCD_InitLUT() -> LCD_L0_SetLUTEntry()`). However, it might be desirable (for various reasons) to modify the LUT at run time. Some possible reasons include:

- Color corrections in order to compensate for display problems (non-linearities) or gamma-correction
- Inversion of the display.
- The need to use more colors (at different times) than the hardware can show (at one time).

If you are simply modifying the LUT at run time, the color conversion routines will not be aware of this and will therefore still assume that the LUT is initialized as it was originally.

Using different colors

The default contents of the color table are defined at compile time in the configuration file `GUIConf.h` (`LCD_PHYSCOLORS`). In order to minimize RAM consumption, this data is normally declared `const` and is therefore stored in ROM. In order to be able to modify it, it needs to be stored in RAM. This can be achieved by activation of the configuration switch `LCD_LUT_IN_RAM`. If this is enabled, the API function `GUI_SetLUTColor()` becomes available and can be used to modify the contents of the color table and the LUT at the same time.

A call to `LCD_InitLUT()` will restore the original (default) settings.

11.6 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	Explanation
Basic color functions	
<code>GUI_GetBkColor()</code>	Return the current background color.
<code>GUI_GetBkColorIndex()</code>	Return the index of the current background color.

Routine	Explanation
<code>GUI_GetColor()</code>	Return the current foreground color.
<code>GUI_GetColorIndex()</code>	Return the index of the current foreground color.
<code>GUI_SetBkColor()</code>	Set the current background color.
<code>GUI_SetBkColorIndex()</code>	Set the index of the current background color.
<code>GUI_SetColor()</code>	Set the current foreground color.
<code>GUI_SetColorIndex()</code>	Set the index of the current foreground color.
Index & color conversion	
<code>GUI_CalcColorDist()</code>	Returns the difference between 2 colors
<code>GUI_CalcVisColorError()</code>	Returns the difference to the next available color
<code>GUI_Color2Index()</code>	Convert color into color index.
<code>GUI_Color2VisColor()</code>	Returns the nearest available color
<code>GUI_ColorIsAvailable()</code>	Checks if given color is available
<code>GUI_Index2Color()</code>	Convert color index into color.
Lookup table (LUT) group	
<code>GUI_InitLUT()</code>	Initialize the LUT (hardware).
<code>GUI_SetLUTColor()</code>	Set color of a color index (both hardware and color table).
<code>GUI_SetLUTEEntry()</code>	Write a value into the LUT (hardware).

11.7 Basic color 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	Meaning
Color	Color for background, 24-bit RGB value.

Return value

The selected background color.

GUI_SetBkColorIndex()

Description

Sets the index of the current background color.

Prototype

```
int GUI_SetBkColorIndex(int Index);
```

Parameter	Meaning
Index	Index of the color to be used.

Return value

The selected background color index.

GUI_SetColor()

Description

Sets the current foreground color.

Prototype

```
void GUI_SetColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Color for foreground, 24-bit RGB value.

Return value

The selected foreground color.

GUI_SetColorIndex()

Description

Sets the index of the current foreground color.

Prototype

```
void GUI_SetColorIndex(int Index);
```

Parameter	Meaning
Index	Index of the color to be used.

Return value

The selected foreground color index.

11.8 Index & color conversion

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:

$$\text{Difference} = (\text{Red1} - \text{Red0})^2 + (\text{Green1} - \text{Green0})^2 + (\text{Blue1} - \text{Blue0})^2$$

Prototype

```
U32 GUI_CalcColorDist(GUI_COLOR Color0, GUI_COLOR Color1))
```

Parameter	Meaning
Color0	RGB value of the first color.
Color1	RGB value of the second color.

Return value

The distance as described above.

GUI_CalcVisColorError()

Calculates the distance to the next available color. For details about the calculation please take a look at GUI_CalcColorDist.

Prototype

```
U32 GUI_CalcVisColorError(GUI_COLOR color)
```

Parameter	Meaning
Color	RGB value of the color to be calculated.

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	Meaning
Color	RGB value of the color to be converted.

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	Meaning
Color	RGB value of the color.

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	Meaning
Color	RGB value of the color.

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	Meaning
Index	Index of the color. to be converted

Return value

The RGB color value.

11.9 Lookup table (LUT) group

These functions are optional and will work only if supported by the LCD controller hardware. An LCD controller with LUT hardware is required. Please consult the manual for the LCD controller you are using for more information on LUTs.

GUI_InitLUT()**Description**

Initializes the lookup table of the LCD controller(s).

Prototype

```
void LCD_InitLUT(void);
```

Add. information

The lookup table needs to be enabled (by the `LCD_INITCONTROLLER` macro) for this function to have any effect.

GUI_SetLUTColor()

Description

Modifies a single entry to the color table and the LUT of the LCD controller(s).

Prototype

```
void GUI_SetLUTColor(U8 Pos, GUI_COLOR Color);
```

Parameter	Meaning
<code>Pos</code>	Position within the lookup table. Should be less than the number of colors (e.g. 0-3 for 2 bpp, 0-15 for 4 bpp, 0-255 for 8 bpp).
<code>Color</code>	24-bit RGB value.

Add. information

The closest value possible will be used for the LUT. If a color LUT is to be initialized, all 3 components are used. In monochrome modes the green component is used, but it is still recommended (for better understanding of the program code) to set all 3 colors to the same value (such as 0x555555 or 0xa0a0a0).

The lookup table needs to be enabled (by the `LCD_INITCONTROLLER` macro) for this function to have any effect. This function is always available, but has an effect only if:

- a) The LUT is used
- b) The color table is located in RAM (`LCD_PHYSCOLORS_IN_RAM`)

GUI_SetLUTEntry()

Description

Modifies a single entry to the LUT of the LCD controller(s).

Prototype

```
void GUI_SetLUTEntry(U8 Pos, GUI_COLOR Color);
```

Parameter	Meaning
<code>Pos</code>	Position within the lookup table. Should be less than the number of colors (e.g. 0-3 for 2 bpp, 0-15 for 4 bpp, 0-255 for 8 bpp).
<code>Color</code>	24-bit RGB value.

Add. information

The closest value possible will be used for the LUT. If a color LUT is to be initialized, all 3 components are used. In monochrome modes the green component is used, but it is still recommended (for better understanding of the program code) to set all 3 colors to the same value (such as 0x555555 or 0xa0a0a0).

The lookup table needs to be enabled (by the `LCD_INITCONTROLLER` macro) for this function to have any effect. This function is often used to ensure that the colors actually displayed match the logical colors (linearization).

Example

```
//
// Linearize the palette of a 4-grayscale LCD
//
GUI_SetLUTEntry(0, 0x000000);
GUI_SetLUTEntry(1, 0x777777); // 555555 would be linear
GUI_SetLUTEntry(2, 0bbbbbb); // aaaaaa would be linear
GUI_SetLUTEntry(3, 0xffffff);
```


Chapter 12

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.

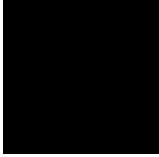
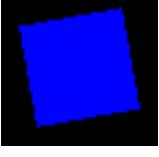
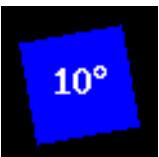
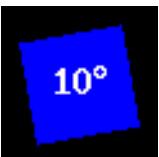
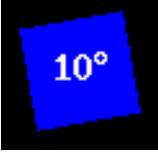
If a memory device is used for such a procedure, however, all drawing operations are executed in memory. The final result is displayed on the screen only when all operations have been carried out, with the advantage of no flickering. This difference can be seen in the example in the following section, 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`.

12.1 Using memory devices: an illustration

The following table shows screen shots 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)		

12.2 Supported color depth (bpp)

Memory devices are available in 3 different color depth:
1bpp, 8 bpp and 16 bpp.

Creating memory devices "compatible" to the display

There are two ways to create memory devices. If they are used 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.

12.3 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.

12.4 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: $XSIZE * YSIZE * 2$

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:

$$\text{Number of required bytes} = (111 * 2 + (111 + 7) / 8) * 33 = 7788 \text{ bytes}$$

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_BITSPIXEL)	Memory usage
1 bpp	1 bpp	1 byte / pixel + 1 byte / 8 pixels: $(\text{XSIZE} + (\text{XSIZE} + 7) / 8) * \text{YSIZE}$
8 bpp	2, 4 and 8 bpp	2 bytes / pixel + 1 byte / 8 pixels: $(\text{XSIZE} * 2 + (\text{XSIZE} + 7) / 8) * \text{YSIZE}$
16 bpp	12 and 16 bpp	2 bytes / pixel + 1 byte / 8 pixels: $(\text{XSIZE} * 2 + (\text{XSIZE} + 7) / 8) * \text{YSIZE}$

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:

$$\text{Number of required bytes} = (200 + (200 + 7) / 8) * 50 = 11250 \text{ bytes}$$

12.5 Performance

Using memory devices typically does no 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, LCDLin driver 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.

12.6 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()`).

12.7 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.

12.8 Multi layer / multi display configurations

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.

12.9 Configuration options

Type	Macro	Default	Explanation
B	GUI_USE_MEMDEV_1BPP_FOR_SCREEN	0	Enables the use of 1bpp memory devices with displays of 1bpp color depth.

12.9.1 GUI_USE_MEMDEV_1BPP_FOR_SCREEN

On systems with a display color depth \leq 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 1
```

12.10 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	Explanation
Basic functions	
GUI_MEMDEV_Clear()	Marks the memory device contents as unchanged.
GUI_MEMDEV_CopyFromLCD()	Copies contents of LCD to memory device.
GUI_MEMDEV_CopyToLCD()	Copies contents of memory device to LCD.
GUI_MEMDEV_CopyToLCDAA()	Copies the contents of memory device antialiased.
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_Delete()	Frees the memory used by the memory 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_ReduceYSize()	Reduces Y-size of memory device.
GUI_MEMDEV_Select()	Selects a memory device as target for drawing operations.
GUI_MEMDEV_SetOrg()	Changes the origin of the memory device on the LCD.
GUI_MEMDEV_Write()	Writes the contents of a memory device into a memory device.
GUI_MEMDEV_WriteAlpha()	Writes the contents of a memory device into a memory device using alpha blending.
GUI_MEMDEV_WriteAlphaAt()	Writes the contents of a memory device into a memory device using the given position and alpha blending.
GUI_MEMDEV_WriteAt()	Writes the contents of a memory device into a memory device to the given position.
GUI_MEMDEV_WriteEx()	Writes the contents of a memory device into a memory device using alpha blending and scaling.

Routine	Explanation
<code>GUI_MEMDEV_WriteExAt()</code>	Writes the contents of a memory device into a memory device to the given position using alpha blending and scaling.
<code>GUI_SelectLCD()</code>	Selects the LCD as target for drawing operations.
	Banding memory device
<code>GUI_MEMDEV_Draw()</code>	Use a memory device for drawing.
	Auto device object functions
<code>GUI_MEMDEV_CreateAuto()</code>	Creates an auto device object.
<code>GUI_MEMDEV_DeleteAuto()</code>	Deletes an auto device object.
<code>GUI_MEMDEV_DrawAuto()</code>	Uses a GUI_AUTODEV object for drawing.
	Measurement device object functions
<code>GUI_MEASDEV_ClearRect()</code>	Clears the measurement rectangle
<code>GUI_MEASDEV_Create()</code>	Creates a measurement device
<code>GUI_MEASDEV_Delete()</code>	Deletes a measurement device
<code>GUI_MEASDEV_GetRect()</code>	Retrieves the measurement result
<code>GUI_MEASDEV_Select()</code>	Selects a measurement device as target for drawing operations

12.11 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	Meaning
<code>hMem</code>	Handle to memory device.

Add. 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_CopyFromLCD()`

Description

Copies the contents of a memory device from LCD data (video memory) to the memory device. In other words: read back the contents of the LCD to the memory device.

Prototype

```
void GUI_MEMDEV_CopyFromLCD(GUI_MEMDEV_Handle hMem);
```

Parameter	Meaning
<code>hMem</code>	Handle to memory device.

`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	Meaning
hMem	Handle to memory device.

Add. information

Do not use this function within a paint callback function called by the window manager, because it deactivates the clipping area of the window manager. The function GUI_MEMDEV_WriteAt should be used instead.

GUI_MEMDEV_CopyToLCDAA()

Description

Copies the contents of a memory device (antialiased) from memory to the LCD.

Prototype

```
void GUI_MEMDEV_CopyToLCDAA(GUI_MEMDEV_Handle MemDev);
```

Parameter	Meaning
hMem	Handle to memory device.

Add. 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);
```

Screen shot of above example

GUI_MEMDEV_CopyToLCDAt()

Description

Copies the contents of a memory device from memory to the LCD at the given position.

Prototype

```
void GUI_MEMDEV_CopyToLCDAt(GUI_MEMDEV_Handle hMem, int x, int y)
```

Parameter	Meaning
hMem	Handle to memory device.
x	Position in X
y	Position in Y

GUI_MEMDEV_Create()

Description

Creates a memory device.

Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_Create(int x0, int y0, int XSize, int YSize)
```

Parameter	Meaning
x0	X-position of memory device.
y0	Y-position of memory device.
xsize	X-size of memory device.
ysize	Y-size of memory device.

Return value

Handle for 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 YSize
                                         int Flags)
```

Parameter	Meaning
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).

Permitted values for parameter Flags	
0 (recommended)	Default: The memory device is created with a transparency flag which ensures that the background 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 for created memory device. If the routine fails the return value is 0.

GUI_MEMDEV_CreateFixed()**Description**

Creates a memory device of fixed size, color depth (bpp) and specified 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	Meaning
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).

Permitted values for parameter Flags	
0 (recommended)	Default: The memory device is created with a transparency flag which ensures that the background 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 and 16bpp 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							
<table border="1"> <tr> <td>GUI_MEMDEV_APILIST_1</td> <td>Create memory device with 1bpp color depth (1 byte per 8 pixels) Use if the specified color conversion requires 1bpp.</td> </tr> <tr> <td>GUI_MEMDEV_APILIST_8</td> <td>Create memory device with 8bpp color depth (1 byte per pixel) Use if the specified color conversion requires 8bpp or less.</td> </tr> <tr> <td>GUI_MEMDEV_APILIST_16</td> <td>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)</td> </tr> </table>		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_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)						

Parameter pColorConvAPI									
This parameter defines the desired color conversion. For more details about the used bits per pixel and the color conversion please refer to the chapter "Colors".									
Permitted values									
<table border="1"> <tr> <td>GUI_COLOR_CONV_1</td> <td>Same color conversion like the fixed palette mode 1. (black/white)</td> </tr> <tr> <td>GUI_COLOR_CONV_2</td> <td>Same color conversion like the fixed palette mode 2. (4 gray scales)</td> </tr> <tr> <td>GUI_COLOR_CONV_4</td> <td>Same color conversion like the fixed palette mode 4. (16 gray scales)</td> </tr> <tr> <td>GUI_COLOR_CONV_8666</td> <td>Same color conversion like the fixed palette mode 8666.</td> </tr> </table>		GUI_COLOR_CONV_1	Same color conversion like the fixed palette mode 1. (black/white)	GUI_COLOR_CONV_2	Same color conversion like the fixed palette mode 2. (4 gray scales)	GUI_COLOR_CONV_4	Same color conversion like the fixed palette mode 4. (16 gray scales)	GUI_COLOR_CONV_8666	Same color conversion like the fixed palette mode 8666.
GUI_COLOR_CONV_1	Same color conversion like the fixed palette mode 1. (black/white)								
GUI_COLOR_CONV_2	Same color conversion like the fixed palette mode 2. (4 gray scales)								
GUI_COLOR_CONV_4	Same color conversion like the fixed palette mode 4. (16 gray scales)								
GUI_COLOR_CONV_8666	Same color conversion like the fixed palette mode 8666.								

Return value

Handle for created memory device. If the routine fails the return value is 0.

Add. 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 sample `MEMDEV_Printing.c` which shows, how to use the function to print something in 1bpp color conversion mode.

Sample

The following sample 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_Delete()

Description

Deletes a memory device.

Prototype

```
void GUI_MEMDEV_Delete(GUI_MEMDEV_Handle MemDev);
```

Parameter	Meaning
hMem	Handle to memory device.

Return value

Handle for deleted memory device.

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	Meaning
hMem	Handle to memory device.

Add. information

The device data is stored from the returned addr. onwards. An application modifying this data has to take extreme caution that it does not overwrite memory outside of this data area. If this data area is used with emWins default memory management, the memory area must remain locked as long as the pointer is in use.

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	Meaning
hMem	Handle to memory device.

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	Meaning
hMem	Handle to memory device.

GUI_MEMDEV_ReduceYSIZE()

Description

Reduces the Y-size of a memory device.

Prototype

```
void GUI_MEMDEV_ReduceYSIZE(GUI_MEMDEV_Handle hMem, int YSize);
```

Parameter	Meaning
hMem	Handle to memory device.
YSize	New Y-size of the memory device.

Add. information

Changing the size of the memory device is more efficient than deleting and then re-creating it.

GUI_MEMDEV_Select()

Description

Activates a memory device (or activates LCD if handle is 0)

Prototype

```
void GUI_MEMDEV_Select(GUI_MEMDEV_Handle hMem)
```

Parameter	Meaning
hMem	Handle to memory device.

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	Meaning
hMem	Handle to memory device.
x0	Horizontal position (of the upper left pixel).
y0	Vertical position (of the upper left pixel).

Add. 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 contents of the given memory device into the currently selected memory device.

Prototype

```
void GUI_MEMDEV_Write(GUI_MEMDEV_Handle hMem);
```

Parameter	Meaning
hMem	Handle to memory device.

GUI_MEMDEV_WriteAlpha()

Description

Writes the contents of the given memory device into the currently selected memory device using alpha blending.

Prototype

```
void GUI_MEMDEV_WriteAlpha(GUI_MEMDEV_Handle hMem, int Alpha);
```

Parameter	Meaning
hMem	Handle to memory device.
Alpha	Alpha blending factor, 0 - 255

Add. 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 memory device.

GUI_MEMDEV_WriteAlphaAt()

Description

Writes the contents of the given memory device into the currently selected memory device at the specified position using alpha blending.

Prototype

```
void GUI_MEMDEV_WriteAlphaAt(GUI_MEMDEV_Handle hMem,
                             int Alpha, int x, int y);
```

Parameter	Meaning
hMem	Handle to memory device.
Alpha	Alpha blending factor, 0 - 255
x	Position in X
y	Position in Y

Add. information

(See GUI_MEMDEV_WriteAlpha)

GUI_MEMDEV_WriteAt()

Description

Writes the contents of the given memory device into the currently selected memory device at the specified position.

Prototype

```
void GUI_MEMDEV_WriteAt(GUI_MEMDEV_Handle hMem, int x, int y);
```

Parameter	Meaning
hMem	Handle to memory device.
x	Position in X
y	Position in Y

GUI_MEMDEV_WriteEx()

Description

Writes the contents of the given memory device into the currently selected memory 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	Meaning
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.

Add. information

A negative scaling factor mirrors the output. Please also refer to function GUI_MEMDEV_WriteExAt().

GUI_MEMDEV_WriteExAt()

Description

Writes the contents of the given memory device into the currently selected memory device at the specified position using alpha blending and scaling.

Prototype

```
void GUI_MEMDEV_WriteEx(GUI_MEMDEV_Handle hMem,
```

```
int xMag, int yMag, int Alpha);
```

Parameter	Meaning
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.

Add. information

A negative scaling factor mirrors the output.

Example

The following sample 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 contents 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 sample



GUI_SelectLCD()

Description

Selects the LCD as target for drawing operations.

Prototype

```
void GUI_SelectLCD(void)
```

Example for using a memory device

This example demonstrates the use of a memory device. Some items are written to a memory device and then copied to the LCD. It is available as `MEMDEV_MemDev.c` in the emWin samples.

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*****
*
*      (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
*
*      Internet: www.segger.com      Support: support@segger.com
*
*****
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
```

```

distributed in any way. We appreciate your understanding and fairness.
-----
File      : MEMDEV_MemDev.c
Purpose   : Simple demo shows the use of memory devices
-----
*/
#include "GUI.h"

/*
*      static variables
*
*****
*/
static GUI_RECT Rect = {0, 130, 100, 180};

/*
*      static code
*
*****
*/
*****_Draw
*/
static void _Draw(int Delay) {
    GUI_SetPenSize(5);
    GUI_SetColor(GUI_RED);
    GUI_DrawLine(Rect.x0 + 3, Rect.y0 + 3, Rect.x1 - 3, Rect.y1 - 3);
    GUI_Delay(Delay);
    GUI_SetColor(GUI_GREEN);
    GUI_DrawLine(Rect.x0 + 3, Rect.y1 - 3, Rect.x1 - 3, Rect.y0 + 3);
    GUI_Delay(Delay);
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_FontComic24B_ASCII);
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_DispStringInRect("Closed", &Rect, GUI_TA_HCENTER | GUI_TA_VCENTER);
    GUI_Delay(Delay);
}

*****_DemoMemDev
*/
static void _DemoMemDev(void) {
    GUI_MEMDEV_Handle hMem;
    int i;
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("MEMDEV_MemDev - Sample", 160, 5);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Shows the advantage of using a\nmemory device", 160, 50);
    GUI_SetFont(&GUI_Font8_1);
    GUI_DispStringHCenterAt("Draws the picture\nwithout a\nmemory device", 50, 90);
    GUI_DispStringHCenterAt("Draws the picture\nusing a\nmemory device", 270, 90);
    /* Create the memory device */
    hMem = GUI_MEMDEV_Create(Rect.x0, Rect.y0, Rect.x1 - Rect.x0, Rect.y1 - Rect.y0);
    /* Routes the drawing operations to the memory device */
    GUI_MEMDEV_Select(hMem);
    _Draw(0);
    /* Routes the drawing operations to the LCD */
    GUI_MEMDEV_Select(0);
    while (1) {
        for (i = 0; i < 3; i++) {
            GUI_Delay(250);
            GUI_ClearRect(LCD_GetXSize() - Rect.x1, Rect.y0, LCD_GetXSize(), Rect.y1);
            GUI_Delay(250);
            GUI_MEMDEV_CopyToLCDAt(hMem, LCD_GetXSize() - Rect.x1, Rect.y0);
        }
        GUI_Delay(500);
        /* Uses no memory device */
        _Draw(400);
        GUI_Delay(400);
        GUI_ClearRect(0, 130, 319, 219);
    }
}

```

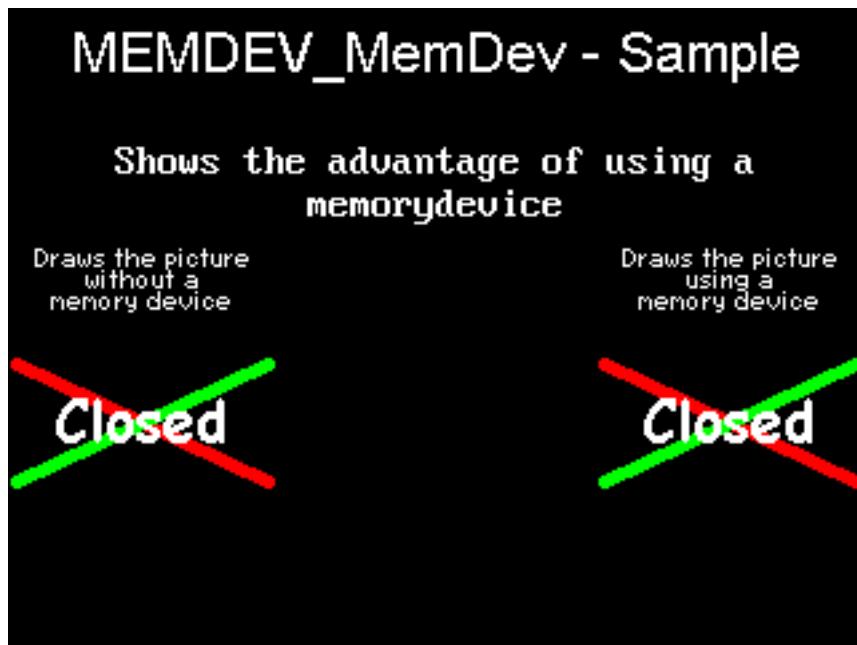
```

    GUI_MEMDEV_Delete(hMem); /* Destroy memory device */
}

*****
*
*      MainTask
*
*      Demonstrates the use of memory devices
*
*****
*/
void MainTask(void) {
    GUI_Init();
    _DemoMemDev();
}

```

Screenshot of the above example:



12.12 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 not be enough memory available to store the complete output area at once, depending on your configuration (see the `GUI_ALLOC_SIZE` configuration macro in Chapter 28: "High-Level 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

Basic drawing function that prevents flickering of the display.

Prototype

```

int GUI_MEMDEV_Draw(GUI_RECT* pRect,
                     GUI_CALLBACK_VOID_P* pfDraw,
                     void* pData,
                     int NumLines,

```

```
    int Flags)
```

Parameter	Meaning
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).

Permitted values for parameter Flags	
0 (recommended)	Default: The memory device is created with a transparency flag which ensures that the background will be drawn correctly.
GUI_MEMDEV_NOTRANS	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.

Add. 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 following example demonstrates the use of a banding memory device. It is available as `MEMDEV_Banding.c`.

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample code
*
*****
```

```
-----
File      : MEMDEV_Banding.c
Purpose   : Example demonstrating the use of banding memory devices
-----
*/
```

```
#include "gui.h"

static const GUI_POINT aPoints[] = {
    {-50, 0},
    {-10, 10},
    {0, 50},
    {10, 10},
    {50, 0},
    {10,-10},
    {0,-50},
    {-10,-10}
};

#define SIZE_OF_ARRAY(Array) (sizeof(Array) / sizeof(Array[0]))

typedef struct {
    int XPos_Poly, YPos_Poly;
    int XPos_Text, YPos_Text;
    GUI_POINT aPointsDest[8];
} tDrawItContext;

*****
*          Drawing routine
*
```

```
*****
*/
static void DrawIt(void * pData) {
    tDrawItContext * pDrawItContext = (tDrawItContext *)pData;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x8);
    GUI_SetTextMode(GUI_TM_TRANS);
    /* draw background */
    GUI_SetColor(GUI_GREEN);
    GUI_FillRect(pDrawItContext->XPos_Text,
                 pDrawItContext->YPos_Text - 25,
                 pDrawItContext->XPos_Text + 100,
                 pDrawItContext->YPos_Text - 5);
    /* draw polygon */
    GUI_SetColor(GUI_BLUE);
    GUI_FillPolygon(pDrawItContext->aPointsDest, SIZE_OF_ARRAY(aPoints), 160, 120);
    /* draw foreground */
    GUI_SetColor(GUI_RED);
    GUI_FillRect(220 - pDrawItContext->XPos_Text,
                 pDrawItContext->YPos_Text + 5,
                 220 - pDrawItContext->XPos_Text + 100,
                 pDrawItContext->YPos_Text + 25);
}
/*****
*
*      Demonstrates the banding memory device
*
*****/
#define USE_BANDING_MEMDEV (1) /* Set to 0 for drawing without banding memory device */

void DemoBandingMemdev(void) {
    int i;
    int XSize = LCD_GET_XSIZE();
    int YSize = LCD_GET_YSIZE();
    tDrawItContext DrawItContext;
    GUI_SetFont(&GUI_Font8x9);
    GUI_SetColor(GUI_WHITE);
    GUI_DispStringHCenterAt("Banding memory device\nwithout flickering",
                           XSize / 2, 40);
    DrawItContext.XPos_Poly = XSize / 2;
    DrawItContext.YPos_Poly = YSize / 2;
    DrawItContext.YPos_Text = YSize / 2 - 4;
    for (i = 0; i < (XSize - 100); i++) {
        float angle = i * 3.1415926 / 60;
        DrawItContext.XPos_Text = i;
        /* Rotate the polygon */
        GUI_RotatePolygon(DrawItContext.aPointsDest,
                           aPoints,
                           SIZE_OF_ARRAY(aPoints), angle);
        #if USE_BANDING_MEMDEV
        {
            GUI_RECT Rect = {0, 70, 320, 170};
            /* Use banding memory device for drawing */
            GUI_MEMDEV_Draw(&Rect, &DrawIt, &DrawItContext, 0, 0);
        }
        #else
        /* Simple drawing without using memory devices */
        DrawIt((void *)&DrawItContext);
        #endif
        #ifdef WIN32
        GUI_Delay(20); /* Use a short delay only in the simulation */
        #endif
    }
}
/*****
*
*      main
*
*****/
void main(void) {
    GUI_Init();
    while(1) {
        DemoBandingMemdev();
    }
}
```

```
}
```

Screen shot of above example



12.13 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	Meaning
<code>pAutoDev</code>	Pointer to a <code>GUI_AUTODEV</code> object.

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	Meaning
<code>pAutoDev</code>	Pointer to a <code>GUI_AUTODEV</code> object.

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	Meaning
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.

Return value

0 if successful, 1 if the routine fails.

Add. 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 informations 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 following example 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 please see Chapter 22: "Antialiasing".

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample code
*
*****
```

```
-----  
File      : MEMDEV_AutoDev.c  
Purpose   : Example demonstrating the use of GUI_AUTODEV-objects  
-----  
*/
```

```
#include "GUI.h"  
#include <math.h>  
#include <stddef.h>

*****
*      defines
*
*****
```

```
/*
 *      static data, scale bitmap
 */
static const GUI_COLOR ColorsScaleR140[] = {
    0x000000, 0x00AA00, 0xFFFFFFF, 0x0000AA,
    0x00FF00, 0xAEAEAE, 0x737373, 0xD3D3D3,
    0xDFDFDF, 0xBBDFBB, 0x6161DF, 0x61DF61,
    0xBBBBDF, 0xC7C7C7, 0x616193
};

static const GUI_LOGPALETTE PalScaleR140 = {
    15, /* number of entries */
    0, /* No transparency */
    &ColorsScaleR140[0]
};

static const unsigned char acScaleR140[] = {
    /* ... The pixel data is not shown in the manual.
     * Please take a look at the sample source file ... */
};

static const GUI_BITMAP bmScaleR140 = {
    200, /* XSize */
    73, /* YSize */
    100, /* BytesPerLine */
    4, /* BitsPerPixel */
    acScaleR140, /* Pointer to picture data (indices) */
    &PalScaleR140 /* Pointer to palette */
};

*****
*      static data, shape of polygon
*
*****
```

```

*/
static const GUI_POINT _aNeedle[] = {
    { MAG * ( 0 ), MAG * ( 0 + 125 ) },
    { MAG * ( -3 ), MAG * ( -15 + 125 ) },
    { MAG * ( -3 ), MAG * ( -65 + 125 ) },
    { MAG * ( 3 ), MAG * ( -65 + 125 ) },
    { MAG * ( 3 ), MAG * ( -15 + 125 ) },
};

/*********************************************
*
*      structure containing information for drawing routine
*/
typedef struct {
    /* Information about what has to be displayed */
    GUI_AUTODEV_INFO AutoDevInfo;
    /* Polygon data */
    GUI_POINT aPoints[7];
    float Angle;
} PARAM;

/*********************************************
*
*      static code
*
*********************************************
*/
/* _GetAngle

This routine returns the value to indicate. In a real
application, this value would somehow be measured.
*/
static float _GetAngle(int tDiff) {
    if (tDiff < 15000) {
        return 225 - 0.006 * tDiff ;
    }
    tDiff -= 15000;
    if (tDiff < 7500) {
        return 225 - 90 + 0.012 * tDiff ;
    }
    tDiff -= 7000;
    return 225;
}

/*********************************************
*
*      _Draw
*/
static void _Draw(void * p) {
    PARAM * pParam = (PARAM *)p;
    /* Fixed background */
    if (pParam->AutoDevInfo.DrawFixed) {
        GUI_ClearRect (60, 80 + bmScaleR140.YSize, 60 + bmScaleR140.XSize - 1, 180);
        GUI_DrawBitmap(&bmScaleR140, 60, 80);
    }
    /* Moving needle */
    GUI_SetColor(GUI_WHITE);
    GUI_AA_FillPolygon(pParam->aPoints, countof(_aNeedle), MAG * 160, MAG * 220);
    /* Fixed foreground */
    if (pParam->AutoDevInfo.DrawFixed) {
        GUI_SetTextMode(GUI_TM_TRANS);
        GUI_SetColor(GUI_RED);
        GUI_SetFont(&GUI_Font24B_ASCII);
        GUI_DispStringHCenterAt("RPM / 1000", 160, 140);
    }
}

/*********************************************
*
*      _DemoScale
*/
static void _DemoScale(void) {
    int Cnt;
    int tDiff, t0;
    PARAM Param;           /* Parameters for drawing routine */
    GUI_AUTODEV AutoDev;   /* Object for banding memory device */
    /* Show message */
}

```

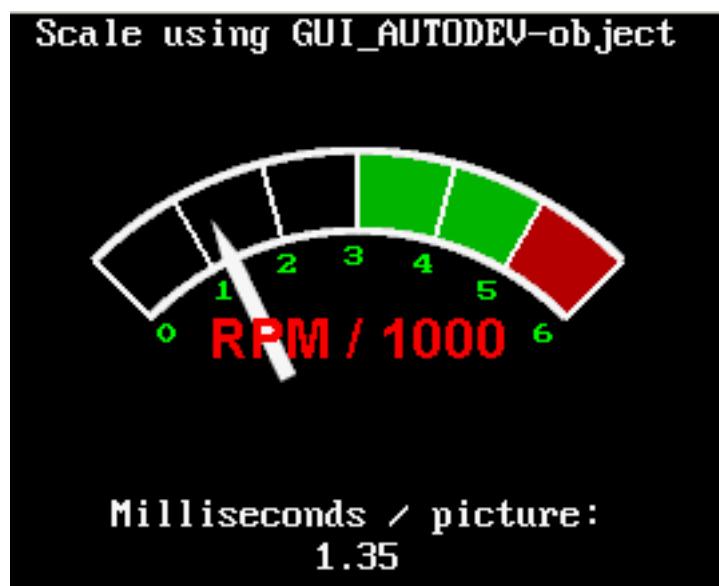
```

GUI_SetBkColor(GUI_BLACK);
GUI_Clear();
GUI_SetColor(GUI_WHITE);
GUI_SetFont(&GUI_Font24_ASCII);
GUI_DispStringHCenterAt("MEMDEV_AutoDev - Sample", 160, 5);
GUI_SetFont(&GUI_Font8x16);
/* Enable high resolution for antialiasing */
GUI_AA_EnableHiRes();
GUI_AA_SetFactor(MAG);
while (1) {
    t0 = GUI_GetTime();
    /* Create GUI_AUTODEV-object */
    GUI_MEMDEV_CreateAuto(&AutoDev);
    /* Show needle for a fixed time */
    for (Cnt = 0; (tDiff = GUI_GetTime() - t0) < 24000; Cnt++) {
        /* Get value to display and calculate polygon for needle */
        Param.Angle = _GetAngle(tDiff)* DEG2RAD;
        GUI_RotatePolygon(Param.aPoints, _aNeedle, countof(_aNeedle), Param.Angle);
        GUI_MEMDEV_DrawAuto(&AutoDev, &Param.AutoDevInfo, &_Draw, &Param);
    }
    /* Display milliseconds / picture */
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Milliseconds / picture:", 160, 200);
    GUI_SetTextAlign(GUI_TA_CENTER);
    GUI_SetTextMode(GUI_TM_NORMAL);
    GUI_DispNextLine();
    GUI_GotoX(160);
    GUI_DispFloatMin((float)tDiff / (float)Cnt, 2);
    /* Delete GUI_AUTODEV-object */
    GUI_MEMDEV_DeleteAuto(&AutoDev);
    GUI_Delay(3000);
    GUI_ClearRect(0, 70, 319, 239);
}
}

*****
*
*      MainTask
*
*      Demonstrates the use of an auto memory device
*
*****
*/
void MainTask(void) {
    GUI_Init();
    while(1) {
        _DemoScale();
    }
}

```

Screen shot of above example



12.14 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_Create()

Description

Creates a measurement device.

Prototype

```
GUI_MEASDEV_Handle GUI_MEASDEV_Create(void);
```

Return value

The handle of the measurement device.

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	Meaning
hMem	Handle to measurement device.

GUI_MEASDEV_Delete()

Description

Deletes a measurement device.

Prototype

```
void GUI_MEASDEV_Delete (GUI_MEASDEV_Handle hMem);
```

Parameter	Meaning
hMem	Handle to measurement device.

GUI_MEASDEV_GetRect()

Description

Retrieves the result of the drawing operations.

Prototype

```
void GUI_MEASDEV_GetRect(GUI_MEASDEV_Handle hMem, GUI_RECT *pRect);
```

Parameter	Meaning
hMem	Handle to measurement device.
pRect	Pointer to GUI_RECT-structure to store result.

GUI_MEASDEV_Select()

Description

Selects a measurement device as target for drawing operations.

Prototype

```
void GUI_MEASDEV_Select (GUI_MEASDEV_Handle hMem);
```

Parameter	Meaning
hMem	Handle to measurement device.

Example

The following sample 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_DisppDec(Rect.x0, 3);
    GUI_DisppString(" Y0:");
    GUI_DisppDec(Rect.y0, 3);
    GUI_DisppString(" X1:");
    GUI_DisppDec(Rect.x1, 3);
    GUI_DisppString(" Y1:");
    GUI_DisppDec(Rect.y1, 3);
}
```

Screenshot of the above example:

X0:010 Y0:020 X1:030 Y1:040

Chapter 13

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.

13.1 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.

13.2 Single task system (superloop)

13.2.1 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.

13.2.2 Superloop example (without emWin)

```
void main (void) {
    HARDWARE_Init();

    /* Init software components */
    XXX_Init();
    YYY_Init();

    /* Superloop: call all software components regularly */
    while (1) {
        /* Exec all components of the software */
        XXX_Exec();
        YYY_Exec();
    }
}
```

13.2.3 Advantages

No real time kernel is used (-> smaller ROM size, just one stack -> less RAM for stacks), no preemption/synchronization problems.

13.2.4 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.

13.2.5 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.

13.2.6 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 regularly */
    while (1) {
        /* Exec all components of the software */
        XXX_Exec();
        YYY_Exec();
        GUI_Exec();          /* Exec emWin for functionality like updating windows */
    }
}
```

13.3 Multitask system: one task calling emWin

13.3.1 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.

13.3.2 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.

13.3.3 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.

13.3.4 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.

13.4 Multitask system: multiple tasks calling emWin

13.4.1 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 behaviour of the system.

Interrupts can, but do not have to be used for real time parts of the software.

13.4.2 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.

13.4.3 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.

13.4.4 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_MAX_TASK 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.

13.4.5 Recommendations

- Call the emWin update functions (i.e. `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.)

13.4.6 Example

This excerpt shows the dedicated emWin update task. It is taken from the example `MT_Multitasking`, which is included in the samples 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 */
    }
}
```

13.5 GUI 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	Explanation
N	GUI_MAXTASK	4	Define the maximum number of tasks from which emWin is called when multitasking support is enabled (see below).
B	GUI_OS	0	Activate to enable multitasking support.
F	GUI_X_SIGNAL_EVENT()	-	Defines a function that signals an event.
F	GUI_X_WAIT_EVENT()	GUI_X_ExecIdle()	Defines a function that waits for an event.

GUI_MAXTASK

Description

Defines the maximum number of tasks from which emWin is called to access the display.

Type

Numerical value

Add. information

This function is only relevant when GUI_OS is activated.

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

Add. 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.

Sample

```
#define GUI_X_SIGNAL_EVENT() GUI_X_SignalEvent()
```

GUI_X_WAIT_EVENT**Description**

Defines a function which waits for an event.

Type

Function replacement

Add. 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.

Sample

```
#define GUI_X_WAIT_EVENT() GUI_X_WaitEvent()
```

13.6 Kernel interface routine 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	Explanation
<code>GUI_X_InitOS()</code>	Initialize the kernel interface module (create a resource semaphore/mutex).
<code>GUI_X_GetTaskId()</code>	Return a unique, 32-bit identifier for the current task/thread.
<code>GUI_X_Lock()</code>	Lock the GUI (block resource semaphore/mutex).

Routine	Explanation
<code>GUI_X_SignalEvent()</code>	Signals an event.
<code>GUI_X_Unlock()</code>	Unlock the GUI (unlock resource semaphore/mutex).
<code>GUI_X_WaitEvent()</code>	Waits for an event.

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_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.

Add. 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_Lock()

Description

Locks the GUI.

Prototype

```
void GUI_X_Lock(void);
```

Add. 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);
```

Add. information

This function is optional, it is used only via the macro `GUI_X_SIGNAL_EVENT()`.

GUI_X_Unlock()

Description

Unlocks the GUI.

Prototype

```
void GUI_X_Unlock(void);
```

Add. 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);
```

Add. information

This function is optional, it is used only via the macro `GUI_X_WAIT_EVENT()`.

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);
}
```

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 inteface for Win32
*
*****


The folling 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 14

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 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`.

14.1 Explanation 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/parent windows, siblings

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).

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.

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/showing 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.

Parent coordinates

Parent coordinates are window coordinates relative to the parent window. The upper left position (the origin) of the window is (0,0).

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/validation

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.

14.2 Callback mechanism, invalidation and rendering

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.

14.2.1 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 your responsibility to manage screen updates!

14.2.2 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	Meaning
<code>pMsg</code>	Pointer to a data structure of type <code>WM_MESSAGE</code> .

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;
    }
}
```

14.2.3 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.

14.2.4 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.

14.2.5 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

14.3 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.

14.3.1 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 WM_MESSAGE

Data type	Element	Meaning
int	MsgId	Type of message (see table below).
WM_HWIN	hWin	Destination window.
WM_HWIN	hWinSrc	Source window.
void*	Data.p	Data pointer.
int	Data.v	Data value.

14.3.2 List of messages

The following messages are defined by emWin.

Message Id (MsgId)	Explanation
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_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_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.
Pointer input device (PID) messages	
WM_MOUSEOVER	Sent to a window if a pointer input device touches the outline of a window.
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 if a pointer input device touches the outline of a window in pressed state.
WM_TOUCH_CHILD	Sent to a parent window if a child window has been touched by the pointer input device.
Notification codes	

Message Id (MsgId)	Explanation
WM_NOTIFICATION_CHILD_DELETED	This notification message will be send from a window to its parent before it will be deleted.
	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.

14.3.3 System-defined messages

These kind of messages are send 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 WM_KEY_INFO

Data type	Element	Meaning
int	Key	The key which has been pressed.
int	PressedCount	> 0 if the key has been pressed, 0 if the key has been released.

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

This message contains no data.

WM_NOTIFY_PARENT

Description

Informs a parent window that something has occurred in one of its child windows. These messages are typically send 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's about the notification codes please refer to the according 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 sample 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_SET_FOCUS

Description

Send 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

Send 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.

14.3.4 Pointer input device (PID) messages

These kind of messages are send 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_MOUSEOVER

Description

A WM_MOUSEOVER message is send to a window if a pointer input device touches the outline of a window. It is send 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.

Elements of GUI_PID_STATE

Data type	Element	Meaning
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.

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 send to the window. This message is send before the touch message is send. 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 WM_PID_STATE_CHANGED_INFO

Data type	Element	Meaning
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

WM_TOUCH

Description

A WM_TOUCH message is send to a window if

- the PID is in pressed state and the pointer is over the visible part of the window
- or the PID just has been released.

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 the structure please refer to the message WM_MOUSEOVER.

Elements of GUI_PID_STATE

Data type	Element	Meaning
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.

WM_TOUCH_CHILD

Description

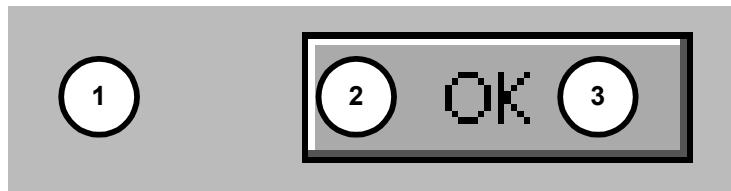
This message is send to the parent window if the outline of a child window has been touched with a pointer input device in pressed or unpressed state.
This message is not sent to disabled windows.

Data

The Data.p pointer of the message points to the touch message send to the child window. For details about the message data please refer to the WM_TOUCH description.

Example

The following sample explains what happens if a pointer input device is dragged over a dialog with a button:



Position	Explanation
1	<p>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</p> <p>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</p>
2	<p>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</p>
3	<p>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</p> <p>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</p>

14.3.5 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. Please do not send system defined notification codes from the user application to a window.

WM_NOTIFICATION_CHILD_DELETED

Description

This notification code is send to the parent window before a child window will be deleted.

Data

The `hWinSrc` element of the message contains the handle of the window to be deleted.

14.3.6 Application-defined messages

The application program can define additional messages for its own usage. In order to ensure that they do not use the same message Id's as those used by emWin, user-defined messages start numbering after `WM_USER`. You would define your own messages as follows:

```
#define MY_MESSAGE_AAA WM_USER+0
#define MY_MESSAGE_BBB WM_USER+1
and so on.
```

14.4 Configuration options

Type	Macro	Default	Explanation
B	WM_SUPPORT_NOTIFY_VIS_CHANGED	0	Enables the WM to send a WM_NOTIFY_VIS_CHANGED message to a window if its visibility has changed.
B	WM_SUPPORT_TRANSPARENCY	1	Enable support for transparent windows. If set to 0 the additional code for transparency support is not included.

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 setting WM_SUPPORT_TRANSPARENCY to 0.

14.5 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	Explanation
Basic functions	
<code>WM_Activate()</code>	Activates the window manager.
<code>WM_AttachWindow()</code>	Attaches a window to a new parent window.
<code>WM_AttachWindowAt()</code>	Attaches a window to a new parent window at the given position.
<code>WM_BroadcastMessage()</code>	Sends a message to all existing windows.
<code>WM_BringToBottom()</code>	Places a window behind its siblings.
<code>WM_BringToTop()</code>	Places a window in front of its siblings.
<code>WM_ClrHasTrans()</code>	Clears the has transparency flag.
<code>WM_CreateWindow()</code>	Creates a window.
<code>WM_CreateWindowAsChild()</code>	Creates a child window.
<code>WM_Deactivate()</code>	Deactivates the window manager.
<code>WM_DefaultProc()</code>	Default routine to handle messages.
<code>WM_DeleteWindow()</code>	Deletes a window.
<code>WM_DetachWindow()</code>	Detaches a window from its parent window.
<code>WM_DisableWindow()</code>	Sets the widget state to disabled.
<code>WM_EnableWindow()</code>	Sets the window state to enabled (default).
<code>WM_Exec()</code>	Redraws invalid windows by executing callbacks (all jobs).
<code>WM_Exec1()</code>	Redraws one invalid window by executing one callback (one job only).
<code>WM_ForEachDesc()</code>	Iterates over all descendants of a window.
<code>WM_GetActiveWindow()</code>	Returns handle of the active window.
<code>WM_GetCallback()</code>	Returns a pointer to the callback function of a window.
<code>WM_GetClientRect()</code>	Returns the size of the active window.
<code>WM_GetClientRectEx()</code>	Returns the size of a window.
<code>WM_GetDesktopWindow()</code>	Returns the window handle of the desktop window
<code>WM_GetDesktopWindowEx()</code>	Returns the window handle of the specified desktop window
<code>WM_GetDialogItem()</code>	Returns the window handle of a dialog box item (widget).
<code>WM_GetFirstChild()</code>	Returns handle of a window's first child window.
<code>WM_GetFocussedWindow()</code>	Returns the handle of the window with the input focus.
<code>WM_GetHasTrans()</code>	Returns current value of the has transparency flag.
<code>WM_GetInvalidRect()</code>	Returns the invalid rectangle of the given window.
<code>WM_GetNextSibling()</code>	Returns the handle of a window's next sibling.
<code>WM_GetOrgX()</code>	Returns the origin in X of the active window.
<code>WM_GetOrgY()</code>	Returns the origin in Y of the active window.
<code>WM_GetParent()</code>	Returns handle of a window's parent window.
<code>WM_GetPrevSibling()</code>	Returns the handle of a window's previous sibling.
<code>WM_GetStayOnTop()</code>	Returns current value of the stay on top flag.
<code>WM_GetUserData()</code>	Retrieves the user data of a window
<code>WM_GetWindowOrgX()</code>	Returns the origin in X of a window.
<code>WM_GetWindowOrgY()</code>	Returns the origin in Y of a window.
<code>WM_GetWindowRect()</code>	Returns the screen coordinates of the active window.
<code>WM_GetWindowRectEx()</code>	Returns the screen coordinates of a window.
<code>WM_GetWindowSizeX()</code>	Returns the horizontal size (width) of a window.
<code>WM_GetWindowSizeY()</code>	Returns the vertical size (height) of a window.
<code>WM_HasCaptured()</code>	Checks if the given window has captured mouse- and touch-screen-input.
<code>WM_HasFocus()</code>	Checks if the given window has the input focus.

Routine	Explanation
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_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 window size.
WM_SelectWindow()	Sets the active window to be used for drawing operations.
WM_SendMessage()	Sends a message to a window.
WM_SendToParent()	Sends the given message to the parent window of the given window.
WM_SetDesktopColor()	Sets desktop window color.
WM_SetDesktopColorEx()	Sets desktop window color of the given desktop.
WM_SetCallback()	Sets the callback routine for a window.
WM_SetCapture()	Starts capturing mouse- and touch screen-input.
WM_SetCreateFlags()	Sets the flags to be used as default when creating new windows
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_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()	Temporarily reduce the clipping area.
WM_SetUserData()	Sets the user data of the given window.
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.
Memory device support (optional)	
WM_DisableMemdev()	Disables usage of memory devices for redrawing.
WM_EnableMemdev()	Enables usage of memory devices for redrawing.
Widget related functions	
WM_GetClientWindow()	Returns the handle of the client window.
WM_GetId()	Returns the ID of a widget.

Routine	Explanation
<code>WM_GetInsideRect()</code>	Returns the size of the active window less the border.
<code>WM_GetInsideRectEx()</code>	Returns the size of a window less the border.
<code>WM_GetScrollPosH()</code>	Returns the scroll position of the windows horizontal scroll bar.
<code>WM_GetScrollPosV()</code>	Returns the scroll position of the windows vertical scroll bar.
<code>WM_GetScrollState()</code>	Gets the state of a scroll bar widget.
<code>WM_SetScrollPosH()</code>	Sets the scroll position of the windows horizontal scroll bar.
<code>WM_SetScrollPosV()</code>	Sets the scroll position of the windows vertical scroll bar.
<code>WM_SetScrollState()</code>	Sets the state of a scroll bar widget.

14.5.1 Using the WM API functions

Many of the WM functions have window handles as parameters. Please observe the following rules when using handles:

- A window handle can be 0. In this case the function returns immediately unless the documentation of the function says otherwise.
- 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 crashes.

14.6 Basic functions

WM_Activate()

Description

Activates the window manager.

Prototype

```
void WM_Activate(void);
```

Add. 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	Meaning
<code>hWin</code>	Window handle.
<code>hWinParent</code>	Window handle of the new parent.

Add. 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	Meaning
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.

Add. 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	Meaning
hWin	Window handle.

Add. information

The window will be placed underneath all other sibling windows, but will remain in front of its parent.

WM_BringToFront()

Description

Places a specified window on top of its siblings.

Prototype

```
void WM_BringToFront(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Add. 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	Meaning
pMsg	Pointer to message to be send.

WM_ClrHasTrans()**Description**

Clears the has transparency flag (sets it to 0).

Prototype

```
void WM_ClrHasTrans(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Add. 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,
                        U8 Style,
                        WM_CALLBACK* cb
                        int NumExtraBytes);
```

Parameter	Meaning
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.

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.
WM_CF_CONST_OUTLINE	This flag is an optimization for transparent windows. It gives the window manager a chance to optimize redraw and invalidation of a transparent window. 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. Can normally be used; in case of doubt do not use it.
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.
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 send to a window. In dependence of other existing windows it can 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 send to an invalid window and the clipping will be done by the drawing routines. The sample folder of emWin contains the sample WM_LateClipping.c to show the effect.
WM_CF_MEMDEV	Automatically use a memory device when redrawing. This will avoid flicker and also improve the output speed in most cases, as clipping is simplified. 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.

Permitted values for parameter <code>Style</code> (OR-combinable)	
<code>WM_CF_MEMDEV_ON_REDRAW</code>	After the window is drawn the first time the WM will automatically use a memory device for redrawing. This flag can be used as a replacement of <code>WM_CF_MEMDEV</code> . It typically accelerates the initial rendering of the window, but maintains the advantage of flicker free updates.
<code>WM_CF_SHOW</code>	Show window after creation.
<code>WM_CF_STAYONTOP</code>	Make sure window stays on top of all siblings created without this flag.

Return value

Handle for created window.

Add. 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, &WinHandler, 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	Meaning
<code>x0</code>	Upper left X-position in window coordinates of the parent window.
<code>y0</code>	Upper left Y-position in window coordinates of the parent window.
<code>width</code>	X-size of window. If 0, X-size of client area of parent window.
<code>height</code>	Y-size of window. If 0, Y-size of client area of parent window.
<code>hWinParent</code>	Handle of parent window.
<code>Style</code>	Window create flags (see <code>WM_CreateWindow()</code>).
<code>cb</code>	Pointer to callback routine, or NULL if no callback used.
<code>NumExtraBytes</code>	Number of extra bytes to be allocated, normally 0.

Return value

Handle for the child window.

Add. 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 top-most.

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);
```

Add. 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	Meaning
pMsg	Pointer to message.

Add. 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	Meaning
hWin	Window handle.

Add. 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	Meaning
hWin	Window handle.

WM_DisableWindow()

Description

Set the specified window to a disabled state. The WM does not pass pointer input device (PID) messages (touch, mouse, joystick, ...) to a disabled window.

Prototype

```
void WM_DisableWindow(WM_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Add. 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	Meaning
hObj	Handle of window.

Add. 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.

Add. 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.

Add. 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	Meaning
<code>hWin</code>	Window handle.
<code>pcb</code>	Pointer to callback function to be called by <code>WM_ForEachDesc</code> .
<code>pData</code>	User data to be passed to the callback function.

Add. 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);
```

Sample

The following sample 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:

```
#include "GUI.h"
```

```

#include "WM.h"

*****
*
*      _cbWin
*/
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);
    }
}

*****
*
*      _cbDoSomething
*/
static void _cbDoSomething(WM_HWIN hWin, void * p) {
    int Value = *(int *)p;
    WM_MoveWindow(hWin, Value, Value);
}

*****
*
*      MainTask
*/
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	Meaning
hWin	Window handle.

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);
```

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	Meaning
pRect	Pointer to a GUI_RECT structure.

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	Meaning
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure.

WM_GetDesktopWindow()**Description**

Returns the handle of the desktop window.

Prototype

```
WM_HWIN WM_GetDesktopWindow(void);
```

Return value

The handle of the desktop window.

Add. 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	Meaning
LayerIndex	Index of layer

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	Meaning
hDialog	Handle of dialog box.
Id	Window Id of the widget.

Return value

The window handle of the widget.

Add. 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	Meaning
hWin	Window handle.

Return value

Handle of the window's first child window; 0 if no child window exists.

Add. 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	Meaning
<code>hWin</code>	Window handle.

Return value

0: no transparency

1: window has transparency

Add. 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	Meaning
<code>hWin</code>	Window handle.
<code>pRect</code>	Pointer to a <code>GUI_RECT</code> -structure for storing the invalid rectangle.

Return value

0 if nothing is invalid, otherwise 1.

WM_GetNextSibling()**Description**

Returns the handle of a specified window's next sibling.

Prototype

```
void WM_GetNextSibling(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

Return value

Handle of the window's next sibling; 0 if none exists.

Add. 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	Meaning
hWin	Window handle.

Return value

Handle of the window's parent window; 0 if none exists.

Add. 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	Meaning
hWin	Window handle.

Return value

Handle of the window's previous sibling; 0 if none exists.

Add. 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	Meaning
hWin	Window handle.

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	Meaning
hWin	Window handle.
pDest	Pointer to buffer.
SizeofBuffer	SIze of buffer.

Return value

Number of bytes retrieved.

Add. 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	Meaning
hWin	Window handle.

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	Meaning
pRect	Pointer to a GUI_RECT structure.

WM_GetWindowRectEx()

Description

Returns the coordinates of a window in desktop coordinates.

Prototype

```
void WM_GetWindowRectEx(WM_HWIN hWin, GUI_RECT* pRect);
```

Parameter	Meaning
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure.

Add. 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	Meaning
hWin	Window handle.

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

Returns 1 if the given window has captured mouse- and touchscreen-input, 0 if not.

Prototype

```
int WM_HasCaptured(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

1 if the given window has captured mouse- and touchscreen-input, 0 if not.

Add. 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	Meaning
hWin	Window handle.

Return value

1 if the given window has the input focus, otherwise 0.

Add. 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	Meaning
hWin	Window handle.

Add. 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	Meaning
pRect	Pointer to a GUI_RECT structure with desktop coordinates.

Add. 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 specified, rectangular area of a window.

Prototype

```
void WM_InvalidateRect(WM_HWIN hWin, GUI_RECT* pRect);
```

Parameter	Meaning
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure with window coordinates of the parent window.

Add. information

Calling this function will tell the WM that the specified area is not updated. The next time `WM_Paint()` is called to redraw the window, the area will be redrawn as well. The coordinates of the GUI_RECT structure have to be in window coordinates.

WM_InvalidateWindow()

Description

Invalidates a specified window.

Prototype

```
void WM_InvalidateWindow(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Add. information

Calling this function tells the WM that the specified window is not updated.

WM_IsCompletelyVisible()

Description

Checks if the given window is completely visible or not.

Prototype

```
char WM_IsCompletelyVisible(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

1 if the given window is completely visible, otherwise 0.

Add. 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	Meaning
hObj	Handle of window.

Return value

1 if the window is enabled, 0 if not.

Add. 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	Meaning
hWin	Window handle.

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	Meaning
hWin	Window handle.

Return value

0: handle is not a valid window handle

1: handle is a valid window handle

Add. 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 send 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	Meaning
hWin	Window handle.

WM_MoveChildTo()

Description

Moves a specified window to a certain position.

Prototype

```
void WM_MoveChildTo(WM_HWIN hWin, int x, int y);
```

Parameter	Meaning
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.

WM_MoveTo()

Description

Moves a specified window to a certain position.

Prototype

```
void WM_MoveTo(WM_HWIN hWin, int x, int y);
```

Parameter	Meaning
hWin	Window handle.
x	New X-position in desktop coordinates.
y	New Y-position in desktop coordinates.

WM_MoveWindow()

Description

Moves a specified window by a certain distance.

Prototype

```
void WM_MoveWindow(WM_HWIN hWin, int dx, int dy);
```

Parameter	Meaning
hWin	Window handle.
dx	Horizontal distance to move.
dy	Vertical distance to move.

WM_NotifyParent()

Description

Sends a WM_NOTIFY_PARENT message to the given window.

Prototype

```
void WM_NotifyParent(WM_HWIN hWin, int Notification);
```

Parameter	Meaning
hWin	Window handle.
Notification	Value to send to the parent window.

Add. information

The `Notification`-parameter will be send in the Data.v element of the message.

WM_Paint()

Description

Draws or redraws a specified window immediately.

Prototype

```
void WM_Paint(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Add. 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	Meaning
hWin	Window handle.

Add. 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);
```

Add. 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	Meaning
hWin	Window handle.
dx	Difference in X.
dy	Difference in Y.

WM_SelectWindow()

Description

Sets the active window to be used for drawing operations.

Prototype

```
WM_HWIN WM_SelectWindow(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

The selected window.

Add. 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	Meaning
hWin	Window handle.
pMsg	Pointer to message.

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	Meaning
hWin	Window handle.
pMsg	Pointer to WM_MESSAGE-structure.

WM_SetDesktopColor()

Description

Sets the color for the desktop window.

Prototype

```
GUI_COLOR WM_SetDesktopColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Color for desktop window, 24-bit RGB value.

Return value

The previously selected desktop window color.

Add. 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	Meaning
Color	Color for desktop window, 24-bit RGB value.
LayerIndex	Index of the layer.

Return value

The previously selected desktop window color.

Add. information

(see WM_SetDesktopColor).

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	Meaning
hWin	Window handle.
cb	Pointer to callback routine.

Return value

Pointer to the previous callback routine.

Add. Information

The given window will be invalidated. This makes sure the window will be redrawn.

WM_SetCapture()**Description**

Routes all mouse- and touchscreen-messages to the given window.

Prototype

```
void WM_SetCapture(WM_HWIN hObj, int AutoRelease);
```

Parameter	Meaning
hWin	Window handle.
AutoRelease	1 if capturing should end when the user releases the touch.

WM_SetCreateFlags()

Description

Sets the flags to be used as default when creating a new window.

Prototype

```
U8 WM_SetCreateFlags (U8 Flags)
```

Parameter	Meaning
Flags	Window create flags (see WM_CreateWindow()).

Return value

Former value of this parameter.

Add. 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.

Example

```
WM_SetCreateFlags(WM_CF_MEMDEV); /* Auto. use memory devices on all windows */
```

WM_SetFocus

Description

Sets the input focus to a specified window.

Prototype

```
void WM_SetFocus(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

0 if window accepted focus; value other than 0 if it could not.

Add. information

The window receives a WM_SETFOCUS message which gives it the input focus. If for some reason the window could not accept the focus, nothing happens.

WM_SetHasTrans()

Description

Sets the has transparency flag (sets it to 1).

Prototype

```
void WM_SetHasTrans(WM_HWIN hWin);
```

Parameter	Meaning
HWin	Window handle.

Add. 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_SetId()

Description

This function sends a WM_SET_ID message to the given window.

Prototype

```
void WM_SetId(WM_HWIN hObj, int Id);
```

Parameter	Meaning
hObj	Window handle.
Id	Id to be send to the window.

Add. information

This function can be used to change the Id of a widget. It works with any `wmWin` widget. When using this function with a application defined window, the callback function of the window should handle the message. Otherwise it will be ignored.

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	Meaning
pf	Pointer to a function of type <code>WM_tfPollPID</code> .

Add. 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	Meaning
hWin	Window handle.
XSize	New size in X.
YSize	New size in Y.

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	Meaning
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.

WM_SetXSize()

Description

Sets the new X-size of a window.

Prototype

```
void WM_SetXSize(WM_HWIN hWin, int Xsize);
```

Parameter	Meaning
hWin	Window handle.
Xsize	New size in X.

WM_SetYSize()

Description

Sets the new Y-size of a window.

Prototype

```
void WM_SetYSize(WM_HWIN hWin, int Ysize);
```

Parameter	Meaning
hWin	Window handle.
Ysize	New size in Y.

WM_SetStayOnTop()

Description

Sets the stay on top flag.

Prototype

```
void WM_SetStayOnTop(WM_HWIN hWin, int OnOff);
```

Parameter	Meaning
hWin	Window handle.
OnOff	(see table below)

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	Meaning
hWin	Window handle.
State	Combination of the flags WM_CF_HASTRANS and WM_CF_CONST_OUTLINE.

Add. information

For details about the flag WM_CF_CONST_OUTLINE please refer to the function WM_CreateWindow().

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	Meaning
pRect	Pointer to a GUI_RECT structure defining the clipping region in desktop coordinates.

Return value

Pointer to the previous clip rectangle.

Add. 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; i.e. 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_DispatchMin(pThis->v); GUI_DispatchChar('%');

/* 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_DispatchMin(pThis->v); GUI_DispatchChar('%');
```

Screen shot of progress bar



WM_SetUserData()

Description

Sets the extra data of a window. Memory for extra data is reserved with the parameter ExtraBytes when creating a window.

Prototype

```
int WM_SetUserData(WM_HWIN hWin, void* pDest, int SizeofBuffer);
```

Parameter	Meaning
hWin	Window handle.
pDest	Pointer to buffer.
SizeofBuffer	Size of buffer.

Return value

Number of bytes reserved when creating the window.

Add. 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	Meaning
hWin	Window handle.

Add. 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	Meaning
hWin	Window handle.

Add. 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	Meaning
hWin	Window handle.

Add. information

The function only draws the invalid window regions.

WM_ValidateRect()

Description

Validates a specified, rectangular area of a window.

Prototype

```
void WM_ValidateRect (WM_HWIN hWin, GUI_RECT* pRect);
```

Parameter	Meaning
hWin	Window handle.
pRect	Pointer to a GUI_RECT structure with window coordinates of the parent window.

Add. 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	Meaning
hWin	Window handle.

Add. 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.

14.7 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 Chapter 12: "Memory Devices".

WM_DisableMemdev()

Description

Disables the use of memory devices for redrawing a window.

Prototype

```
void WM_DisableMemdev (WM_HWIN hWin)
```

Parameter	Meaning
hWin	Window handle.

WM_EnableMemdev()

Description

Enables the use of memory devices for redrawing a window.

Prototype

```
void WM_EnableMemdev (WM_HWIN hWin)
```

Parameter	Meaning
hWin	Window handle.

14.8 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	Meaning
hWin	Handle of widget.

Return value

Handle of the client window.

Add. 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	Meaning
hObj	Handle of widget.

Return value

The ID of the widget specified at creation.

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	Meaning
pRect	Pointer to a GUI_RECT structure.

WM_GetInsideRectEx()

Description

Returns the coordinates of a window less the border size. For details please take a look at WM_GetInsideRect described above.

Prototype

```
void WM_GetInsideRectEx(WM_HWIN hObj, GUI_RECT* pRect);
```

Parameter	Meaning
hObj	Handle of widget.
pRect	Pointer to a GUI_RECT structure.

WM_GetScrollPosH()

Description

If a horizontal scroll bar is attached to the window, the function returns the scroll position of it.

Prototype

```
int WM_GetScrollPosH(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

Scroll position of the horizontal scroll bar attached to this window.

Add. information

If no horizontal scroll bar is attached to the window, the function returns 0. For more information, please refer to the scroll bar widget section.

WM_GetScrollPosV()

Description

If a vertical scroll bar is attached to the window, the function returns the scroll position of it.

Prototype

```
int WM_GetScrollPosV(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

Scroll position of the vertical scroll bar attached to this window.

Add. information

If no vertical scroll bar is attached to the window, the function returns 0. For more information, please refer to the scroll bar widget section.

WM_GetScrollState()

Description

Fills a data structure with information on the current state of a specified scroll bar widget window.

Prototype

```
void WM_GetScrollState(WM_HWIN hObj, WM_SCROLL_STATE* pScrollState);
```

Parameter	Meaning
hObj	Handle of scroll bar widget.
pScrollState	Pointer to a data structure of type WM_SCROLL_STATE.

Add. 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.

For more information, please refer to the scroll bar widget section.

Elements of WM_SCROLL_STATE

Data type	Element	Meaning
int	NumItems	Number of items.
int	v	Current value.
int	PageSize	Number of items visible on one page.

WM_SetScrollPosH()

Description

If a horizontal scroll bar is attached to the window, the function sets the scroll position of it.

Prototype

```
void WM_SetScrollPosH(WM_HWIN hWin, unsigned ScrollPos);
```

Parameter	Meaning
hWin	Window handle.
ScrollPos	New scroll position of the scroll bar.

Add. information

If no horizontal scroll bar is attached to the window, the function returns.
For more information, please refer to the scroll bar widget section.

WM_SetScrollPosV()

Description

If a vertical scroll bar is attached to the window, the function sets the scroll position of it.

Prototype

```
void WM_SetScrollPosV(WM_HWIN hWin, unsigned ScrollPos);
```

Parameter	Meaning
hWin	Window handle.
ScrollPos	New scroll position of the scroll bar.

Add. information

If no vertical scroll bar is attached to the window, the function returns.
For more information, please refer to the scroll bar widget section.

WM_SetScrollState()

Description

Sets the state of a specified scroll bar widget.

Prototype

```
void WM_SetScrollState(WM_HWIN hObj, const WM_SCROLL_STATE* pState);
```

Parameter	Meaning
hObj	Handle of scroll bar widget.

14.9 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 samples shipped with emWin:

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample 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 15

Window Objects (Widgets)

Widgets are windows with object-typeproperties; they are called controls in the windows world 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 need to be created, have properties which may be changed at any time during their existence and are then typically deleted when they are no longer needed. Just like a window, a widget is referenced by a handle which is returned by its 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. Widgets are not required when writing an application or a user interface, but they can make programming much easier.

15.1 Some basics

15.1.1 Available widgets

The following emWin widgets are currently available:

Name	Screenshot	Explanation
BUTTON		Button which can be pressed. Text or bitmaps may be displayed on a button.
CHECKBOX		Check box which may be checked or unchecked.
DROPODOWN		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.
LISTBOX		Listbox which highlights items as they are selected by the user.
LISTVIEW		Listview widgets are used to creates tables.
MENU		Menu widgets are used to create horizontal and vertical menus.
MULTIEDIT		Multiedit widgets are used to edit multiple lines of text.

Name	Screenshot	Explanation
MULTIPAGE		Multipage widgets are used to create dialogs with multiple pages.
PROGBAR		Progress bar used for visualization.
RADIOBUTTON		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.
TEXT		Static text controls typically used in dialogs.

15.1.2 Understanding the redrawing mechanism

A widget draws itself according to its properties. This is done when `WM_Exec()` is called. If you do not call `WM_Exec()` from within your program, `WM_Paint()` must be called for the widget. 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). It is then not necessary to manually call `WM_Paint()`; however, it is still legal to do so and may also make sense if you want to ensure that the widget is redrawn immediately.

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).

15.1.3 How to use widgets

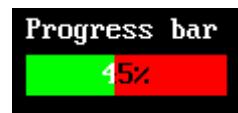
Suppose we would like to display a progress bar. All that is needed is the following code:

```
PROGBAR_Handle hProgBar;
GUI_DisplStringAt("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 if `WM_Exec()` is called at a later point or 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 one of 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.

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

There is no function like WM_GetWidgetType() to determine the type of a widget. The type can be determined only by comparing the callback function of a specific widget with the public callback functions of the widget API. This works fine if the callback function is not overwritten. The following shows a short sample how to determine the type of a widget. In case of overwritten callback functions the method should be adapted:

```
WM_CALLBACK * pCb = WM_GetCallback(hWidget);
if      (pCb == BUTTON_Callback) {
    /* Widget is a button */
} else if (pCb == DROPODOWN_Callback) {
    /* Widget is a dropdown */
} else if (pCb == LISTBOX_Callback) {
    /* Widget is a listbox */
} else if (...) {
    ...
}
```

15.2 General widget API

15.2.1 API reference: 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	Explanation
<code>WM_DeleteWindow()</code>	Delete a window.
<code>WM_DisableMemdev()</code>	Disable usage of memory devices for redrawing.

Routine	Explanation
<code>WM_EnableMemdev()</code>	Enable usage of memory devices for redrawing.
<code>WM_InvalidateWindow()</code>	Invalidate a window.
<code>WM_Paint()</code>	Draw or redraw a window immediately.

For a complete list of WM-related functions, please refer to Chapter 14: "The Window Manager".

15.2.2 API reference: routines common to all widgets

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	Explanation
<code><WIDGET>_Callback()</code>	Default callback function.
<code><WIDGET>_CreateIndirect()</code>	Used for automatic creation in dialog boxes.
<code>WIDGET_GetDefaultEffect()</code>	Returns the default effect used for new widgets.
<code>WIDGET_SetDefaultEffect()</code>	Sets the default effect used for new widgets.
<code>WIDGET_SetDefaultEffect_None()</code>	Sets the default effect used for new widgets to 'None'.
<code>WIDGET_SetDefaultEffect_3D()</code>	Sets the default effect used for new widgets to '3D'.
<code>WIDGET_SetDefaultEffect_Simple()</code>	Sets the default effect used for new widgets to 'Simple'.
<code>WIDGET_SetEffect()</code>	Sets the effect used for a given widget.

`<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	Meaning
<code>pMsg</code>	Pointer to a data structure of type WM_MESSAGE.

Add. 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 please refer to the beginning of the chapter 'The Window Manager'.

`<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	Meaning
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.

Add. 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. Please see Chapter 16: "Dialogs" 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)
} 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 the chapter.

WIDGET_GetDefaultEffect()

Description

Returns the default effect used for new widgets.

Prototype

```
const WIDGET_EFFECT * WIDGET_GetDefaultEffect(void);
```

Return value

The result of the function is a pointer to a `WIDGET_EFFECT` structure.

Add. information

For more information please refer to the function `WIDGET_SetDefaultEffect()`.

WIDGET_SetDefaultEffect()

Description

Sets the default effect used for new widgets.

Prototype

```
const WIDGET_EFFECT * WIDGET_SetDefaultEffect(const WIDGET_EFFECT* pEffect);
```

Parameter	Meaning
pEffect	Pointer to a WIDGET_EFFECT structure. See table below.

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.

Add. information

The following table shows the appearance of some widgets in dependence of the used effect:

'3D'	'None'	'Simple'

WIDGET_SetDefaultEffect_3D()

Description

Sets the default effect used for new widgets to '3D'.

Prototype

```
void WIDGET_SetDefaultEffect_3D(void);
```

Add. information

For more information please refer to the function [WIDGET_SetDefaultEffect\(\)](#).

WIDGET_SetDefaultEffect_None()

Description

Sets the default effect used for new widgets to 'None'.

Prototype

```
void WIDGET_SetDefaultEffect_None(void);
```

Add. information

For more information please refer to the function `WIDGET_SetDefaultEffect()`.

WIDGET_SetDefaultEffect_Simple()**Description**

Sets the default effect used for new widgets to 'Simple'.

Prototype

```
void WIDGET_SetDefaultEffect_Simple(void);
```

Add. information

For more information please refer to the function `WIDGET_SetDefaultEffect()`.

WIDGET_SetEffect()**Description**

Sets the effect for the given widget.

Prototype

```
void WIDGET_SetEffect(WM_HWIN hObj, const WIDGET_EFFECT* pEffect);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>pEffect</code>	Pointer to a <code>WIDGET_EFFECT</code> structure. For details please refer to the function <code>WIDGET_SetDefaultEffect()</code>

15.2.3 User drawn widgets

Some widgets supports owner drawing e.g. 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 a application-defined owner draw function should be defined as follows:

Prototype

```
int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo)
```

Parameter	Meaning
<code>pDrawItemInfo</code>	Pointer to a <code>WIDGET_ITEM_DRAW_INFO</code> structure.

Elements of WIDGET_ITEM_DRAW_INFO

Data type	Element	Meaning
WM_HWIN	hWin	Handle to the listbox.
int	Cmd	(see table below)
int	ItemIndex	Zero based index of LISTBOX entry.
int	x0	X position in window coordinates to be used to draw the item.
int	y0	Y position in window coordinates to be used to draw the item.

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.

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 (e.g. 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 is 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 in a future version of the software and add. command is introduced.

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.

15.3 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`.

15.3.1 Configuration options

Type	Macro	Default	Explanation
N	<code>BUTTON_3D_MOVE_X</code>	1	Number of pixels that text/bitmap moves in horizontal direction in pressed state.
N	<code>BUTTON_3D_MOVE_Y</code>	1	Number of pixels that text/bitmap moves in vertical direction in pressed state.
N	<code>BUTTON_ALIGN_DEFAULT</code>	<code>GUI_TA_HCENTER</code> <code>GUI_TA_VCENTER</code>	Alignment used to display the button text.
N	<code>BUTTON_BKCOLOR0_DEFAULT</code>	0xAAAAAA	Background color, unpressed state.
N	<code>BUTTON_BKCOLOR1_DEFAULT</code>	<code>GUI_WHITE</code>	Background color, pressed state.
N	<code>BUTTON_FOCUSCOLOR_DEFAULT</code>	<code>GUI_BLACK</code>	Default color for rendering the focus rectangle.
S	<code>BUTTON_FONT_DEFAULT</code>	<code>&GUI_Font13_1</code>	Font used for button text.
B	<code>BUTTON.REACT_ON_LEVEL</code>	0	See description below.
N	<code>BUTTON_TEXTCOLOR0_DEFAULT</code>	<code>GUI_BLACK</code>	Text color, unpressed state.
N	<code>BUTTON_TEXTCOLOR1_DEFAULT</code>	<code>GUI_BLACK</code>	Text color, pressed state.

BUTTON.REACT_ON_LEVEL

A button per default reacts on each touch message. For example if touching a dialog with a pointer input device (PID) not exactly on the button and then move the PID in pressed state over the button, the button changes from unpressed to pressed state. This behavior can be useful if using a touch panel.

If a button should only react on level changes, `BUTTON.REACT_ON_LEVEL` should be set to 1. Then a button changes the state only if the PID is pressed and released on the button. If then moving a PID in pressed state over the button it does not react. This behavior can be useful if dialogs should react on `WM_NOTIFICATION_CLICKED`.

Example (`BUTTON.REACT_ON_LEVEL = 0`): One dialog (dialog 2) is shown over an other dialog (dialog 1). The close button of dialog 2 is on the same position as a button of dialog 1. Now the close button of dialog 2 is pressed, which removes dialog 2. The PID now is in pressed state. If now moving the button before releasing it the button of dialog 1 would change from unpressed to pressed state.

This unwanted behavior can be avoided by setting `BUTTON.REACT_ON_LEVEL` to 1.

BUTTON_BKCOLOR0_DEFAULT, BUTTON_BKCOLOR1_DEFAULT

The default for the button 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 to be the same in both its pressed and unpressed states, change `BUTTON_BKCOLOR1_DEFAULT` to `BUTTON_BKCOLOR0_DEFAULT`.

15.3.2 Notification codes

The following events are sent from a button widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off of the button without releasing.

15.3.3 Keyboard reaction

The 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.

15.3.4 BUTTON API

The table below lists the available emWin BUTTON-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
BUTTON_Create()	Create the button. (Obsolete)
BUTTON_CreateAsChild()	Create the button as a child window. (Obsolete)
BUTTON_CreateEx()	Creates the button.
BUTTON_CreateIndirect()	Create the button from resource table entry.
BUTTON_GetBitmap()	Returns a pointer to the BUTTON bitmap.
BUTTON_GetBkColor()	Returns the background color of the BUTTON
BUTTON_GetDefaultBkColor()	Returns the default background color for new BUTTON widgets.
BUTTON_GetDefaultFont()	Returns the default font for new BUTTON widgets.
BUTTON_GetDefaultTextAlign()	Returns the default text alignment for new BUTTON widgets.
BUTTON_GetDefaultTextColor()	Returns the default text color for new BUTTON widgets.
BUTTON_GetFont()	Returns a pointer to the font of the BUTTON
BUTTON_GetText()	Retrieves the text of a specified BUTTON.
BUTTON_GetTextColor()	Returns the text color of the specified BUTTON.
BUTTON_IsPressed()	Returns if a button is pressed or not.
BUTTON_SetBitmap()	Set the bitmap used when displaying the BUTTON.
BUTTON_SetBitmapEx()	Set the bitmap used when displaying the BUTTON.
BUTTON_SetBkColor()	Set the background color of the button.
BUTTON_SetBMP()	Set the bitmap used when displaying the BUTTON.
BUTTON_SetBMPE()	Set the bitmap used when displaying the BUTTON.
BUTTON_SetDefaultBkColor()	Sets the default background color for new BUTTON widgets.
BUTTON_SetDefaultFont()	Sets the default font for new BUTTON widgets.
BUTTON_SetDefaultTextAlign()	Sets the default text alignment for new BUTTON widgets.
BUTTON_SetDefaultTextColor()	Sets the default text color for new BUTTON widgets.
BUTTON_SetFocussable()	Sets the ability to receive the input focus.
BUTTON_SetFont()	Select the font for the text.
BUTTON_SetPressed()	Sets the state of the button to pressed or unpressed.

Routine	Explanation
<code>BUTTON_SetStreamedBitmap()</code>	Set the bitmap used when displaying the BUTTON.
<code>BUTTON_SetStreamedBitmapEx()</code>	Set the bitmap used when displaying the BUTTON.
<code>BUTTON_SetText()</code>	Set the text.
<code>BUTTON_SetTextAlign()</code>	Sets the alignment of the BUTTON text.
<code>BUTTON_SetTextColor()</code>	Set the color(s) for the text.

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	Meaning
<code>x0</code>	Leftmost pixel of the button (in parent coordinates).
<code>y0</code>	Topmost pixel of the button (in parent coordinates).
<code>xsize</code>	Horizontal size of the button (in pixels).
<code>ysize</code>	Vertical size of the button (in pixels).
<code>Id</code>	ID to be returned when button is pressed.
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).

Return value

Handle for the created BUTTON widget; 0 if the routine 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	Meaning
<code>x0</code>	X-position of the button relative to the parent window.
<code>y0</code>	Y-position of the button relative to the parent window.
<code>xsize</code>	Horizontal size of the button (in pixels).
<code>ysize</code>	Vertical size of the button (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new button window will be a child of the desktop (top-level window).
<code>Id</code>	ID to be returned when button is pressed.
<code>Flags</code>	Window create flags (see <code>BUTTON_Create()</code>).

Return value

Handle for the created BUTTON widget; 0 if the routine 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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	Not used yet, reserved for future use.
Id	Window ID of the widget.

Return value

Handle for the created BUTTON widget; 0 if the routine fails.

BUTTON_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements Flags and Para of the resource passed as parameter are not used.

BUTTON_GetBitmap()

Description

Returns a pointer to the optional button bitmap.

Prototype

```
const GUI_BITMAP * BUTTON_GetBitmap(BUTTON_Handle hObj,
                                    unsigned int Index);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Index of desired bitmap (see table below).

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.

Add. information

For details about how to set a button bitmap please refer to the functions `BUTTON_SetBitmap()` and `BUTTON_SetBitmapEx()`.

BUTTON_GetBkColor()

Description

Returns the background color of the given BUTTON widget.

Prototype

```
GUI_COLOR BUTTON_GetBkColor(BUTTON_Handle hObj, unsigned int Index);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Index for color (see table below).

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 used by new BUTTON widgets.

Prototype

```
GUI_COLOR BUTTON_GetDefaultBkColor(unsigned Index);
```

Parameter	Meaning
Index	Index for color (see table below).

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 used by new BUTTON widgets.

BUTTON_GetDefaultFont()

Description

Returns a pointer to the GUI_FONT structure used used to display the BUTTON text of new BUTTON widgets.

Prototype

```
const GUI_FONT * BUTTON_GetDefaultFont(void);
```

Return value

Pointer to the GUI_FONT structure used used to display the text of new BUTTON widgets.

BUTTON_GetDefaultTextAlign()

Description

Returns the default text alignment used to display the text of new BUTTON widgets.

Prototype

```
int BUTTON_GetDefaultTextAlign(void);
```

Return value

Default text alignment used to display the text of new BUTTON widgets.

BUTTON_GetDefaultTextColor()

Description

Returns the default text color used to display the text of new BUTTON widgets.

Prototype

```
GUI_COLOR BUTTON_GetDefaultTextColor(unsigned Index);
```

Parameter	Meaning
Index	Index for color (see table below).

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 new 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	Meaning
hObj	Handle of widget.

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	Meaning
hObj	Handle of widget.
pBuffer	Pointer to buffer.
MaxLen	Size of buffer.

BUTTON_GetTextColor()**Description**

Returns the text color of the given BUTTON widget.

Prototype

```
GUI_COLOR BUTTON_GetTextColor(BUTTON_Handle hObj, unsigned int Index);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Index for color (see table below).

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_IsPressed()**Description**

Returns if the BUTTON is pressed or not.

Prototype

```
unsigned BUTTON_IsPressed(BUTTON_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

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.

Prototype

```
void BUTTON_SetBitmap(BUTTON_Handle hObj, unsigned int Index,
```

```
const GUI_BITMAP* pBitmap);
```

Parameter	Meaning
hObj	Handle of button.
Index	Index for bitmap (see table below).
pBitmap	Pointer to the bitmap structure.

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.

Add. information

If only a bitmap for the unpressed state is set the button will show it also when it is pressed or disabled.

BUTTON_SetBitmapEx()

Description

Sets the bitmap(s) to be used when displaying a specified button.

Prototype

```
void BUTTON_SetBitmapEx(BUTTON_Handle hObj, unsigned int Index,
                        const GUI_BITMAP* pBitmap, int x, int y);
```

Parameter	Meaning
hObj	Handle of button.
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.

Add. information

If only a bitmap for the unpressed state is set the button will show it also when it is pressed or disabled.

BUTTON_SetBkColor()

Description

Sets the button background color.

Prototype

```
void BUTTON_SetBkColor(BUTTON_Handle hObj, unsigned int Index,
                       GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of button.
Index	Index for color (see table below).
Color	Background color to be set.

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(s) to be used when displaying a specified button.

Prototype

```
void BUTTON_SetBMP(BUTTON_Handle hObj, unsigned int Index,
                   const void * pBitmap);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Index for bitmap (see table below).
pBitmap	Pointer to bitmap file data

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.

Add. information

If only a bitmap for the unpressed state is set the button will show it also when it is pressed or disabled.

For additional information's regarding bitmap files please take a look at chapter 6, "BMP File Support".

BUTTON_SetBMPEx()

Description

Sets the bitmap(s) to be used when displaying a specified button.

Prototype

```
void BUTTON_SetBMPEx(BUTTON_Handle hObj,
                      unsigned int Index,
                      const void * pBitmap,
                      int x, int y);
```

Parameter	Meaning
hObj	Handle of 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.

Add. information

If only a bitmap for the unpressed state is set the button will show it also when it is pressed or disabled.

For additional informations regarding bitmap files please take a look at chapter 6, "BMP File Support".

BUTTON_SetDefaultBkColor()

Description

Sets the default background color used by new BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Meaning
Color	Color to be used.
Index	Index for color (see table below).

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 new BUTTON widgets.

Prototype

```
GUI_COLOR BUTTON_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Default color to be used for new buttons.

Return value

Previous default focus rectangle color.

Add. information

For more information please refer to the function `BUTTON_SetFocusColor()`.

BUTTON_SetDefaultFont()

Description

Sets a pointer to a GUI_FONT structure used to display the text of new BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Meaning
pFont	Pointer to GUI_FONT structure to be used.

BUTTON_SetDefault TextAlign()

Description

Sets the default text alignment used to display the text of new BUTTON widgets.

Prototype

```
void BUTTON_SetDefault TextAlign(int Align);
```

Parameter	Meaning
Align	Text alignment to be used. For details refer to <code>GUI_SetTextAlign()</code>

BUTTON_SetDefaultTextColor()

Description

Seturns the default text color used to display the text of new BUTTON widgets.

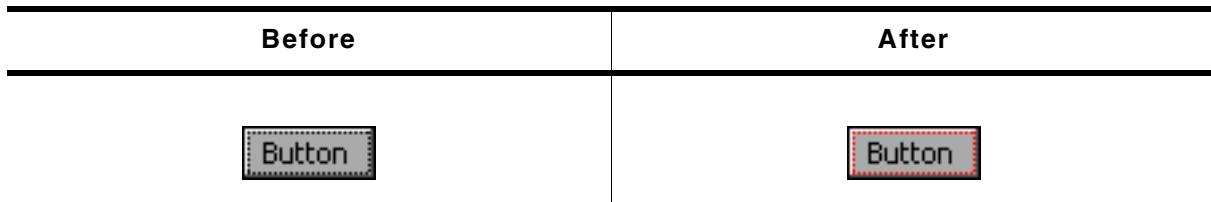
Prototype

```
void BUTTON_SetDefaultTextColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Meaning
Color	Default text color to be used.
Index	Index for color (see table below).

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()



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	Meaning
hObj	Handle of widget.
Color	Color to be used for the focus rectangle.

Return value

Previous color of the focus rectangle.

Add. 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	Meaning
hWin	Window handle.
State	see table below

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 button font.

Prototype

```
void BUTTON_SetFont(BUTTON_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
hObj	Handle of button.
pFont	Pointer to the font.

Add. 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	Meaning
hObj	Handle of button.
State	State, 1 for pressed, 0 for unpressed

BUTTON_SetStreamedBitmap()

Description

Sets the streamed bitmap(s) to be used when displaying a specified button object.

Prototype

```
void BUTTON_SetStreamedBitmap(BUTTON_Handle hObj, unsigned int Index,
                             const GUI_BITMAP_STREAM* pBitmap);
```

Parameter	Meaning
hObj	Handle of button.
Index	Index for bitmap (see <code>BUTTON_SetBitmap()</code>).
pBitmap	Pointer to a bitmap stream.

Add. information

To be able to use this function you must include the following line in `LCDConf.h`:

```
#define BUTTON_SUPPORT_STREAMED_BITMAP 1
```

For details about streamed bitmaps, please see `GUI_DrawStreamedBitmap()`.

BUTTON_SetStreamedBitmapEx()

Description

Sets the streamed bitmap(s) to be used when displaying a specified button object.

Prototype

```
void BUTTON_SetStreamedBitmapEx(BUTTON_Handle hObj,
                                 unsigned int Index,
                                 const GUI_BITMAP_STREAM * pBitmap,
                                 int x, int y);
```

Parameter	Meaning
hObj	Handle of button.
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.

Add. information

To be able to use this function you must include the following line in LCDConf.h:

```
#define BUTTON_SUPPORT_STREAMED_BITMAP 1
```

For details about streamed bitmaps, please see GUI_DrawStreamedBitmap().

BUTTON_SetText()**Description**

Sets the text to be displayed on the button.

Prototype

```
void BUTTON_SetText(BUTTON_Handle hObj, const char* s);
```

Parameter	Meaning
hObj	Handle of button.
s	Text to display.

BUTTON_Set.TextAlign()**Description**

Sets the alignment of the button text.

Prototype

```
void BUTTON_Set.TextAlign(BUTTON_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of button.
Align	Text alignment to be set (see GUI_Set.TextAlign())

Add. information

The default value of the text alignment is GUI_TA_HCENTER | GUI_TA_VCENTER.

BUTTON_SetTextColor()**Description**

Sets the button text color.

Prototype

```
void BUTTON_SetTextColor(BUTTON_Handle hObj, unsigned int Index,
```

```
GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of button.
Index	Index for text color (see <code>BUTTON_SetBkColor()</code>).
Color	Text color to be set.

15.3.5 Examples

Using the BUTTON widget

The following example demonstrates how to use two bitmaps to create a simple button. It is available as `WIDGET_SimpleButton.c` in the samples shipped with emWin:

```
/*
 *      SEGGER MICROCONTROLLER SYSTEME GmbH
 *      Solutions for real time microcontroller applications
 */
/*
 *      (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
 *
 *      Internet: www.segger.com      Support: support@segger.com
 */
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
distributed in any way. We appreciate your understanding and fairness.
-----
File      : WIDGET_SimpleButton.c
Purpose   : Example demonstrating the use of a BUTTON widget
-----
*/
#include "GUI.h"
#include "BUTTON.h"

/*
 *      static code
 */
***** _DemoButton
*/
static void _DemoButton(void) {
    BUTTON_Handle hButton;
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Click on button...", 160, 90);
    /* Create the button*/
    hButton = BUTTON_Create(110, 110, 100, 40, GUI_ID_OK, WM_CF_SHOW);
    /* Set the button text */
    BUTTON_SetText(hButton, "Click me...");
    /* Let window manager handle the button */
    while (GUI_WaitKey() != GUI_ID_OK);
    /* Delete the button*/
    BUTTON_Delete(hButton);
    GUI_ClearRect(0, 50, 319, 239);
    GUI_Delay(1000);
}

/*
 *      MainTask
 *
 *      Demonstrates the use of a simple button
 */
*/
```

```
void MainTask(void) {
    GUI_Init();
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DisplayStringHCenterAt("WIDGET_SimpleButton - Sample", 160, 5);
    while (1) {
        _DemoButton();
    }
}
```

Screen shot of above example



Advanced use of the BUTTON widget

The following example illustrates how to create a button which displays a picture of a telephone. When pressed, the picture changes to show the receiver lifted. The example is also available in the samples as `WIDGET_PhoneButton.c`:


```
*      MainTask
*
*      Demonstrates the use of a BUTTON widget
*
*****
*/
void MainTask(void) {
    GUI_Init();
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("WIDGET_PhoneButton - Sample", 160, 5);
    while (1) {
        _DemoButton();
    }
}
```

Screen shot of above example



15.4 CHECKBOX: Check box 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	<input type="checkbox"/> Item A	<input checked="" type="checkbox"/> Item B	<input checked="" type="checkbox"/> Item C
Disabled	<input type="checkbox"/> Item D	<input checked="" type="checkbox"/> Item E	<input checked="" type="checkbox"/> Item F

All CHECKBOX-related routines are located in the file(s) CHECKBOX*.c, CHECKBOX.h. All identifiers are prefixed CHECKBOX.

15.4.1 Configuration options

Type	Macro	Default	Explanation
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.

15.4.2 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	Explanation
WM_NOTIFICATION_CLICKED	Check box has been clicked.
WM_NOTIFICATION_RELEASED	Check box has been released.
WM_NOTIFICATION_MOVED_OUT	Check box has been clicked and pointer has been moved out off the box without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Status of check box has been changed.

15.4.3 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 widget.

15.4.4 CHECKBOX API

The table below lists the available emWin CHECKBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
CHECKBOX_Check()	Set the check box state to checked. (Obsolete)
CHECKBOX_Create()	Create the check box. (Obsolete)
CHECKBOX_CreateEx()	Creates the check box.
CHECKBOX_CreateIndirect()	Create the check box from resource table entry.
CHECKBOX_GetDefaultBkColor()	Returns the default background color of new check box widgets.
CHECKBOX_GetDefaultFont()	Returns the default font used to display the text of new check box widgets.
CHECKBOX_GetDefaultSpacing()	Returns the default spacing between the box and the text of new check box widgets.
CHECKBOX_GetDefault TextAlign()	Returns the default alignment used to display the check box text of new check box widgets..
CHECKBOX_GetDefaultTextColor()	Returns the default text color used to display the text of new check box widgets.
CHECKBOX_GetState()	Returns the current state of the check box.
CHECKBOX_GetText()	Returns the text of the check box.
CHECKBOX_IsChecked()	Return the current state (checked or not checked) of the check box.
CHECKBOX_SetBkColor()	Sets the background color of new check box widgets.
CHECKBOX_SetBoxBkColor()	Sets the background color of the check box widget.
CHECKBOX_SetDefaultBkColor()	Sets the default background color of new check box widgets.
CHECKBOX_SetDefaultFocusColor()	Sets the default focus rectangle color for new check box widgets.
CHECKBOX_SetDefaultFont()	Sets the default font used to display the text of new check box widgets.
CHECKBOX_SetDefaultImage()	Sets the default image to be shown when a new box has been checked.
CHECKBOX_SetDefaultSpacing()	Sets the default spacing between the box and the text of new check box widgets.
CHECKBOX_SetDefault TextAlign()	Sets the default alignment used to display the check box text.
CHECKBOX_SetDefaultTextColor()	Sets the default text color used to display the text of new check box widgets.
CHECKBOX_SetFocusColor()	Sets the color of the focus rectangle.
CHECKBOX_SetImage()	Sets the image to be shown when box has been checked.
CHECKBOX_SetNumStates()	Sets the number of possible states of the check box (2 or 3).
CHECKBOX_SetSpacing()	Sets the spacing between the box and the check box text.
CHECKBOX_SetState()	Sets the state of the check box.
CHECKBOX_SetText()	Sets the text of the check box.
CHECKBOX_SetTextAlign()	Sets the alignment used to display the check box text.
CHECKBOX_SetTextColor()	Sets the color used to display the text of the check box.
CHECKBOX_Uncheck()	Set the check box state to unchecked (default).

CHECKBOX_Check()

Before	After
<input type="checkbox"/> Item 1	<input checked="" type="checkbox"/> Item 1

Description

Sets a specified check box to a checked state.

Prototype

```
void CHECKBOX_Check(CHECKBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of check box.

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	Meaning
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 <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).

Return value

Handle for the created CHECKBOX widget; 0 if the routine fails.

Add. 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_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	Not used yet, reserved for future use.
Id	Window ID of the widget.

Return value

Handle for the created widget; 0 if the routine fails.

Add. 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 usefull 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_GetDefaultBkColor()

Description

Returns the default background color of new check box widgets.

Prototype

```
GUI_COLOR CHECKBOX_GetDefaultBkColor (void);
```

Return value

Default background color of new check box widgets.

Add. 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 please refer to `CHECKBOX_SetBkColor()`.

CHECKBOX_GetDefaultFont()

Description

Returns a pointer to a GUI_FONT structure used to display the check box text of new check box widgets.

Prototype

```
const GUI_FONT * CHECKBOX_GetDefaultFont(void);
```

Return value

Pointer to a GUI_FONT structure used to display the check box text of new check box widgets.

Add. information

For more information please refer to `CHECKBOX_SetFont()`.

CHECKBOX_GetDefaultSpacing()

Description

Returns the default spacing between box and text used to display the check box text of new check box widgets.

Prototype

```
int CHECKBOX_GetDefaultSpacing(void);
```

Return value

Default spacing between box and text used to display the check box text of new check box widgets.

Add. information

For more information please refer to `CHECKBOX_SetSpacing()`.

CHECKBOX_GetDefault TextAlign()

Description

Returns the default alignment used to display the check box text of new check box widgets.

Prototype

```
int CHECKBOX_GetDefaultAlign(void);
```

Return value

Default alignment used to display the check box text.

Add. information

For more information please refer to `CHECKBOX_SetTextAlign()`.

CHECKBOX_GetDefaultTextColor()

Description

Returns the default text color used to display the check box text of new check box widgets.

Prototype

```
GUI_COLOR CHECKBOX_GetDefaultTextColor(void);
```

Return value

Default text color used to display the check box text of new check box widgets.

Add. information

For more information please refer to `CHECKBOX_SetTextColor()`.

CHECKBOX_GetState()

Description

Returns the current state of the given check box.

Prototype

```
int CHECKBOX_GetState(CHECKBOX_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.

Return value

Current state of the given check box.

Add. 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 please refer to `CHECKBOX_SetNumStates()`.

CHECKBOX_GetText()**Description**

Returns the optional text of the given check box.

Prototype

```
int CHECKBOX_GetText(CHECKBOX_Handle hObj, char * pBuffer, int MaxLen);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>pBuffer</code>	Pointer to buffer to which the text will be copied.
<code>MaxLen</code>	Buffer size in bytes.

Return value

Length of the text copied into the buffer.

Add. information

If the check box contains no text the function returns 0 and the buffer remains unchanged.

CHECKBOX_IsChecked()**Description**

Returns the current state (checked or not checked) of a specified CHECKBOX widget.

Prototype

```
int CHECKBOX_IsChecked(CHECKBOX_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of check box.

Return value

0: not checked
1: checked

CHECKBOX_SetBkColor()

Before	After
Item 1	Item 1

Description

Sets the background color used to display the background of the check box.

Prototype

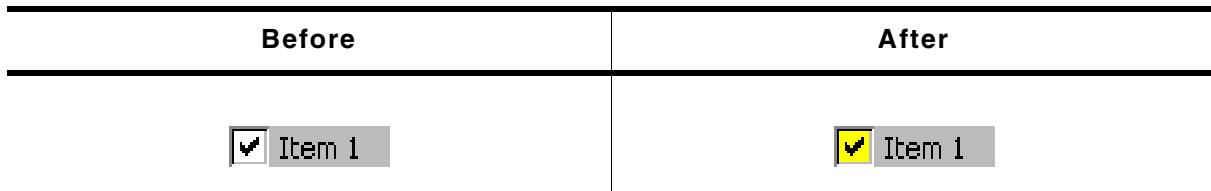
```
void CHECKBOX_SetBkColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Color	Color to be used to draw the background or GUI_INVALID_COLOR to work in transparent mode.

Add. information

If the check box should work in transparent mode GUI_INVALID_COLOR should be used.

CHECKBOX_SetBoxBkColor()



Description

Sets the background color of the box area.

Prototype

```
GUI_COLOR CHECKBOX_SetBoxBkColor(CHECKBOX_Handle hObj,
                                  GUI_COLOR Color, int Index);
```

Parameter	Meaning
hObj	Handle of widget.
Color	Color to be used.
Index	(see table below)

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.

Add. 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 check box widgets.

Prototype

```
void CHECKBOX_SetDefaultBkColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Color to be used, GUI_INVALID_COLOR for transparency.

Add. information

For more information please refer to `CHECKBOX_SetBkColor()`.

CHECKBOX_SetDefaultFocusColor()**Description**

Sets the color used to render the focus rectangle of new check box widgets.

Prototype

```
GUI_COLOR CHECKBOX_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Meaning
<code>Color</code>	Color to be used.

Return value

Previous color used to render the focus rectangle.

Add. information

For mode information please refer to `CHECKBOX_SetFocusColor()`.

CHECKBOX_SetDefaultFont()**Description**

Sets a pointer to a `GUI_FONT` structure used to display the check box text of new check box widgets.

Prototype

```
void CHECKBOX_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Meaning
<code>pFont</code>	Pointer to <code>GUI_FONT</code> structure to be used.

Add. information

For more information please refer to `CHECKBOX_SetFont()`.

CHECKBOX_SetDefaultImage()**Description**

Sets the images used for new check boxes to be shown if they has been checked.

Prototype

```
void CHECKBOX_SetDefaultImage(const GUI_BITMAP * pBitmap,
                             unsigned int Index);
```

Parameter	Meaning
<code>pBitmap</code>	Pointer to bitmap.
<code>Index</code>	(see table below)

Permitted values for parameter <code>Index</code>	
<code>CHECKBOX_BI_INACTIV_UNCHECKED</code>	Sets the bitmap displayed when the check box is unchecked and disabled.
<code>CHECKBOX_BI_ACTIV_UNCHECKED</code>	Sets the bitmap displayed when the check box is unchecked and enabled.
<code>CHECKBOX_BI_INACTIV_CHECKED</code>	Sets the bitmap displayed when the check box is checked and disabled.

Permitted values for parameter Index	
CHECKBOX_BI_ACTIV_CHECKED	Sets the bitmap displayed when the check box is checked and enabled.
CHECKBOX_BI_INACTIV_3STATE	Sets the bitmap displayed when the check box is in the third state and disabled.
CHECKBOX_BI_ACTIV_3STATE	Sets the bitmap displayed when the check box is in the third state and enabled.

Add. information

The image has to fill the complete inner area of the check box.

CHECKBOX_SetDefaultSpacing()

Description

Sets the default spacing between box and text used to display the check box text of new check box widgets.

Prototype

```
void CHECKBOX_SetDefaultSpacing(int Spacing);
```

Parameter	Meaning
Spacing	Number of pixels between box and text used for new check box widgets.

Add. information

For more information please refer to `CHECKBOX_SetSpacing()`.

CHECKBOX_SetDefaultTextAlign()

Description

Sets the default alignment used to display the check box text of new check box widgets.

Prototype

```
void CHECKBOX_SetDefault TextAlign(int Align);
```

Parameter	Meaning
Align	Text alignment used to display the text of new check box widgets.

Add. information

For more information please refer to `CHECKBOX_Set.TextAlign()`.

CHECKBOX_SetDefaultTextColor()

Description

Sets the default text color used to display the check box text of new check box widgets.

Prototype

```
void CHECKBOX_SetDefaultTextColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Color to be used.

Add. information

For more information please refer to `CHECKBOX_SetTextColor()`.

CHECKBOX_SetFocusColor()

Before	After
	

Description

Sets the color used to render the focus rectangle.

Prototype

```
GUI_COLOR CHECKBOX_SetFocusColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.

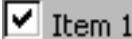
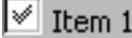
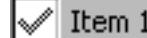
Return value

Previous color of the focus rectangle.

Add. information

The focus rectangle is only visible if the widget has the input focus.

CHECKBOX_SetImage()

Before	After
	
	

Description

Sets the images to be shown if the check box has been checked.

Prototype

```
void CHECKBOX_SetImage(CHECKBOX_Handle hObj,
                      const GUI_BITMAP * pBitmap,
                      unsigned int Index);
```

Parameter	Meaning
<code>hObj</code>	Handle of check box.
<code>pBitmap</code>	Pointer to bitmap.
<code>Index</code>	(see table shown under <code>CHECKBOX_SetDefaultImage</code>)

Add. 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 check box.

Prototype

```
void CHECKBOX_SetNumStates(CHECKBOX_Handle hObj, unsigned NumStates);
```

Parameter	Meaning
hObj	Handle of widget.
NumStates	Number of possible states of the given check box. Currently supported are 2 or 3 states.

Add. 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
Item 1	Item 1

Description

Sets the number of pixels between box and text of a given check box widget.

Prototype

```
void CHECKBOX_SetSpacing(CHECKBOX_Handle hObj, unsigned Spacing);
```

Parameter	Meaning
hObj	Handle of widget.
Spacing	Number of pixels between box and text to be used.

Add. 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
Item 1	Item 1 Item 1

Description

Sets the new state of the given check box widget.

Prototype

```
void CHECKBOX_SetState(CHECKBOX_Handle hObj, unsigned State);
```

Parameter	Meaning
hObj	Handle of widget.
State	Zero based number of new state.

Permitted values for parameter State	
0	Unchecked
1	Checked
2	Third state

Add. information

The passed state should not be greater than the number of possible states set with CHECKBOX_SetNumStates() minus 1.

CHECKBOX_SetText()

Before	After
	Item 1

Description

Sets the optional text shown beside the box.

Prototype

```
void CHECKBOX_SetText(CHECKBOX_Handle hObj, const char * pText);
```

Parameter	Meaning
hObj	Handle of widget.
pText	Pointer to text to be shown beside the box.

Add. information

Clicking on the text beside the box has the same effect as clicking into the box.

CHECKBOX_SetTextAlign()

Before	After
Item 1	Item 1

Description

Sets the alignment used to display the check box text beside the box.

Prototype

```
void CHECKBOX_SetTextAlign(CHECKBOX_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of widget.
Align	Desired text alignment.

Add. information

Per default the text alignment is GUI_TA_LEFT | GUI_TA_VCENTER. The function CHECKBOX_SetDefault TextAlign() and the configuration macro CHECKBOX_TEXTALIGN_DEFAULT can be used to set a user defined default value.

CHECKBOX_SetTextColor()

Before	After
Item 1	Item 1

Description

Sets the color used to display the check box text.

Prototype

```
void CHECKBOX_SetTextColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Color	Desired color of check box text.

Add. 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_Uncheck()

Before	After
Item 1	Item 1

Description

Sets a specified check box to an unchecked state.

Prototype

```
void CHECKBOX_Uncheck(CHECKBOX_Handle hObj);
```

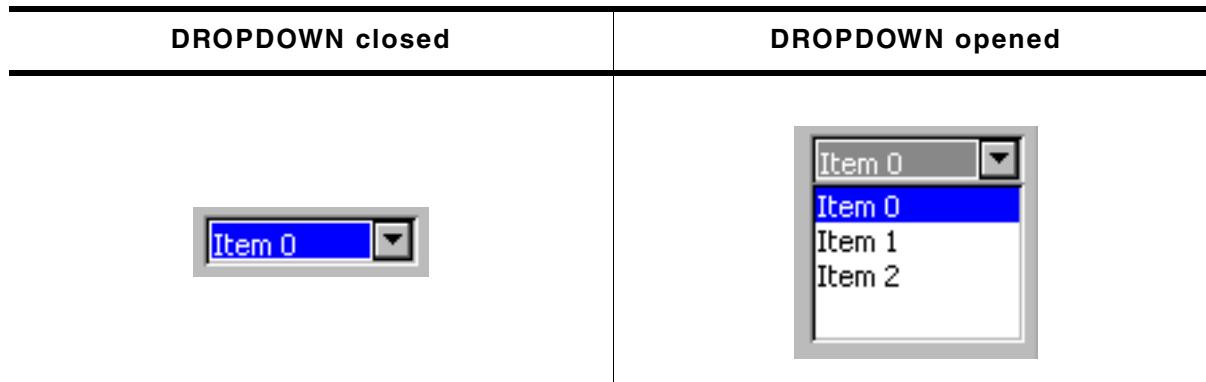
Parameter	Meaning
hObj	Handle of check box.

Add. information

This is the default setting for check boxes.

15.5 DROPODOWN: Dropdown widget

DROPODOWN 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 DROPODOWN widget a LISTBOX appears to select a new item.



15.5.1 Configuration options

Type	Macro	Default	Explanation
N	DROPODOWN_ALIGN_DEFAULT	GUI_TA_LEFT	Text alignment used to display the drop-down text in closed state.
S	DROPODOWN_FONT_DEFAULT	&GUI_Font13_1	Default font
N	DROPODOWN_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	DROPODOWN_BKCOLOR1_DEFAULT	GUI_GRAY	Background color, selected state without focus.
N	DROPODOWN_BKCOLOR2_DEFAULT	GUI_BLUE	Background color, selected state with focus.
N	DROPODOWN_KEY_EXPAND	GUI_KEY_SPACE	Key which can be used to expand the dropdown list.
N	DROPODOWN_KEY_SELECT	GUI_KEY_ENTER	Key which can be used to select an item from the open dropdown list.
N	DROPODOWN_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	DROPODOWN_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, selected state without focus.
N	DROPODOWN_TEXTCOLOR2_DEFAULT	GUI_WHITE	Enable 3D support.

15.5.2 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off of the widget without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scrollbar of the opened dropdown widget has been changed.

15.5.3 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.

15.5.4 DROPODOWN API

The table below lists the available emWin DROPODOWN-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
DROPODOWN_AddString()	Adds an element to the DROPODOWN list.
DROPODOWN_Collapse()	Closes the dropdown list.
DROPODOWN_Create()	Creates a DROPODOWN widget. (Obsolete)
DROPODOWN_CreateEx()	Creates a DROPODOWN widget.
DROPODOWN_CreateIndirect()	Creates a DROPODOWN widget from a resource table entry.
DROPODOWN_DecSel()	Decrements selection.
DROPOWN_DeleteItem()	Deletes an item of the DROPODOWN list.
DROPODOWN_Expand()	Opens the dropdown list.
DROPODOWN_GetDefaultFont()	Returns the default font used to create DROPODOWN widgets.
DROPODOWN_GetNumItems()	Returns the number of items in the dropdown list.
DROPODOWN_GetSel()	Returns the number of the currently selected element.
DROPODOWN_IncSel()	Increments selection.
DROPODOWN_InsertString()	Inserts a string to the dropdown list.
DROPODOWN_SetAutoScroll()	Enables the automatic use of a scrollbar in the dropdown list.
DROPODOWN_SetBkColor()	Sets the background color.
DROPODOWN_SetColor()	Sets the color of the arrow and the button of the widget.
DROPODOWN_SetDefaultColor()	Sets the default color of the arrow and the button for new DROPODOWN widgets.
DROPODOWN_SetDefaultFont()	Sets the default font used to create DROPODOWN widgets.
DROPODOWN_SetDefaultScrollbarColor()	Sets the default colors of the optional scrollbar in the dropdown list.
DROPODOWN_SetFont()	Sets the font of the given DROPODOWN widget
DROPODOWN_SetItemSpacing()	Sets an additional spacing between the items of the dropdown list.
DROPODOWN_SetScrollbarColor()	Sets the colors of the scrollbar in the dropdown list.
DROPODOWN_SetSel()	Sets the current selection
DROPODOWN_SetTextAlign()	Sets the text alignment used to display the dropdown text in closed state.
DROPODOWN_SetTextColor()	Sets the text color of the given DROPODOWN widget.
DROPODOWN_SetTextHeight()	Sets the height of the rectangle used to display the dropdown text in closed state.

DROPODOWN_AddString()

Description

Adds a new element to the dropdown list.

Prototype

```
void DROPODOWN_AddString(DROPODOWN_Handle hObj, const char* s);
```

Parameter	Meaning
hObj	Handle of widget
s	Pointer to string to be added

DROPDOWN_Collapse()

Description

Closes the dropdown list of the DROPODOWN widget.

Prototype

```
void DROPODOWN_Collapse(DROPODOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

DROPDOWN_Create()

(Obsolete, DROPODOWN_CreateEx should be used instead)

Description

Creates a DROPODOWN widget of a specified size at a specified location.

Prototype

```
DROPODOWN_Handle DROPODOWN_Create(WM_HWIN hWinParent,
                                    int x0, int y0,
                                    int xsize, int ysize,
                                    int Flags);
```

Parameter	Meaning
hWinParent	Handle of parent window
x0	Leftmost pixel of the DROPODOWN widget (in parent coordinates).
y0	Topmost pixel of the DROPODOWN widget (in parent coordinates).
xsize	Horizontal size of the DROPODOWN widget (in pixels).
ysize	Vertical size of the DROPODOWN widget in open state (in pixels).
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 12: "The Window Manager" for a list of available parameter values).

Return value

Handle for the created DROPODOWN widget, 0 on failure.

Add. 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 DROPODOWN widget.

DROPDOWN_CreateEx()

Description

Creates a DROPODOWN widget of a specified size at a specified location.

Prototype

```
DROPODOWN_Handle DROPODOWN_CreateEx(int x0, int y0, int xsize, int ysize,
                                       WM_HWIN hParent, int WinFlags,
                                       int ExFlags, int Id);
```

Parameter	Meaning
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).

Parameter	Meaning
<code>hParent</code>	Handle of parent window. If 0, the new DROPODOWN widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	(see table below)
<code>Id</code>	Window ID of the widget.

Permitted values for parameter <code>ExFlags</code>	
0	No function.
DROPODOWN_CF_AUTOSCROLLBAR	Enable automatic use of a scrollbar. For details please refer to DROPODOWN_SetAutoScroll
DROPODOWN_CF_UP	Creates a DROPODOWN 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 for the created widget; 0 if the routine fails.

DROPODOWN_CreateIndirect()

Prototype explained at the beginning of the chapter.

DROPODOWN_DecSel()

Description

Decrement the selection, moves the selection of a specified DROPODOWN widget up by one item.

Prototype

```
void DROPODOWN_DecSel(DROPODOWN_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget

DROPODOWN_DeleteItem()

Description

Deletes the given item of the dropdown list.

Prototype

```
void DROPODOWN_DeleteItem(DROPODOWN_Handle hObj, unsigned int Index);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>Index</code>	Zero based index of the item to be deleted.

Add. information

If the index is greater than the number of items < 1 the function returns immediately.

DROPODOWN_Expand()

Description

Opens the dropdown list of the widget.

Prototype

```
void DROPODOWN_Expand(DROPODOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Add. information

The dropdown list remains open until an element has been selected or the focus has been lost.

DROPODOWN_GetNumItems()**Description**

Returns the number of items in the given DROPODOWN widget.

Prototype

```
int DROPODOWN_GetNumItems(DROPODOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

number of items in the given DROPODOWN widget

DROPODOWN_GetSel()**Description**

Returns the number of the currently selected item in a specified DROPODOWN widget.

Prototype

```
int DROPODOWN_GetSel(DROPODOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

Number of the currently selected item.

DROPODOWN_IncSel()**Description**

Increment the selection, moves the selection of a specified DROPODOWN widget down by one item.

Prototype

```
void DROPODOWN_IncSel(DROPODOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

DROPODOWN_InsertString()**Description**

Inserts a string to the dropdown list at the given position.

Prototype

```
void DROPODOWN_InsertString(DROPODOWN_Handle hObj,
                            const char* s,
                            unsigned int Index);
```

Parameter	Meaning
hObj	Handle of widget.
s	Pointer to the string to be inserted.
Index	Zero based index of the position.

Add. information

If the given index is greater than the number of items the string will be appended to the end of the dropdown list.

DROPODOWN_SetAutoScroll()

Description

Enables the automatic use of a vertical scrollbar in the dropdown list.

Prototype

```
void DROPODOWN_SetAutoScroll(DROPODOWN_Handle hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of widget.
OnOff	(see table below)

Permitted values for parameter OnOff	
0	Disable automatic use of a scrollbar.
1	Enable automatic use of a scrollbar.

Add. information

If enabled the dropdown list checks if all elements fits into the listbox. If not a vertical scrollbar will be added.

DROPODOWN_SetBkColor()

Before	After
	

Description

Sets the background color of the given DROPODOWN widget.

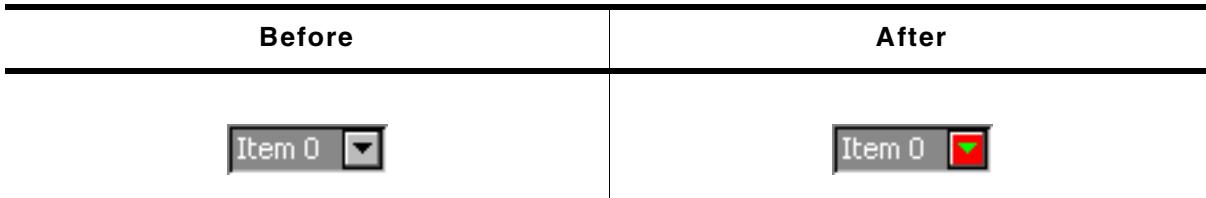
Prototype

```
void DROPODOWN_SetBkColor(DROPODOWN_Handle hObj,
                           unsigned int Index,
                           GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index for background color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
DROPDOWN_CI_UNSEL	Unselected element.
DROPDOWN_CI_SEL	Selected element, without focus.
DROPDOWN_CI_SELF_FOCUS	Selected element, with focus.

DROPDOWN_SetColor()



Description

Sets the color of the button or the arrow of the given DROPODOWN widget.

Prototype

```
void DROPODOWN_SetColor(DROPODOWN_Handle hObj,
                        unsigned int Index,
                        GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index of desired item (see table below).
Color	Color to be used.

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 DROPODOWN widgets.

Prototype

```
GUI_COLOR DROPODOWN_SetDefaultColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Please refer to the function DROPODOWN_SetColor().
Color	Color to be used.

DROPDOWN_SetDefaultFont()

Description

Sets the default font used for new DROPODOWN widgets.

Prototype

```
void DROPODOWN_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Meaning
pFont	Pointer to GUI_FONT structure.

DROPDOWN_SetDefaultScrollbarColor()

Description

Sets the default colors used for the optional scrollbar in the dropdown list.

Prototype

```
GUI_COLOR DROPODOWN_SetDefaultScrollbarColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Please refer to the function DROPODOWN_SetScrollbarColor().
Color	Color to be used.

DROPDOWN_SetFont()

Description

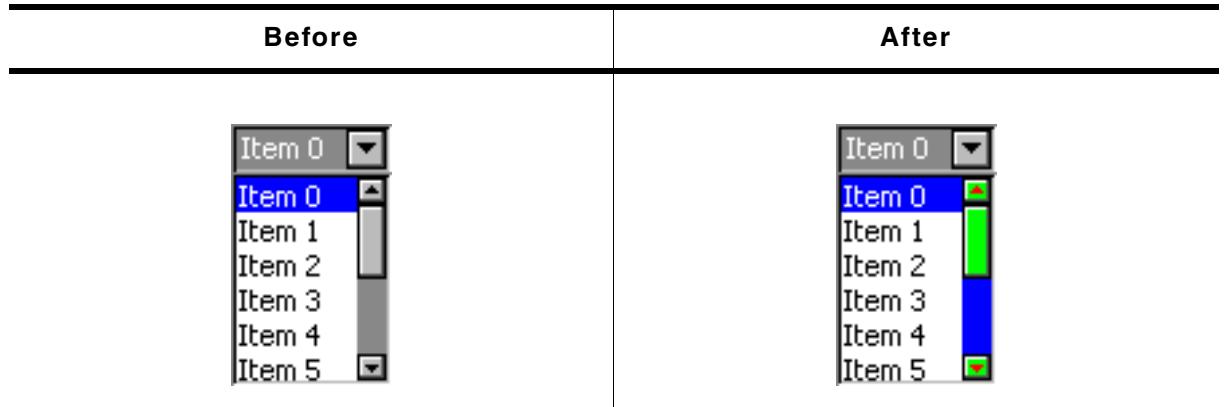
Sets the font used to display the given DROPODOWN widget.

Prototype

```
void DROPODOWN_SetFont(DROPODOWN_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
hObj	Handle of widget
pFont	Pointer to the font.

DROPDOWN_SetScrollbarColor()



Description

Sets the colors of the optional scrollbar in the dropdown list.

Prototype

```
void DROPODOWN_SetScrollbarColor(DROPODOWN_Handle hObj,
                                  unsigned int Index,
                                  GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index of desired item (see table below).
Color	Color to be used.

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 scrollbars used by the dropdown list of the given DROPODOWN widget.

Prototype

```
void DROPODOWN_SetScrollbarWidth(DROPODOWN_Handle hObj, unsigned Width);
```

Parameter	Meaning
hObj	Handle of widget.
Width	Width of the scrollbar(s) used by the dropdown list of the given DROPODOWN widget.

DROPDOWN_SetSel()

Description

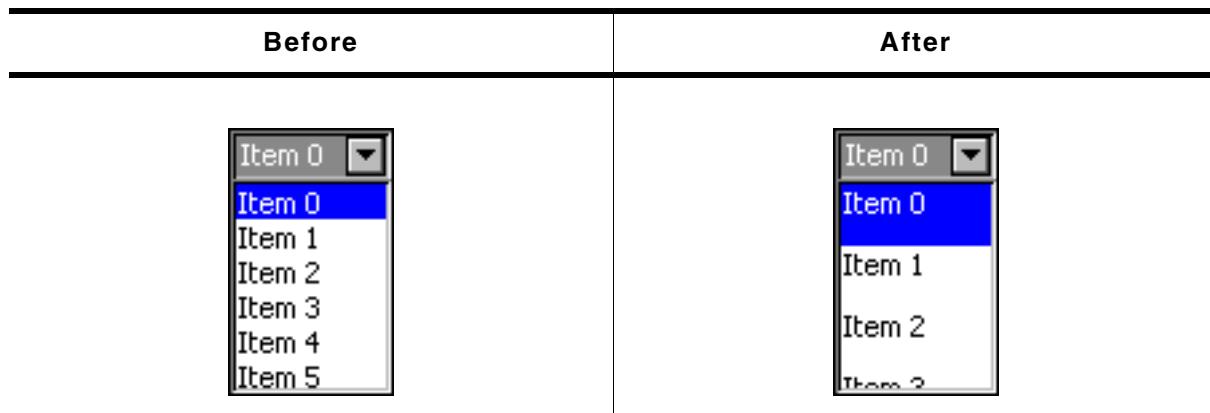
Sets the selected item of a specified DROPODOWN widget.

Prototype

```
void DROPODOWN_SetSel(DROPODOWN_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of widget
Sel	Element to be selected.

DROPDOWN_SetItemSpacing()



Description

Sets an additional spacing below the items of the dropdown list.

Prototype

```
void DROPODOWN_SetItemSpacing(DROPODOWN_Handle hObj, unsigned Value);
```

Parameter	Meaning
hObj	Handle of widget
Value	Number of pixels used as additional space between the items of the dropdown list.

DROPDOWN_SetTextAlign()

Before	After
	

Description

Sets the alignment used to display the dropdown text in closed state.

Prototype

```
void DROPOWN_SetTextAlign(DROPOWN_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of widget
Align	Alignment used to display the dropdown text in closed state.

DROPDOWN_SetTextColor()

Description

Sets the background color of the given DROPOWN widget.

Prototype

```
void DROPOWN_SetTextColor(DROPOWN_Handle hObj,
                           unsigned int Index,
                           GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index for background color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
DROPOWN_CI_UNSEL	Unselected element.
DROPOWN_CI_SEL	Selected element, without focus.
DROPOWN_CI_SELFOCUS	Selected element, with focus.

DROPDOWN_SetTextHeight()

Before	After
	

Description

Sets the height of the rectangle used to display the DROPOWN text in closed state.

Prototype

```
void DROPOWN_SetTextHeight(DROPOWN_Handle hObj, unsigned TextHeight);
```

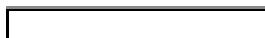
Parameter	Meaning
hObj	Handle of widget
TextHeight	Height of the rectangle used to display the text in closed state.

Add. Information

Per default the height of the DROPOWN widget depends on the used font. Using this function with `TextHeight > 0` means the given value should be used. `TextHeight = 0` means the default behavior should be used.

15.6 EDIT: Edit widget

Edit fields are commonly used as the primary user interface for text input:

Blank edit field	Edit field with user input
	

You can also use edit fields for entering values in binary, decimal, or hexadecimal modes. A decimal-mode edit field might appear similar to those in the following table. Like a check box, an edit field will appear gray if disabled:

Edit field with user input (decimal)	Disabled edit field
	

All EDIT-related routines are located in the file(s) EDIT*.c, EDIT.h. All identifiers are prefixed EDIT.

15.6.1 Configuration options

Type	Macro	Default	Explanation
S	EDIT_ALIGN_DEFAULT	GUI_TA_RIGHT GUI_TA_VCENTER	Alignment for edit field text.
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 field.

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.

15.6.2 Notification codes

The following events are sent from an edit widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
WM_NOTIFICATION_VALUE_CHANGED	Value (content) of the edit widget has changed.

15.6.3 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 please refer to the function EDIT_SetInsertMode().

15.6.4 EDIT API

The table below lists the available emWin EDIT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
EDIT_AddKey()	Key input routine.
EDIT_Create()	Create the edit field. (Obsolete)
EDIT_CreateAsChild()	Create the edit field as a child window. (Obsolete)
EDIT_CreateEx()	Create the edit field.
EDIT_CreateIndirect()	Create the edit field from resource table entry.
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_GetNumChars()	Returns the number of characters of the given edit widget.
EDIT_GetText()	Get user input.
EDIT_GetValue()	Returns the current value.
EDIT_SetBinMode()	Enables the binary edit mode.
EDIT_SetBkColor()	Sets the background color of the edit field.
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 fields.
EDIT_SetDefaultTextAlign()	Sets the default text alignment for edit fields.
EDIT_SetDefaultTextColor()	Sets the default text color.
EDIT_SetFloatMode()	Enables the floating point edit mode.
EDIT_SetFloatValue()	Sets the floating point value if using the floating point edit mode.
EDIT_SetFont()	Select the font for the text.
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 field.
EDIT_SetpfAddKeyEx()	Sets a function which is called to add a character.
EDIT_SetSel()	Sets the current selection.

Routine	Explanation
<code>EDIT_SetText()</code>	Sets the text.
<code>EDIT_SetTextAlign()</code>	Sets the text alignment for the edit field.
<code>EDIT_SetTextColor()</code>	Sets the color(s) for the text.
<code>EDIT_SetTextMode()</code>	Sets the edit mode of the widget back to text mode.
<code>EDIT_SetValue()</code>	Sets the current value.
<code>GUI>EditBin()</code>	Edits a binary value at the current cursor position.
<code>GUI>EditDec()</code>	Edits a decimal value at the current cursor position.
<code>GUI>EditHex()</code>	Edits a hexadecimal value at the current cursor position.
<code>GUI>EditString()</code>	Edits a string at the current cursor position.

EDIT_AddKey()

Description

Adds user input to a specified edit field.

Prototype

```
void EDIT_AddKey(EDIT_Handle hObj, int Key);
```

Parameter	Meaning
<code>hObj</code>	Handle of edit field.
<code>Key</code>	Character to be added.

Add. 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	Meaning
<code>x0</code>	Leftmost pixel of the edit field (in parent coordinates).
<code>y0</code>	Topmost pixel of the edit field (in parent coordinates).
<code>xsize</code>	Horizontal size of the edit field (in pixels).
<code>ysize</code>	Vertical size of the edit field (in pixels).
<code>Id</code>	ID to be returned.
<code>MaxLen</code>	Maximum count of characters.
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).

Return value

Handle for the created EDIT widget; 0 if the routine 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	Meaning
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 returned.
Flags	Window create flags (see EDIT_Create()).
MaxLen	Maximum count of characters.

Return value

Handle for the created EDIT widget; 0 if the routine fails.

EDIT_CreateEx()

Description

Creates a 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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" 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.

Return value

Handle for the created widget; 0 if the routine fails.

EDIT_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flags may be used as the `Flags` element of the resource passed as parameter:

Permitted indirect creation flags ("OR" combinable)	
EDIT_CF_LEFT	Horizontal alignment: left
EDIT_CF_RIGHT	Horizontal alignment: right
EDIT_CF_HCENTER	Horizontal alignment: center
EDIT_CF_TOP	Vertical alignment: top
EDIT_CF_BOTTOM	Vertical alignment: bottom
EDIT_CF_VCENTER	Vertical alignment: center

The `Para` element is used as maximum length of a string to display / max. no. of digits if used in decimal, bin or hex mode.

EDIT_GetDefaultBkColor()

Description

Returns the default background color used for EDIT widgets.

Prototype

```
GUI_COLOR EDIT_GetDefaultBkColor(unsigned int Index);
```

Parameter	Meaning
Index	(see table below)

Permitted values for parameter Index	
0	Color to be used when widget is inactive.
1	Color to be used when widget is active.

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	Meaning
Index	Has to be 0, reserved for future use.

Return value

Default text color used for EDIT widgets.

EDIT_GetFloatValue()

Description

Returns the current value of the edit field as floating point value.

Prototype

```
float EDIT_GetFloatValue(EDIT_Handle hObj);
```

Parameter	Meaning
hObj	Handle of edit field.

Return value

The current value.

Add. Information

The use of this function makes only sense if the edit field is in floating point edit mode.

EDIT_GetNumChars

Description

Returns the number of characters of the specified edit field.

Prototype

```
int EDIT_GetNumChars(EDIT_Handle hObj);
```

Parameter	Meaning
hObj	Handle of edit field.

Return value

Number of characters of the specified edit field.

EDIT_GetText()

Description

Retrieves the user input of a specified edit field.

Prototype

```
void EDIT_GetText(EDIT_Handle hObj, char* sDest, int MaxLen);
```

Parameter	Meaning
hObj	Handle of edit field.
sDest	Pointer to buffer.
MaxLen	Size of buffer.

EDIT_GetValue()

Description

Returns the current value of the edit field. The current value is only useful if the edit field is in binary, decimal or hexadecimal mode.

Prototype

```
I32 EDIT_GetValue(EDIT_Handle hObj);
```

Parameter	Meaning
hObj	Handle of edit field.

Return value

The current value.

EDIT_SetBinMode()

Description

Enables the binary edit mode of the edit field. The given value can be modified in the given range.

Prototype

```
void EDIT_SetBinMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

Parameter	Meaning
hObj	Handle of edit field.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.

EDIT_SetBkColor()

Description

Sets the edit field background color.

Prototype

```
void EDIT_SetBkColor(EDIT_Handle hObj, unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of edit field.
Index	Index for background color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
EDIT_CI_DISABLED	Disabled state.
EDIT_CI_ENABLED	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	Meaning
hObj	Handle of edit field.
xPos	Character position to set cursor to.

Add. 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	Meaning
hObj	Handle of edit field.
Pos	Pixel position to set cursor to.

EDIT_SetDecMode()

Description

Enables the decimal edit mode of the edit field. 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	Meaning
hObj	Handle of edit field.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Shift	If > 0 it specifies the position of the decimal point.
Flags	See table below.

Permitted values for parameter Flags ("OR" combinable)	
0 (default)	Edit in normal mode, sign is only displayed 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	Meaning
Index	(see table below)
Color	Color to be used.

Permitted values for parameter Index	
0	Color to be used when widget is inactive.
1	Color to be used when widget is active.

EDIT_SetDefaultFont()

Description

Sets the default font used for edit fields.

Prototype

```
void EDIT_SetDefaultFont(const GUI_FONT* pFont);
```

Parameter	Meaning
pFont	Pointer to the font to be set as default.

EDIT_SetDefaultTextAlign()

Description

Sets the default text alignment for edit fields.

Prototype

```
void EDIT_SetDefault TextAlign(int Align);
```

Parameter	Meaning
Align	Default text alignment. For details please refer to GUI_Set TextAlign().

EDIT_SetDefaultTextColor()

Description

Sets the default text color used for edit widgets.

Prototype

```
void EDIT_SetDefaultTextColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Has to be 0, reserved for future use.
Color	Color to be used.

EDIT_SetFloatMode()

Description

Enables the floating point edit mode of the edit field. 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	Meaning
hObj	Handle of edit field.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.
Shift	Number of post decimal positions.
Flags	See table below.

Permitted values for parameter Flags ("OR" combinable)	
0 (default)	Edit in normal mode, sign is only displayed if the value is negative
GUI_EDIT_SIGNED	"+" and "-" sign is displayed.

EDIT_SetFloatValue**Description**

The function can be used to set the floating point value of the edit field if working in floating point mode.

Prototype

```
void EDIT_SetFloatValue(EDIT_Handle hObj, float Value);
```

Parameter	Meaning
hObj	Handle of edit field.
Value	New floating point value of the edit field.

Add. Information

The use of this function makes only sense if the edit field 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 field is undefined.

EDIT_SetFont()**Description**

Sets the edit field font.

Prototype

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
hObj	Handle of edit field.
pFont	Pointer to the font.

EDIT_SetHexMode()**Description**

Enables the hexadecimal edit mode of the edit field. The given value can be modified in the given range.

Prototype

```
void EDIT_SetHexMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

Parameter	Meaning
hObj	Handle of edit field.
Value	Value to be modified.
Min	Minimum value.
Max	Maximum value.

EDIT_SetInsertMode

Description

Enables or disables the insert mode of the edit widget.

Prototype

```
int EDIT_SetInsertMode(EDIT_Handle hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of edit field.
OnOff	1 for insert mode (default if working in text mode), 0 for overwrite mode.

Permitted values for parameter OnOff	
0	Disable insert mode.
1	Enable insert mode.

Return value

Returns the previous insert mode state.

Add. 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 field.

Prototype

```
void EDIT_SetMaxLen(EDIT_Handle hObj, int MaxLen);
```

Parameter	Meaning
hObj	Handle of edit field.
MaxLen	Number of characters.

EDIT_SetpfAddKeyEx

Description

Sets the function pointer which is used by the EDIT widget to call the function which is responsible for adding characters.

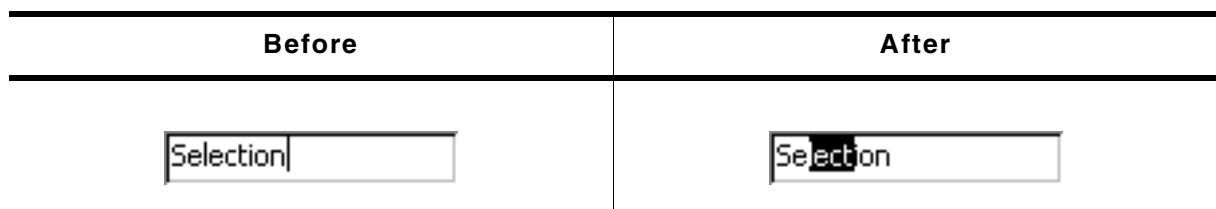
Prototype

```
void EDIT_SetpfAddKeyEx(EDIT_Handle hObj, tEDIT_AddKeyEx * pfAddKeyEx);
```

Parameter	Meaning
hObj	Handle of edit field.
pfAddKeyEx	Function pointer to the function to be used to add a character.

Add. 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 contents of the text buffer.

EDIT_SetSel**Description**

Used to set the current selection of the edit field.

Prototype

```
void EDIT_SetSel(EDIT_Handle hObj, int FirstChar, int LastChar);
```

Parameter	Meaning
hObj	Handle of edit field.
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.

Add. Information

Selected characters are usually displayed in reverse. Setting the cursor position deselects all characters.

Sample

```
EDIT_SetSel(0, -1) /* Selects all characters of the widget */
EDIT_SetSel(-1, 0) /* Deselect all characters */
EDIT_SetSel(0, 2) /* Selects the first 3 characters */
```

EDIT_SetText()**Description**

Sets the text to be displayed in the edit field.

Prototype

```
void EDIT_SetText(EDIT_Handle hObj, const char* s)
```

Parameter	Meaning
hObj	Handle of edit field.
s	Text to display.

EDIT_SetTextAlign()

Description

Sets the text alignment of the edit field.

Prototype

```
void EDIT_SetTextAlign(EDIT_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of edit field.
Align	Text alignment to be set (see <code>GUI_SetTextAlign()</code>)

EDIT_SetTextColor()

Description

Sets the edit field text color.

Prototype

```
void EDIT_SetTextColor(EDIT_Handle hObj, unsigned int Index,
                      GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of edit field.
Index	Has to be 0, reserved for future use.
Color	Color to be set.

EDIT_SetTextMode()

Description

Sets the edit mode of the widget back to text mode.

Prototype

```
void EDIT_SetTextMode(EDIT_Handle hEdit);
```

Parameter	Meaning
hObj	Handle of widget.

Add. information

If one of the functions `EDIT_SetBinMode()`, `EDIT_SetDecMode()`, `EDIT_SetFloatMode()` or `EDIT_SetHexMode()` has been used to set the edit field to one of the numeric edit modes, this function sets the edit mode back to text mode. It also clears the contents of the widget.

EDIT_SetValue()

Description

Sets the current value of the edit field. Only useful if binary, decimal or hexadecimal edit mode is set.

Prototype

```
void EDIT_SetValue(EDIT_Handle hObj, I32 Value);
```

Parameter	Meaning
hObj	Handle of edit field.
Value	New value.

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	Meaning
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.

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Add. Information

The routine returns after pressing <ENTER> or <ESC>. The contents 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	Meaning
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().

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Add. Information

The routine returns after pressing <ENTER> or <ESC>. The contents 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	Meaning
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.

Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

Add. Information

The routine returns after pressing <ENTER> or <ESC>. The contents 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	Meaning
pString	Pointer to the string to be edited.
Len	Maximum number of characters.
xsize	Pixel-size in X of the edit field.

Add. Information

The routine returns after pressing <ENTER> or <ESC>. The contents of the given text will be modified only if <ENTER> is pressed.

15.6.5 Example

The following example demonstrates the use of the EDIT widget. It is available as WIDGET>Edit.c in the samples shipped with emWin:

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*****
*
*          (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
*
*          Internet: www.segger.com      Support: support@segger.com
*
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
distributed in any way. We appreciate your understanding and fairness.
-----
File       : WIDGET>Edit.c
Purpose    : Example demonstrating the use of a EDIT widget
-----
*/
#include "GUI.h"
```

```
#include "EDIT.h"

*****
*
*      static code
*
*****
*/
/* _DemoEdit
*
   Edit a string until ESC or ENTER is pressed
*/
static void _DemoEdit(void) {
    EDIT_Handle hEdit;
    char aBuffer[28];
    int Key;
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("WIDGET_Edit - Sample", 160, 5);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Use keyboard to modify string...", 160, 90);
    /* Create edit widget */
    hEdit = EDIT_Create( 50, 110, 220, 25, ' ', sizeof(aBuffer), 0 );
    /* Modify edit widget */
    EDIT_SetText(hEdit, "Press <ENTER> when done...");
    EDIT_SetFont(hEdit, &GUI_Font8x16);
    EDIT_SetTextColor(hEdit, 0, GUI_RED);
    /* Set keyboard focus to edit widget */
    WM_SetFocus(hEdit);
    /* Handle keyboard until ESC or ENTER is pressed */
    do {
        WM_Exec();
        Key = GUI_GetKey();
    } while ((Key != GUI_KEY_ENTER) && (Key != GUI_KEY_ESCAPE));
    /* Fetch result from edit widget */
    if (Key == GUI_KEY_ENTER) {
        EDIT_GetText(hEdit, aBuffer, sizeof(aBuffer));
    }
    /* Delete the edit widget */
    EDIT_Delete(hEdit);
    GUI_ClearRect(0, 50, 319, 239);
    /* Display the changed string */
    if (Key == GUI_KEY_ENTER) {
        GUI_Delay(250);
        GUI_DispStringHCenterAt("The string you have modified is:", 160, 90);
        GUI_DispStringHCenterAt(aBuffer, 160, 110);
        GUI_Delay(3000);
        GUI_ClearRect(0, 50, 319, 239);
    }
    GUI_Delay(500);
}

*****
*
*      MainTask
*
*      Demonstrates the use of a EDIT widget
*
*****
*/
void MainTask(void) {
    GUI_Init();
    while (1) {
        _DemoEdit();
    }
}
```

Screen shot of above example

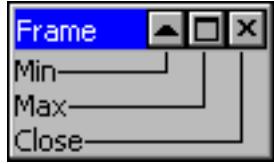
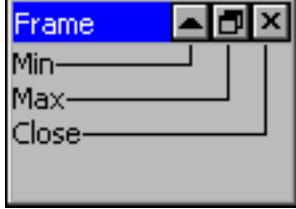
Use keyboard to modify string...
Press <ENTER> when done...

15.7 FRAMEWIN: Frame window widget

Frame windows 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:

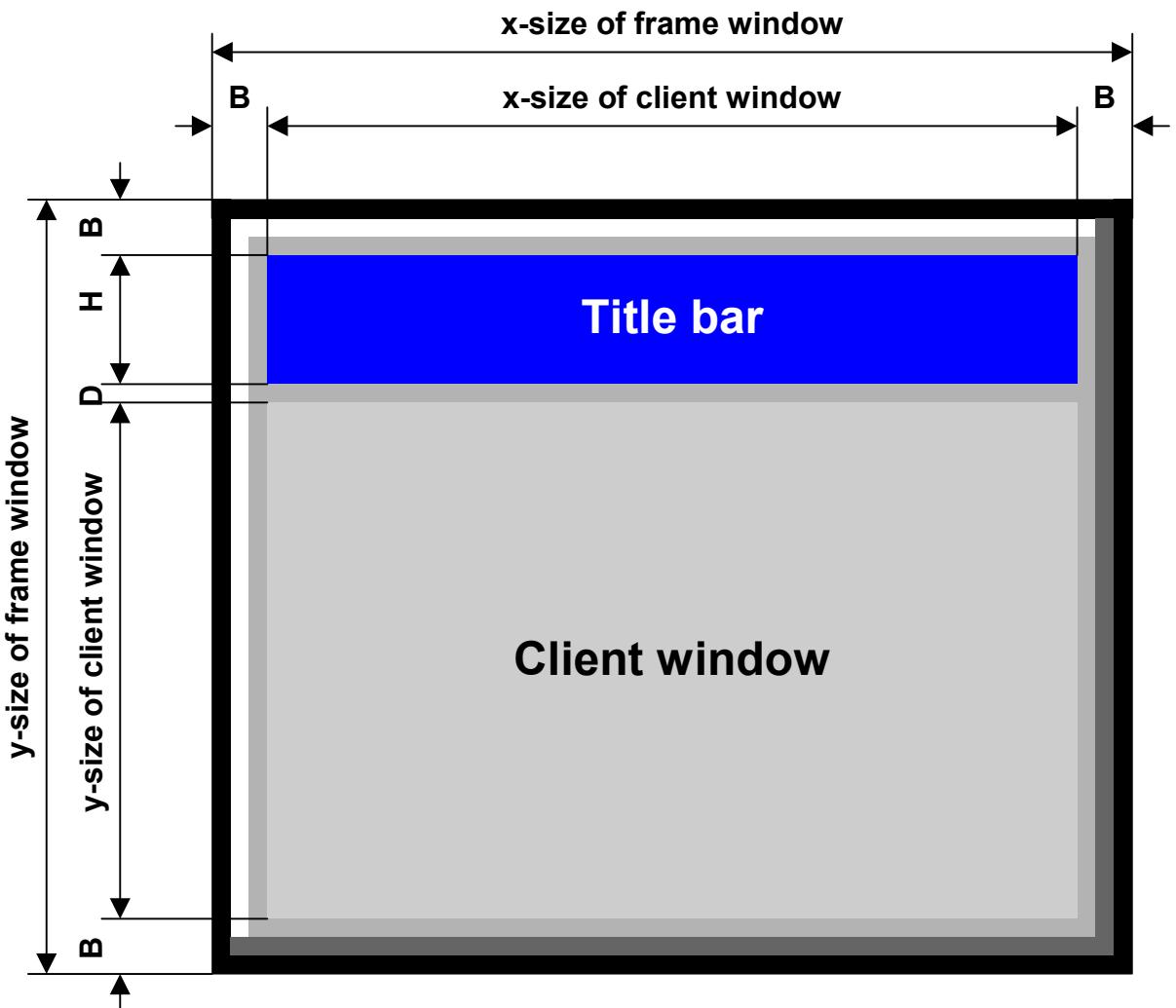
Active frame window	Inactive frame window
	

You can attach predefined buttons to the title bar as seen below or you can attach your own buttons to a title bar:

Explanation	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	

15.7.1 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
<code>B</code>	Border size of the frame window. The default size of the border is 3 pixels.
<code>H</code>	Height of the title bar. Depends on the size of the used font for the title.
<code>D</code>	Spacing between title bar and client window. (1 pixel)
<code>Title bar</code>	The title bar is part of the frame window and not a separate window.
<code>Client window</code>	The client window is a separate window created as a child window of the frame window.

15.7.2 Configuration options

Type	Macro	Default	Explanation
N	FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT	0xff0000	Title bar color, active state.
N	FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT	0x404040	Title bar color, inactive state.
N	FRAMEWIN_BORDER_DEFAULT	3	Outer border width, in pixels.
N	FRAMEWIN_CLIENTCOLOR_DEFAULT	0xc0c0c0	Color of client window area.
S	FRAMEWIN_DEFAULT_FONT	&GUI_Font8_1	Font used for title bar text.
N	FRAMEWIN_FRAMECOLOR_DEFAULT	0xaaaaaaaa	Frame color.
N	FRAMEWIN_IBORDER_DEFAULT	1	Inner border width, in pixels.
N	FRAMEWIN_TITLEHEIGHT_DEFAULT	0	Default height of title bar.

15.7.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

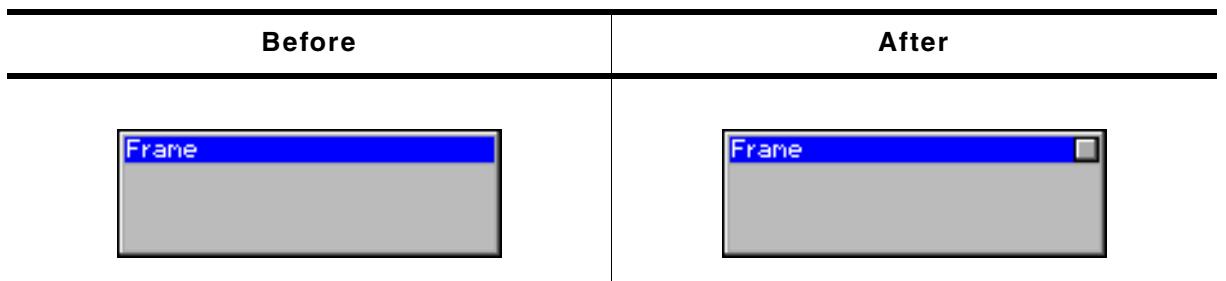
15.7.4 FRAMEWIN API

The table below lists the available emWin FRAMEWIN-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
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 the frame window. (Obsolete)
FRAMEWIN_CreateAsChild()	Creates the frame window as a child window. (Obsolete)
FRAMEWIN_CreateEx()	Creates the frame window.
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
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_GetTitleHeight()	Returns the height of the title bar.
FRAMEWIN_IsMinimized()	Returns if the framewin is minimized or not.
FRAMEWIN_IsMaximized()	Returns if the framewin is maximized or not.
FRAMEWIN_Maximize()	Enlarges the frame window to the size of its parent.
FRAMEWIN_Minimize()	Hides the client area of the frame window.
FRAMEWIN_Restore()	Restores a minimized or maximized frame window.
FRAMEWIN_SetActive()	Sets the state of the frame window. (Obsolete)
FRAMEWIN_SetBarColor()	Sets the color of the title bar.
FRAMEWIN_SetBorderSize()	Sets the border size of the frame window.
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

Routine	Explanation
<code>FRAMEWIN_SetFont()</code>	Selects the font used for the title text.
<code>FRAMEWIN_SetMoveable()</code>	Sets the frame window to a moveable/non-moveable state.
<code>FRAMEWIN_SetResizeable()</code>	Sets the frame window to resizable state.
<code>FRAMEWIN_SetText()</code>	Sets the title text.
<code>FRAMEWIN_SetTextAlign()</code>	Sets the alignment of the title text.
<code>FRAMEWIN_SetTextColor()</code>	Sets the color(s) for the title text.
<code>FRAMEWIN_SetTextColorEx()</code>	Sets the color(s) for the title text.
<code>FRAMEWIN_SetTitleHeight()</code>	Sets the height of the title bar.
<code>FRAMEWIN_SetTitleVis()</code>	Sets the visibility flag of the title bar

FRAMEWIN_AddButton()



Description

Adds a button to the title bar of the frame window.

Prototype

```
WM_HWIN FRAMEWIN_AddButton(FRAMEWIN_Handle hObj,
                           int Flags, int Off, int Id);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>Flags</code>	(see table below)
<code>Off</code>	X-offset used to create the BUTTON widget
<code>Id</code>	ID of the BUTTON widget

Permitted values for parameter <code>Flags</code>	
0	The BUTTON will be created at the left side.
FRAMEWIN_BUTTON_RIGHT	The BUTTON will be created at the right side.

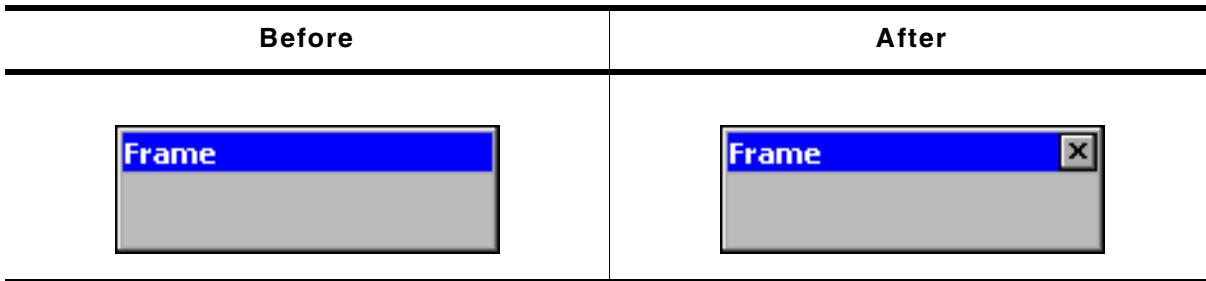
Return value

Handle of the BUTTON widget.

Add. information

The button will be created as a child window from the frame window. So the window manager keeps sure it will be deleted when the frame window 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 frame window or the space between the previous created button.

FRAMEWIN_AddCloseButton()



Description

Adds a close button to the title bar of the frame window.

Prototype

```
WM_HWIN FRAMEWIN_AddCloseButton(FRAMEWIN_Handle hObj,
                                  int Flags, int Off);
```

Parameter	Meaning
hObj	Handle of frame window.
Flags	(see table below)
Off	X-offset used to create the BUTTON widget

Permitted values for parameter Index	
0	The BUTTON will be created at the left side.
FRAMEWIN_BUTTON_RIGHT	The BUTTON will be created at the right side.

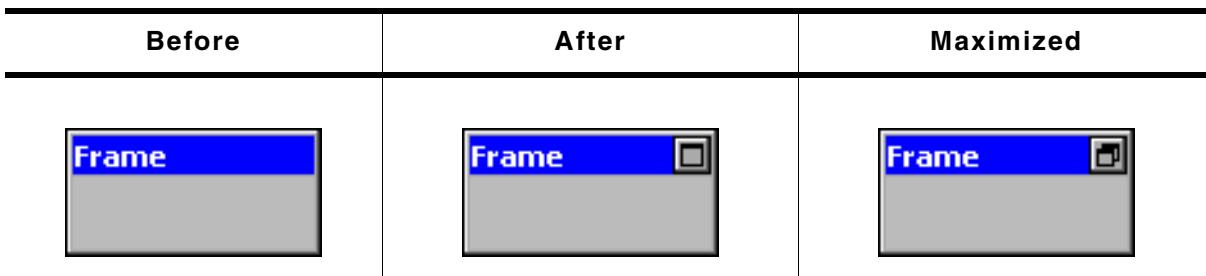
Return value

Handle of the close button.

Add. information

When the user presses the close button the frame window and all its children will be deleted.

FRAMEWIN_AddMaxButton()



Description

Adds a maximize button to the title bar of the frame window.

Prototype

```
WM_HWIN FRAMEWIN_AddMaxButton(FRAMEWIN_Handle hObj,
```

```
int Flags, int Off);
```

Parameter	Meaning
hObj	Handle of frame window.
Flags	(see table below)
Off	X-offset used to create the BUTTON widget

Permitted values for parameter Index	
0	The BUTTON will be created at the left side.
FRAMEWIN_BUTTON_RIGHT	The BUTTON will be created at the right side.

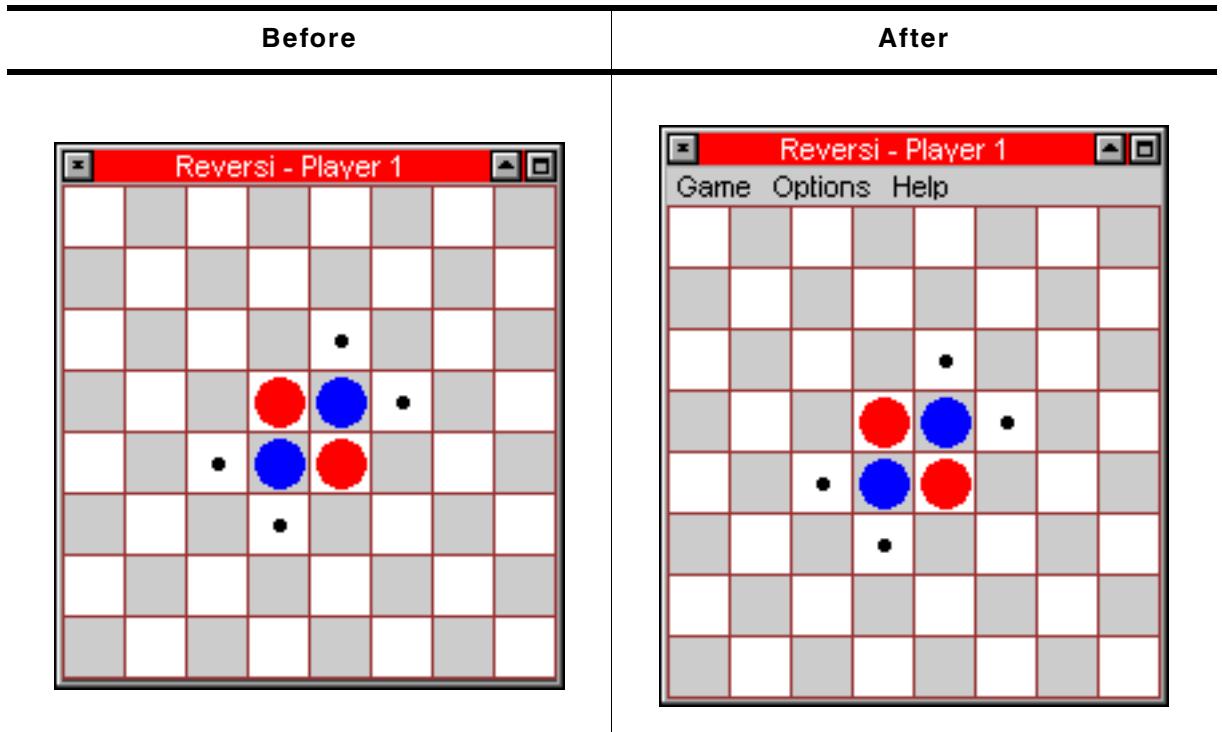
Return value

Handle of the maximize button.

Add. information

When the user presses the maximize button the first time the frame window 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()



Description

Adds the given menu to a frame window. The menu is shown below the title bar.

Prototype

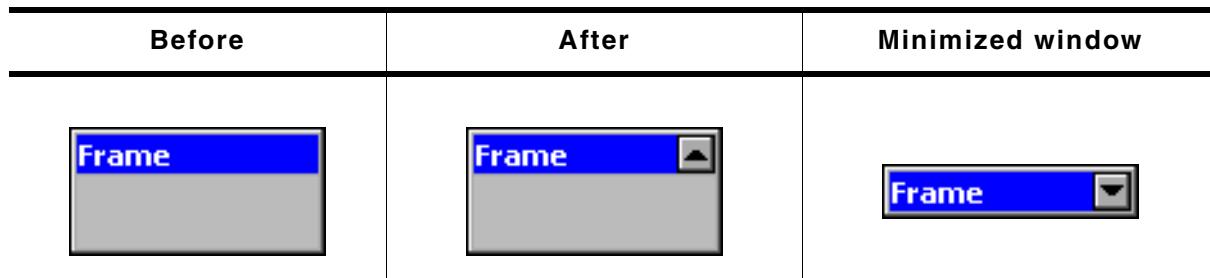
```
void FRAMEWIN_AddMenu(FRAMEWIN_Handle hObj, WM_HWIN hMenu);
```

Parameter	Meaning
hObj	Handle of frame window.
hMenu	Handle of menu widget to be added.

Add. information

The added menu is attached as a child of the frame window. If the frame window has been created with a callback routine, the function makes sure, that the WM_MENU messages are passed to the client window of the frame window.

FRAMEWIN_AddMinButton()



Description

Adds a minimize button to the title bar of the frame window.

Prototype

```
WM_HWIN FRAMEWIN_AddMinButton(FRAMEWIN_Handle hObj,
                                int Flags, int Off);
```

Parameter	Meaning
hObj	Handle of frame window.
Flags	(see table below)
Off	X-offset used to create the BUTTON widget

Permitted values for parameter Index	
0	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.

Add. information

When the user presses the minimize button the first time the client area of the frame window will be hidden and only the title bar remains visible. The second use of the button will restore the frame window 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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" 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).

Return value

Handle for the created FRAMEWIN widget; 0 if the routine 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	Meaning
x0	X-position of the frame window relative to the parent window.
y0	Y-position of the frame window relative to the parent window.
xsize	Horizontal size of the frame window (in pixels).
ysize	Vertical size of the frame window (in pixels).
hParent	Handle of parent window.
pText	Text to be displayed in the title bar.
cb	Pointer to callback routine.
Flags	Window create flags (see FRAMEWIN_Create()).

Return value

Handle for the created FRAMEWIN widget; 0 if the routine 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	Meaning
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).

Parameter	Meaning
<code>ysize</code>	Vertical size of the widget (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new FRAMEWIN widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	(see table below)
<code>Id</code>	Window ID of the widget.
<code>pTitle</code>	Title displayed in the title bar.
<code>cb</code>	Pointer to user callback routine. This user callback routine is called by the window callback routine of the client window.

Permitted values for parameter <code>ExFlags</code>	
0	No function.
FRAMEWIN_CF_MOVEABLE	Sets the new frame window to a moveable state. For details please refer to <code>FRAMEWIN_SetMoveable</code>

Return value

Handle for the created widget; 0 if the routine fails.

Add. 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 desireable 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_GetBorderSize()

Description

Returns the border size of the given frame window.

Prototype

```
int FRAMEWIN_GetBorderSize(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.

Return value

The border size of the given frame window.

FRAMEWIN_GetDefaultBarColor()

Description

Returns the default color for title bars in frame windows.

Prototype

```
const GUI_COLOR* FRAMEWIN_GetDefaultBarColor(unsigned int Index);
```

Parameter	Meaning
Index	Index for state of frame window (see table below).
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 frame windows in the specified state.

FRAMEWIN_GetDefaultBorderSize()

Description

Returns the default size of a frame window border.

Prototype

```
int FRAMEWIN_GetDefaultBorderSize(void);
```

Return value

Default size of a frame window border.

FRAMEWIN_GetDefaultClientColor()

Description

Returns the default color of client areas in frame windows.

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 frame window captions.

Prototype

```
const GUI_FONT* FRAMEWIN_GetDefaultFont(void);
```

Return value

Pointer to the default font used for frame window captions.

FRAMEWIN_GetDefaultTextColor()

Description

Returns the default text color of the title.

Prototype

```
GUI_COLOR FRAMEWIN_GetDefaultTextColor(unsigned Index);
```

Parameter	Meaning
Index	(see table below)

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_GetDefaultTitleHeight()

Description

Returns the default height of title bars in frame windows.

Prototype

```
int FRAMEWIN_GetDefaultCaptionSize(void);
```

Return value

Default title bar height. For more informations about the title height please take a look to FRAMEWIN_SetDefaultTitleHeight.

FRAMEWIN_GetTitleHeight()

Description

Returns the height of title bar of the given frame window.

Prototype

```
int FRAMEWIN_GetTitleHeight(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Return value

The height of title bar of the given frame window. For more informations about the title height please take a look to FRAMEWIN_SetDefaultTitleHeight.

FRAMEWIN_IsMaximized()

Description

Returns if the frame window is maximized or not.

Prototype

```
int FRAMEWIN_IsMaximized(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Return value

1 if the frame window is maximized, 0 if not.

FRAMEWIN_IsMinimized()

Description

Returns if the frame window is minimized or not.

Prototype

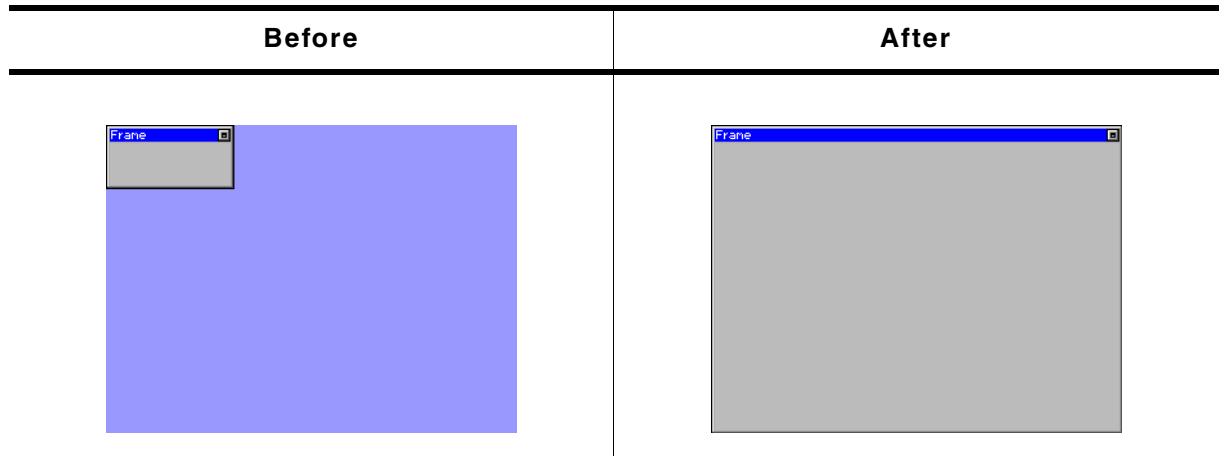
```
int FRAMEWIN_IsMinimized(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Return value

1 if the frame window is minimized, 0 if not.

FRAMEWIN_Maximize()



Description

Enlarges a frame window to the size of its parent window.

Prototype

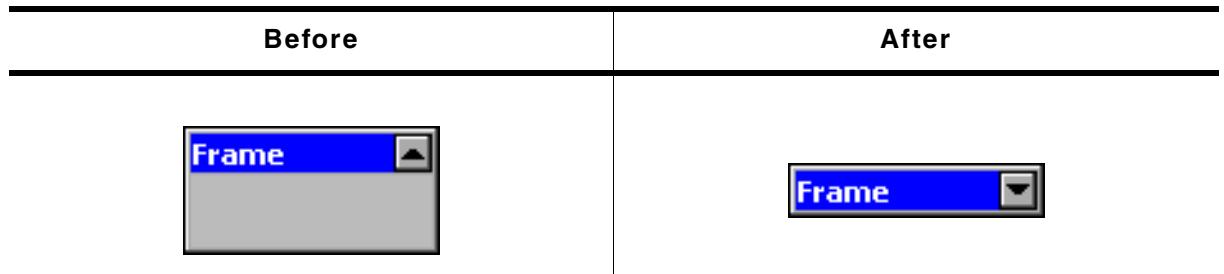
```
void FRAMEWIN_Maximize(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Add. information

When calling this function the frame window will show the same behavior as when the user presses the maximize button. The frame window will be enlarged to the size of its parent window.

FRAMEWIN_Minimize()



Description

Hides the client area of the given frame window.

Prototype

```
void FRAMEWIN_Minimize(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Add. information

When calling this function the frame window 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_Restore()

Before	After
	

Description

Restores a minimized or maximized frame window to its old size and position.

Prototype

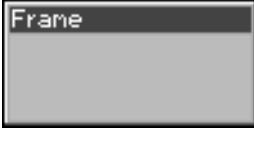
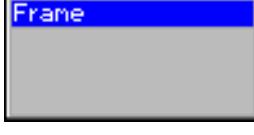
```
void FRAMEWIN_Restore(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Add. information

If the given frame window is neither maximized nor minimized the function takes no effect.

FRAMEWIN_SetActive()

Before	After
	

Description

Sets the state of a specified frame window. Depending on the state, the color of the title bar will change.

Prototype

```
void FRAMEWIN_SetActive(FRAMEWIN_Handle hObj, int State);
```

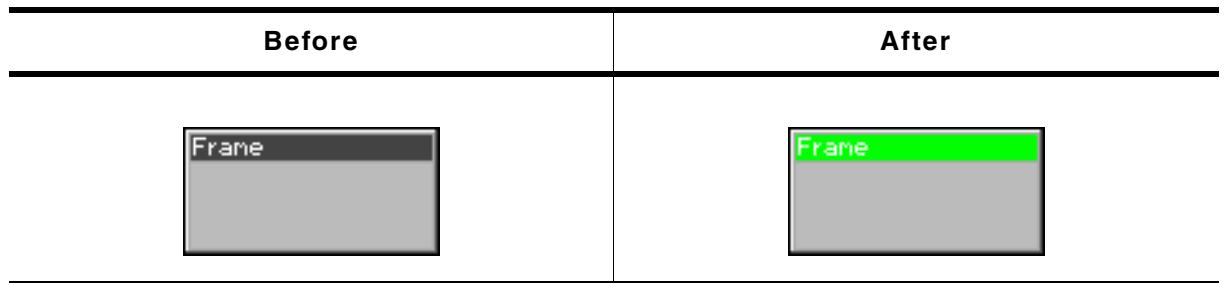
Parameter	Meaning
hObj	Handle of frame window.
State	State of frame window (see table below).

Permitted values for parameter State	
0	Frame window is inactive.
1	Frame window is active.

Add. information

This function is obsolete. If pointing with a input device to a child of a frame window the frame window will become active automatically. It is not recommended to use this function. If using this function to set a frame window to active state, it is not warranted that the state becomes inactive if an other frame window becomes active.

FRAMEWIN_SetBarColor()



Description

Sets the color of the title bar of a specified frame window.

Prototype

```
void FRAMEWIN_SetBarColor(FRAMEWIN_Handle hObj, unsigned int Index,
                           GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of frame window.
Index	Index for state of frame window (see table below).
Color	Color to be set.

Permitted values for parameter Index	
0	Sets the color to be used when frame window is inactive.
1	Sets the color to be used when frame window is active.

FRAMEWIN_SetBorderSize()

Before	After
	

Description

Sets the border size of a specified frame window.

Prototype

```
void FRAMEWIN_SetBorderSize(FRAMEWIN_Handle hObj, unsigned Size);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>Size</code>	New border size of the frame window.

FRAMEWIN_SetClientColor()

Before	After
	

Description

Sets the color of the client window area of a specified frame window.

Prototype

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>Color</code>	Color to be set.

FRAMEWIN_SetDefaultBarColor()

Description

Sets the default color for title bars in frame windows.

Prototype

```
void FRAMEWIN_SetDefaultBarColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
<code>Index</code>	Index for state of frame window (see table below).
<code>Color</code>	Color to be set.

Permitted values for parameter Index	
0	Sets the color to be used when frame window is inactive.
1	Sets the color to be used when frame window is active.

FRAMEWIN_SetDefaultBorderSize()

Description

Sets the default border size of frame windows.

Prototype

```
void FRAMEWIN_SetDefaultBorderSize(int BorderSize);
```

Parameter	Meaning
BorderSize	Size to be set.

FRAMEWIN_SetDefaultClientColor()

Description

Sets the default color for client areas in frame windows.

Prototype

```
void FRAMEWIN_SetDefaultClientColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Color to be set.

FRAMEWIN_SetDefaultFont()

Description

Sets the default font used to display the title in frame windows.

Prototype

```
void FRAMEWIN_SetDefaultFont(const GUI_FONT* pFont);
```

Parameter	Meaning
pFont	Pointer to font to be used as default

FRAMEWIN_SetDefaultTextColor()

Description

Sets the default text color of the title.

Prototype

```
void FRAMEWIN_SetDefaultTextColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	(see table below)
Color	Color to be used.

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

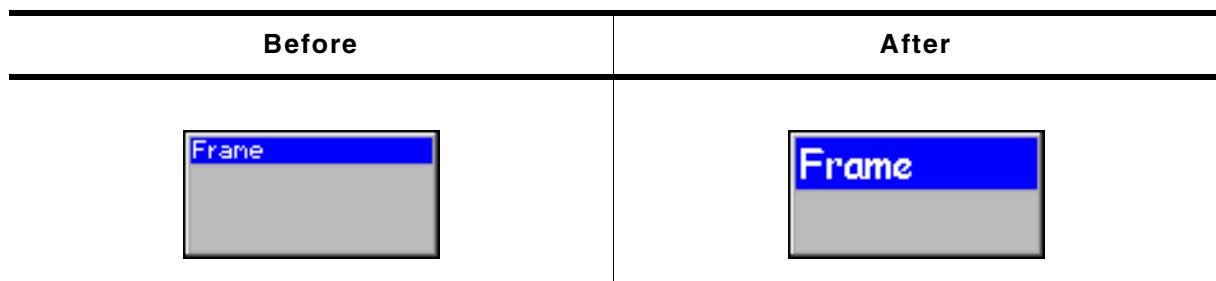
```
void FRAMEWIN_SetDefaultTitleHeight(int Height);
```

Parameter	Meaning
Height	Size to be set

Add. 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 frame window will use this height for the title height and not the height of the font of the title.

FRAMEWIN_SetFont()



Description

Sets the title font.

Prototype

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
hObj	Handle of frame window.
pFont	Pointer to the font.

FRAMEWIN_SetMoveable()

Description

Sets a frame window to a moveable or fixed state.

Prototype

```
void FRAMEWIN_SetMoveable(FRAMEWIN_Handle hObj, int State);
```

Parameter	Meaning
hObj	Handle of frame window.
State	State of frame window (see table below).

Permitted values for parameter State	
0	Frame window is fixed (non-moveable).
1	Frame window is moveable.

Add. information

The default state of a frame window after creation is fixed.

FRAMEWIN_SetResizeable()

Before	After
	

Description

Sets the resizable state of the given frame window.

Prototype

```
void FRAMEWIN_SetResizeable(FRAMEWIN_Handle hObj, int State);
```

Parameter	Meaning
hObj	Handle of frame window.
State	1 if the frame window should be resizable, 0 if not.

Add. information

If the frame window is resizable 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 pointing to the edge of the border, the X and Y size of the window can be changed simultaneously.

FRAMEWIN_SetText()

Before	After
	

Description

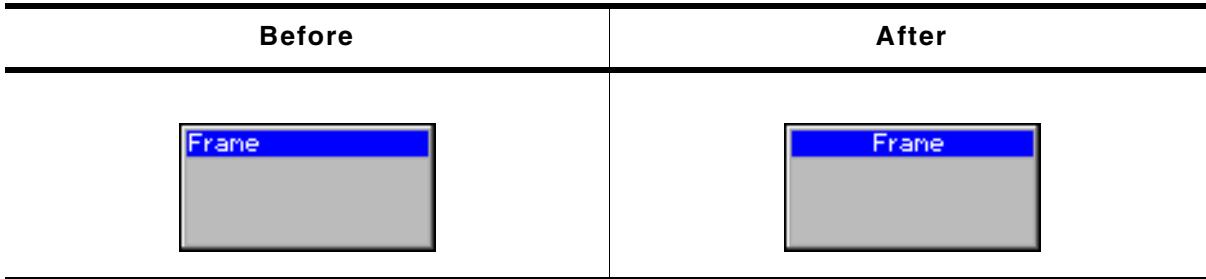
Sets the title text.

Prototype

```
void FRAMEWIN_SetText(FRAMEWIN_Handle hObj, const char* s);
```

Parameter	Meaning
hObj	Handle of frame window.
s	Text to display as the title.

FRAMEWIN_SetTextAlign()



Description

Sets the text alignment of the title bar.

Prototype

```
void FRAMEWIN_SetTextAlign(FRAMEWIN_Handle hObj, int Align);
```

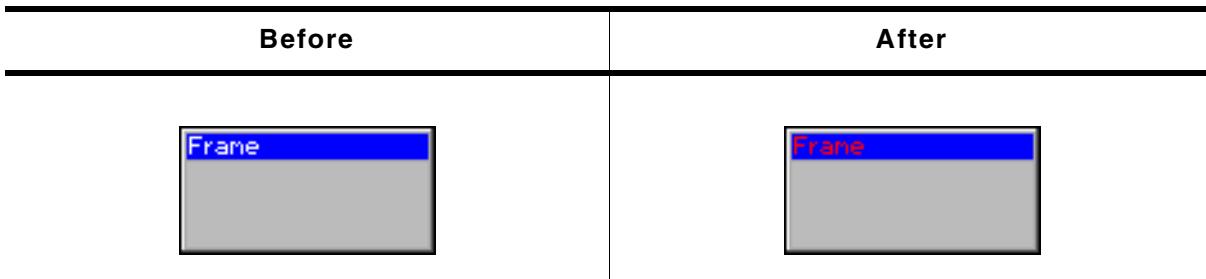
Parameter	Meaning
hObj	Handle of frame window.
Align	Alignment attribute for the title (see table below).

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.

Add. information

If this function is not called, the default behavior is to display the text centered.

FRAMEWIN_SetTextColor()



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	Meaning
hObj	Handle of frame window.
Color	Color to be set.

FRAMEWIN_SetTextColorEx()

Before	After
	

Description

Sets the text color for the given state.

Prototype

```
void FRAMEWIN_SetTextColorEx(FRAMEWIN_Handle hObj,
                             unsigned Index,
                             GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of frame window.
Index	(see table below)
Color	Color to be used.

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
	

Description

Sets the height of the title bar.

Prototype

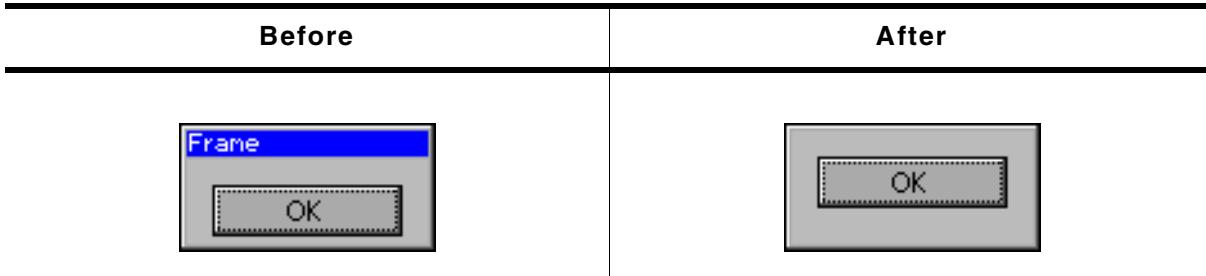
```
int FRAMEWIN_SetTitleHeight(FRAMEWIN_Handle hObj, int Height);
```

Parameter	Meaning
hObj	Handle of frame window.
Height	Height of the title bar.

Add. 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()



Description

Sets the visibility flag of the title bar.

Prototype

```
void FRAMEWIN_SetTitleVis(FRAMEWIN_Handle hObj, int Show);
```

Parameter	Meaning
hObj	Handle of frame window.
Show	1 for visible (default), 0 for hidden

15.8 GRAPH: Graph widget

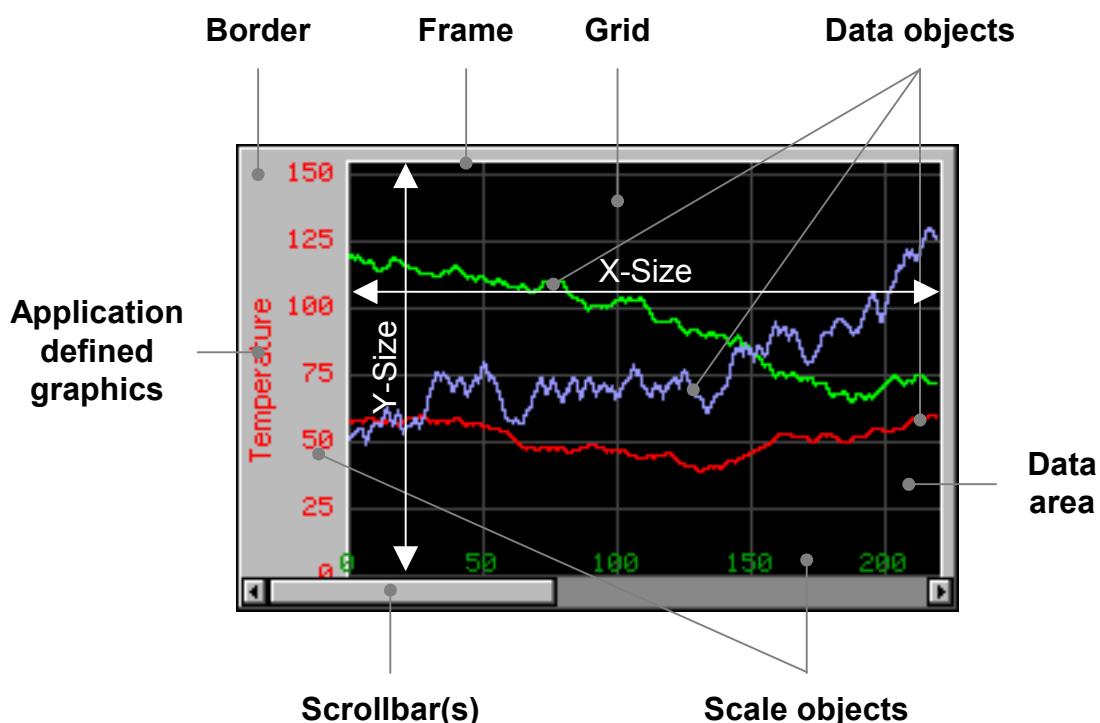
Graph widgets can be used to visualize data. Typical applications for graph widgets are showing measurement values or the curve of a function graph. Multiple curves can be shown simultaneously. Horizontal and vertical scales can be used to label the curves. A grid with different horizontal and vertical 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 scrollbars which allow scrolling through large data arrays.

15.8.1 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 scrollbar.
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.

15.8.2 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 sample 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);
```

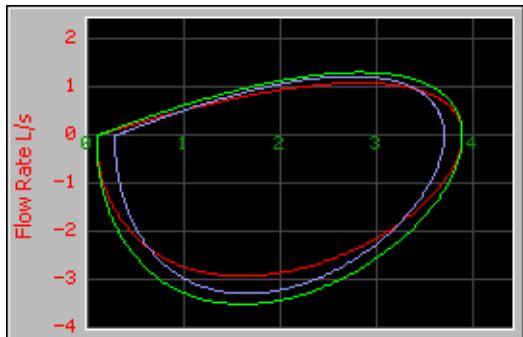
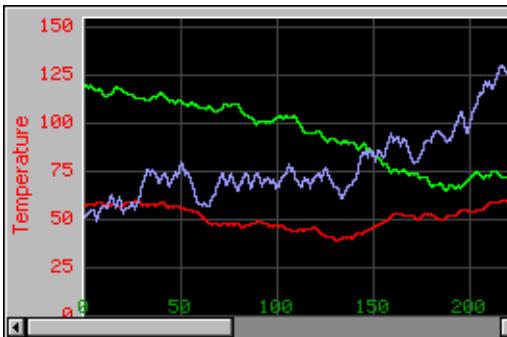
15.8.3 Drawing process

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.

15.8.4 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:

GRAPH_DATA_XY	GRAPH_DATA_YT
	

15.8.4.1 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.

15.8.4.2 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.

15.8.5 Configuration options

15.8.5.1 Graph widget

Type	Macro	Default	Explanation
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.

15.8.5.2 Scale object

Type	Macro	Default	Explanation
N	GRAPH_SCALE_TEXTCOLOR_DEFAULT	GUI_WHITE	Default text color.
S	GRAPH_SCALE_FONT_DEFAULT	&GUI_Font6x8	Default font used to draw the values.

15.8.6 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.8.7 GRAPH API

The table below lists the available emWin GRAPH-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
Common routines	
GRAPH_AttachData()	Attaches a data object to an existing graph widget.
GRAPH_AttachScale()	Attaches a scale object to an existing graph widget.
GRAPH_CreateEx()	Creates a graph widget.
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_SetGridVis()	Enables the drawing of a grid.

Routine	Explanation
<code>GRAPH_SetLineStyleH()</code>	Sets the line style for the horizontal grid lines.
<code>GRAPH_SetLineStyleV()</code>	Sets the line style for the vertical grid lines.
<code>GRAPH_SetVSizeX()</code>	Sets the horizontal range of the graph widget.
<code>GRAPH_SetVSizeY()</code>	Sets the vertical range of the graph widget.
<code>GRAPH_SetUserDraw()</code>	Sets the user callback function.
GRAPH_DATA_YT related routines	
<code>GRAPH_DATA_YT_AddValue()</code>	Adds one data item to the data object.
<code>GRAPH_DATA_YT_Create()</code>	Creates a GRAPH_DATA_YT object.
<code>GRAPH_DATA_YT_SetOffY()</code>	Sets a vertical offset for drawing the data.
GRAPH_DATA_XY related routines	
<code>GRAPH_DATA_XY_AddPoint()</code>	Adds one point to the data object.
<code>GRAPH_DATA_XY_Create()</code>	Creates a GRAPH_DATA_YT object.
<code>GRAPH_DATA_XY_SetOffX()</code>	Sets a horizontal offset for drawing the data.
<code>GRAPH_DATA_XY_SetOffY()</code>	Sets a vertical offset for drawing the data.
<code>GRAPH_DATA_XY_SetPenSize()</code>	Sets the pen size used to draw the data.
Scale related routines	
<code>GRAPH_SCALE_Create()</code>	Creates a scale object.
<code>GRAPH_SCALE_SetFactor()</code>	Sets a calculation factor used to calculate from pixels to the desired unit.
<code>GRAPH_SCALE_SetFixed()</code>	Avoids scrolling of the scale.
<code>GRAPH_SCALE_SetFont()</code>	Sets the font used to draw the numbers.
<code>GRAPH_SCALE_SetNumDecs()</code>	Sets the number of digits of the fractional portion.
<code>GRAPH_SCALE_SetOff()</code>	Sets an optional offset which is added to the numbers.
<code>GRAPH_SCALE_SetPos()</code>	Sets the horizontal or vertical position of the scale.
<code>GRAPH_SCALE_SetTextColor()</code>	Sets the text color of the scale.
<code>GRAPH_SCALE_SetTickDist()</code>	Sets the distance in pixels between the tick marks.

15.8.7.1 Common routines

GRAPH_AttachData()



Description

Attaches a data object to an existing graph widget.

Prototype

```
void GRAPH_AddGraph(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

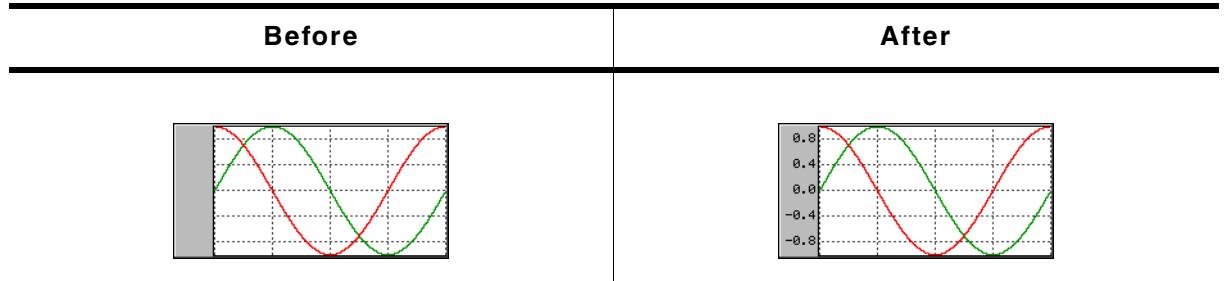
Parameter	Meaning
<code>hObj</code>	Handle of widget
<code>hData</code>	Handle of the data object to be added to the widget. The data object should be created with <code>GRAPH_DATA_YT_Create()</code> or <code>GRAPH_DATA_XY_Create()</code>

Add. 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 please refer to the functions `GRAPH_DATA_YT_Create()` and `GRAPH_DATA_XY_Create()`.

GRAPH_AttachScale()



Description

Attaches a scale object to an existing graph widget.

Prototype

```
void GRAPH_AttachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget
<code>hScale</code>	Handle of the scale to be added.

Add. 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 please refer to the function `GRAPH_SCALE_Create()`.

GRAPH_CreateEx()

Description

Creates a new graph widget.

Prototype

```
GRAPH_Handle GRAPH_CreateEx(int x0, int y0,
                           int xsize, int ysize, WM_HWIN hParent,
                           int WinFlags, int ExFlags, int Id);
```

Parameter	Meaning
<code>x0</code>	Leftmost pixel of the widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the widget (in pixels).
<code>ysize</code>	Vertical size of the widget (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new button window will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	(see table below)
<code>Id</code>	Window Id of the widget.

Permitted values for parameter <i>x</i>	
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 for the created graph widget; 0 if the routine fails.

GRAPH_SetBorder()

Before	After

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	Meaning
hObj	Handle of 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.

Add. 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 please refer to `GRAPH_SetColor()`.

GRAPH_SetColor()

Before	After

Description

Sets the desired color of the given graph widget.

Prototype

```
GUI_COLOR GRAPH_SetColor(GRAPH_Handle hObj, GUI_COLOR Color,
```

```
unsigned Index);
```

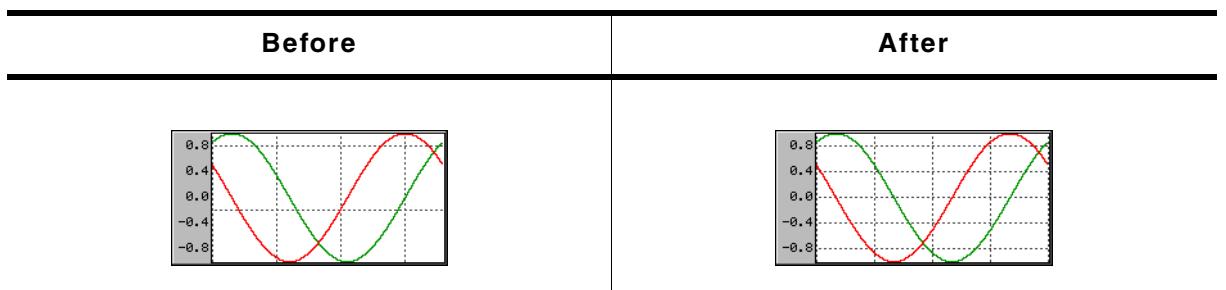
Parameter	Meaning
hObj	Handle of widget.
Color	Color to be used for the desired item.
Index	(see table below)

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()



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	Meaning
hObj	Handle of widget
Value	Distance in pixels from one grid line to the next, default is 50 pixel.

Return value

Previous grid line distance.

Add. 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	Meaning
<code>hObj</code>	Handle of widget.
<code>OnOff</code>	1 if grid should be fixed in X-axis, 0 if not (default).

Return value

Previous value used

Add. 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 please refer to the functions `GRAPH_SetVSizeX()` and `GRAPH_SetVSizeY()`.

GRAPH_SetGridOffY()



Description

Adds an offset used to show the horizontal grid lines.

Prototype

```
unsigned GRAPH_SetGridOffY(GRAPH_Handle hObj, unsigned Value);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>Value</code>	Offset to be used.

Return value

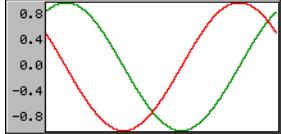
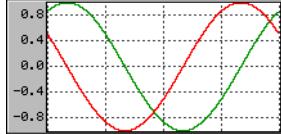
Previous offset used to draw the horizontal grid lines.

Add. 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 please refer to the functions `GRAPH_SetGridSpacingX()` and `GRAPH_SetGridSpacingY()`.

GRAPH_SetGridVis()

Before	After
	

Description

Sets the visibility of the grid lines.

Prototype

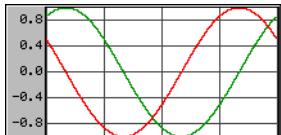
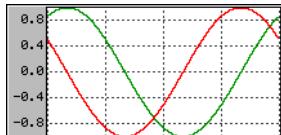
```
unsigned GRAPH_SetGridVis(GRAPH_Handle hObj, unsigned OnOff);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>OnOff</code>	1 if the grid should be visible, 0 if not (default).

Return value

Previous value of the grid visibility.

GRAPH_SetLineStyleH(), GRAPH_SetLineStyleV

Before	After
	

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	Meaning
<code>hObj</code>	Handle of widget.
<code>LineStyle</code>	Line style to be used. For details about the supported line styles please refer to the function <code>GUI_SetLineStyle()</code> . Default is <code>GUI_LS_SOLID</code> .

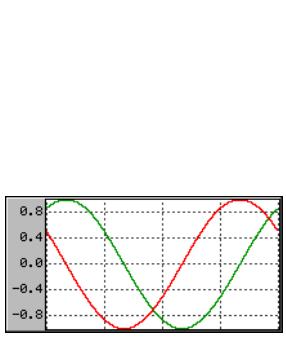
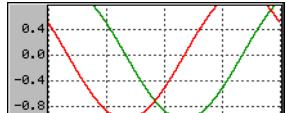
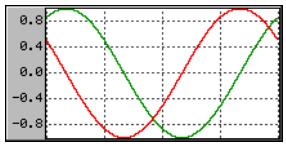
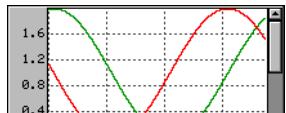
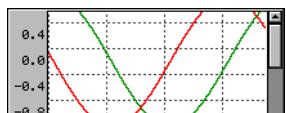
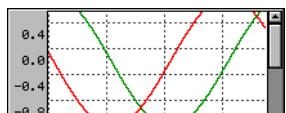
Return value

Previous line style used to draw the horizontal/vertical grid lines.

Add. Information

Please note that using other styles than `GUI_LS_SOLID` will need more time to show the grid.

GRAPH_SetVSizeX(), GRAPH_SetVSizeY()

Before	After
	
	
	

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	Meaning
hObj	Handle of widget.
Value	Virtual size in pixels in X or Y axis.

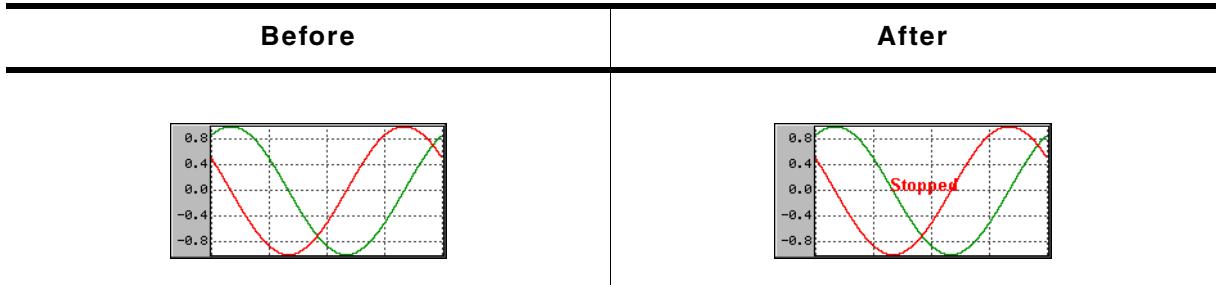
Return value

Previous virtual size of the widgets data area in X or Y-axis.

Add. Information

If the widgets virtual size is bigger than the visible size of the data area, the widget automatically shows a scrollbar. 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 scrollbar, provided that the number of data items is bigger than the X-size of the visible data area.

GRAPH_SetUserDraw()



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	Meaning
<code>hObj</code>	Handle of widget
<code>pUserDraw</code>	Pointer to application function to be called by the widget during the drawing process.

Permitted values for parameter <code>Stage</code>	
GRAPH_DRAW_FIRST	Function call after filling the background of the data area. Gives the application for example the possibility to draw a user defined grid.
GRAPH_DRAW_LAST	Function call after drawing all graph items. Gives the application for example the possibility to label the data with a user defined scale.

Add. 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.

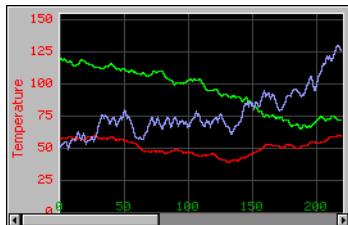
Example

The following small sample 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 */
}
...
```

15.8.7.2 GRAPH_DATA_YT related routines



GRAPH_DATA_YT_AddValue()

Before	After

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	Meaning
<code>hDataObj</code>	Handle of data object.
<code>Value</code>	Value to be added to the data object.

Add. 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.

GRAPH_DATA_YT_Create()

Description

Creates a data object of type GRAPH_DATA_YT.

Prototype

```
GRAPH_DATA_Handle GRAPH_DATA_YT_Create(GUI_COLOR Color,
                                         unsigned MaxNumItems,
                                         I16 * pData,
                                         unsigned NumItems);
```

Parameter	Meaning
<code>Color</code>	Color to be used to draw the data.
<code>MaxNumItems</code>	Maximum number of data items.
<code>pData</code>	Pointer to data to be added to the object. The pointer should point to an array of <code>I16</code> values.
<code>NumItems</code>	Number of data items to be added.

Return value

Handle of data object if creation was successful, otherwise 0.

Add. 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 scrollbar 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_SetOffY()`

Before	After

Description

Sets a vertical offset used to draw the object data.

Prototype

```
void GRAPH_DATA_YT_SetOffY(GRAPH_DATA_Handle hDataObj, int Off);
```

Parameter	Meaning
<code>hDataObj</code>	Handle of data object.
<code>Off</code>	Vertical offset which should be used to draw the data.

Add. Information

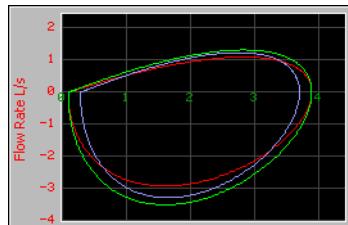
The vertical range of data, which is shown by the data object, is the range (0) - (Y-size of data area - 1). In case of using a scroll bar the current scroll position is added to the range.

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);
```

15.8.7.3 GRAPH_DATA_XY related routines



GRAPH_DATA_XY_AddPoint()

Before	After

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	Meaning
<code>hDataObj</code>	Handle of data object.
<code>pPoint</code>	Pointer to a GUI_POINT structure to be added to the data object.

Add. 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 data object of type GRAPH_DATA_XY.

Prototype

```
GRAPH_DATA_Handle GRAPH_DATA_XY_Create(GUI_COLOR Color,
                                         unsigned MaxNumItems,
                                         GUI_POINT * pData,
                                         unsigned NumItems);
```

Parameter	Meaning
<code>Color</code>	Color to be used to draw the data.
<code>MaxNumItems</code>	Maximum number of points.
<code>pData</code>	Pointer to data to be added to the object. The pointer should point to an array of GUI_POINT's.
<code>NumItems</code>	Number of points to be added.

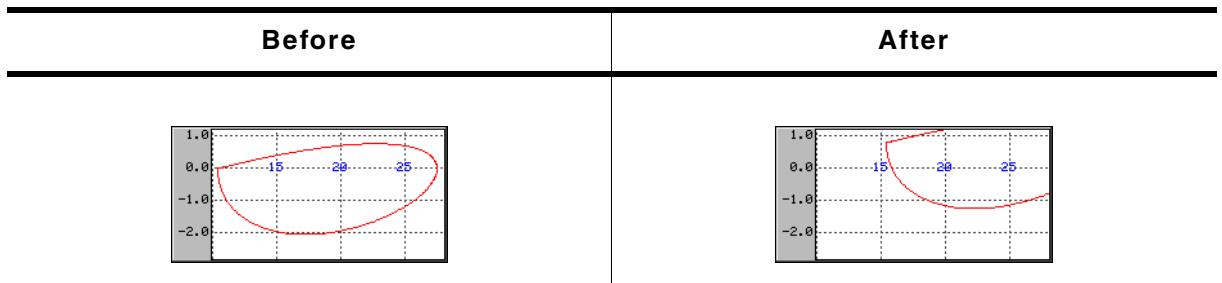
Return value

Handle of data object if creation was successful, otherwise 0.

Add. 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_SetOffX(), GRAPH_DATA_XY_SetOffY()



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	Meaning
<code>hDataObj</code>	Handle of data object.
<code>Off</code>	Horizontal/vertical offset which should be used to draw the polyline.

Add. 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 the following offsets need to be used:

```
GRAPH_DATA_XY_SetOffX(hDataObj, -100);
GRAPH_DATA_XY_SetOffY(hDataObj, 1200);
```

GRAPH_DATA_XY_SetPenSize()



Description

Sets the pen size used to draw the polyline.

Prototype

```
void GRAPH_DATA_XY_SetPenSize(GRAPH_DATA_Handle hDataObj, int PenSize);
```

Parameter	Meaning
hDataObj	Handle of data object.
PenSize	Pen size which should be used to draw the polyline.

15.8.7.4 Scale 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 scale object.

Prototype

```
GRAPH_SCALE_Handle GRAPH_SCALE_Create(int Pos, int Align,
                                       unsigned Flags, unsigned TickDist);
```

Parameter	Meaning
Pos	Position relative to the left/top edge of the graph widget.
TextAlign	Text alignment used to draw the numbers. For details please refer to the function GUI_SetTextAlign().
Flags	(see table below)
TickDist	Distance from one tick mark to the next.

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.

Add. Information

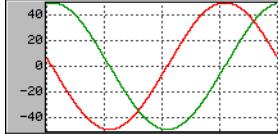
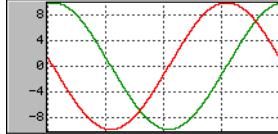
A horizontal scale object starts labeling from the bottom edge of the data area to the top and a vertical scale object from the left edge (horizontal scale) to the right, where the first position is the zero point. The parameter TickDist specifies the distance between the numbers.

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. Please 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_SetFactor()

Before	After
	

Description

Sets a factor used to calculate the numbers to be drawn.

Prototype

```
float GRAPH_SCALE_SetFactor(GRAPH_SCALE_Handle hScaleObj, float Factor);
```

Parameter	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>Factor</code>	Factor to be used to calculate the number.

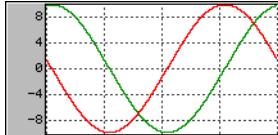
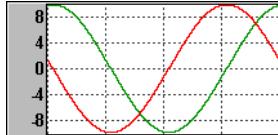
Return value

Old factor used to calculate the numbers.

Add. 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
	

Description

Sets the font used to draw the scale numbers.

Prototype

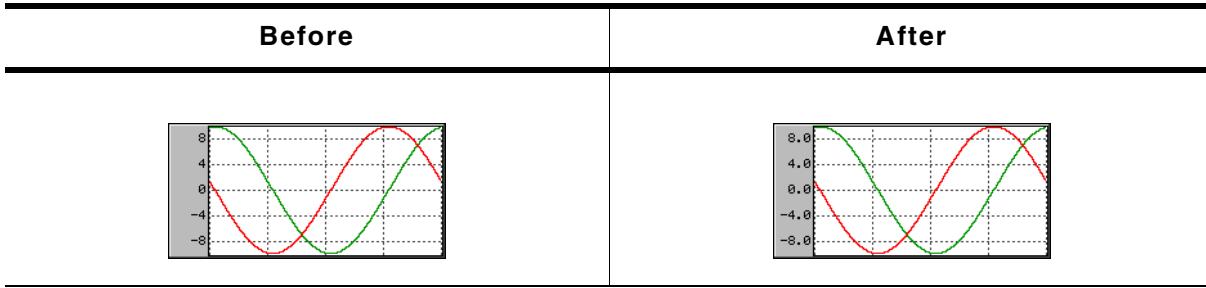
```
const GUI_FONT * GRAPH_SCALE_SetFont(GRAPH_SCALE_Handle hScaleObj,
                                      const GUI_FONT * pFont);
```

Parameter	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>pFont</code>	Font to be used.

Return value

Previous used font used to draw the numbers.

GRAPH_SCALE_SetNumDecs()



Description

Sets the number of post decimal positions to be shown.

Prototype

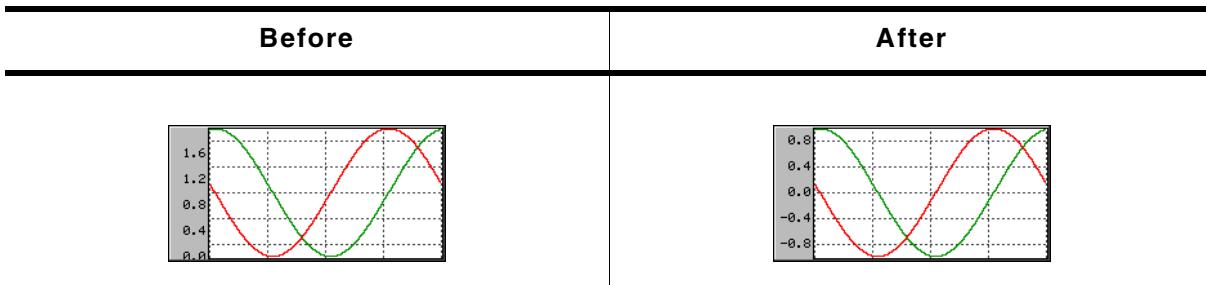
```
int GRAPH_SCALE_SetNumDecs(GRAPH_SCALE_Handle hScaleObj, int NumDecs);
```

Parameter	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>NumDecs</code>	Number of post decimal positions.

Return value

Previous number of post decimal positions.

GRAPH_SCALE_SetOff()



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	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>Off</code>	Offset used for drawing the scale.

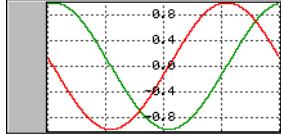
Return value

Previous used offset.

Add. 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
	

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	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>Pos</code>	Position, at which the scale should be shown.

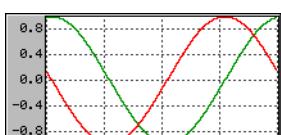
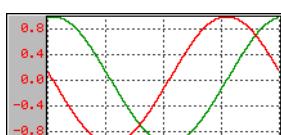
Return value

Previous position of the scale object.

Add. 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. Please note, that the actual text position also depends on the text alignment of the scale object.

GRAPH_SCALE_SetTextColor()

Before	After
	

Description

Sets the text color used to draw the numbers.

Prototype

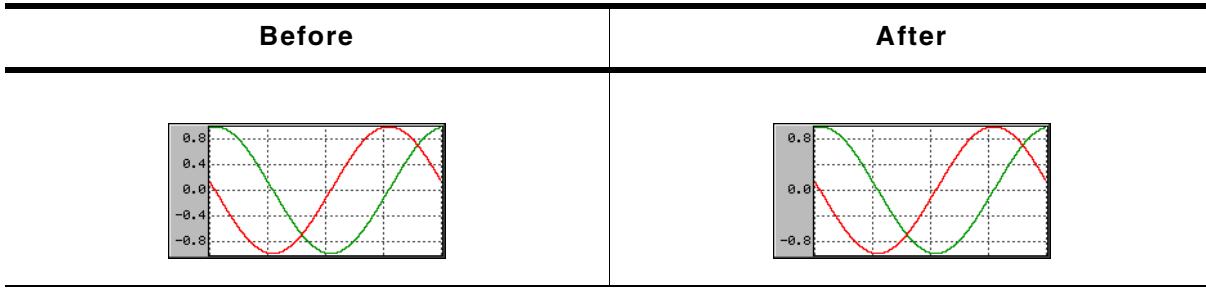
```
GUI_COLOR GRAPH_SCALE_SetTextColor(GRAPH_SCALE_Handle hScaleObj,
                                    GUI_COLOR Color);
```

Parameter	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>Color</code>	Color to be used to show the numbers.

Return value

Previous color used to show the numbers.

GRAPH_SCALE_SetTickDist()



Description

Sets the distance from one number to the next.

Prototype

```
unsigned GRAPH_SCALE_SetTickDist(GRAPH_SCALE_Handle hScaleObj,
                                  unsigned Dist);
```

Parameter	Meaning
<code>hScaleObj</code>	Handle of scale object.
<code>Dist</code>	Distance in pixels between the numbers.

Return value

Previous distance between the numbers.

15.9 HEADER: Header widget

HEADER widgets are used to label columns of a table:



When working with a touch screen or with mouse-support you can also manage the column width of a table. If you move the cursor nearby a divider the cursor will change:



There are 2 predefined cursors as shown below:

GUI_CursorHeaderM (default)	GUI_CursorHeaderMI

You can also create and use your own cursors when using a HEADER widget as described later in this chapter.

15.9.1 Configuration options

Type	Macro	Default	Explanation
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

15.9.2 Notification codes

The following events are sent from a HEADER widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off of the widget without releasing.

15.9.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.9.4 HEADER API

The table below lists the available emWin HEADER-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
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_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_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.
HEADER_SetBMPE()	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_SetFont()	Sets the font of the widget.
HEADER_SetHeight()	Sets the height of the widget.
HEADER_SetTextAlign()	Sets the alignment of the given item.
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_SetTextColor()	Sets the Text color of the widget.

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	Meaning
hObj	Handle of 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.

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.

Add. 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	Meaning
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 <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
SpecialFlags	(Reserved for later use)

Return value

Handle for the created HEADER widget; 0 if the routine 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	Meaning
<code>hObj</code>	Handle of widget
<code>Id</code>	Id of the HEADER widget
<code>SpecialFlags</code>	(Not used, reserved for later use)

Return value

Handle for the created HEADER widget; 0 if the routine fails.

Add. 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	Meaning
<code>x0</code>	Leftmost pixel of the widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the widget (in pixels).
<code>ysize</code>	Vertical size of the widget (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new HEADER widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	Not used, reserved for future use.
<code>Id</code>	Window ID of the widget.

Return value

Handle for the created widget; 0 if the routine fails.

HEADER_CreateIndirect()**Description**

Prototype explained at the beginning of the chapter.

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.

Add. 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	Meaning
hObj	Handle of widget

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	Meaning
hObj	Handle of widget
Index	Index of the item

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	Meaning
hObj	Handle of widget

Return value

Number of items of the given HEADER widget.

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	Meaning
hObj	Handle of widget
Index	Index of the item
pBitmap	Pointer to a bitmap structure to be displayed

Add. 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(HDR_HANDLE hObj,
                        unsigned int Index,
                        const GUI_BITMAP * pBitmap,
                        int x, int y);
```

Parameter	Meaning
hObj	Handle of 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

Add. information

One item of a HEADER widget can contain text and a bitmap.

Example:

```
...
HDR_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	Meaning
hObj	Handle of widget
Color	Background color to be set

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	Meaning
hObj	Handle of widget
Index	Index of HEADER item
pBitmap	Pointer to bitmap file data

Add. information

For additional informations regarding bitmap files please take a look at chapter 6.

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	Meaning
hObj	Handle of widget
Index	Index of the item
pBitmap	Pointer to bitmap file data
x	Additional offset in x
y	Additional offset in y

Add. information

For additional informations regarding bitmap files please take a look at chapter 6.

HEADER_SetDefaultBkColor()**Description**

Sets the default background color used when creating a HEADER widget.

Prototype

```
GUI_COLOR HEADER_SetDefaultBkColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Background color to be used

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	Meaning
Spacing	Value to be used

Return value

Previous default value.

Add. 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	Meaning
Spacing	Value to be used

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	Meaning
pCursor	Pointer to the cursor to be shown when dragging the width of an HEADER item

Return value

Pointer to the previous default cursor.

Add. 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	Meaning
pFont	Pointer to font to be used

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	Meaning
Color	Color to be used

Return value

Previous default value.

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	Meaning
hObj	Handle of widget
pFont	Pointer to font to be used

HEADER_SetHeight()**Description**

Sets the height of the given HEADER widget.

Prototype

```
void HEADER_SetHeight(HEADER_Handle hObj, int Height);
```

Parameter	Meaning
hObj	Handle of widget
Height	New height

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	Meaning
hObj	Handle of widget
Index	Index of header item
Align	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

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_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	Meaning
hObj	Handle of widget
Index	Index of HEADER item
s	Pointer to string to be displayed

Add. 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	Meaning
hObj	Handle of widget
Index	Index of HEADER item
Width	New width

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	Meaning
hObj	Handle of widget
Index	Index of the item
pBitmap	Pointer to streamed bitmap data to be displayed

Add. information

For additional informations regarding streamed bitmap files please take a look at chapter 6, "2-D Graphic Library".

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	Meaning
hObj	Handle of 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

Add. information

For additional informations regarding streamed bitmap files please take a look at chapter 6, "2-D Graphic Library".

HEADER_SetTextColor()**Description**

Sets the text color used when displaying the widget.

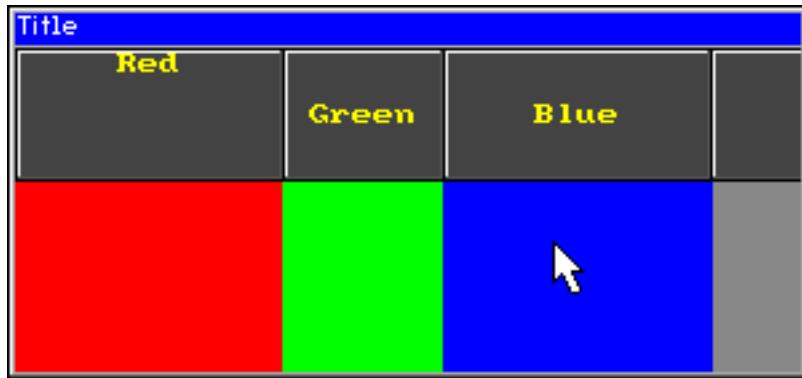
Prototype

```
void HEADER_SetTextColor(HEADER_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Color	Color to be used

15.9.5 Example

The source of the following sample is available as `WIDGET_Header.c` in the samples shipped with emWin:



15.10 LISTBOX: List box widget

List boxes 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 screen shots 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
	

All LISTBOX-related routines are in the file(s) LISTBOX*.c, LISTBOX.h. All identifiers are prefixed LISTBOX.

15.10.1 Configuration options

Type	Macro	Default	Explanation
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.

15.10.2 Notification codes

The following events are sent from a list box widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
WM_NOTIFICATION_CLICKED	List box has been clicked.
WM_NOTIFICATION_RELEASED	List box has been released.
WM_NOTIFICATION_MOVED_OUT	List box has been clicked and pointer has been moved out of the box without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scrollbar has been changed.
WM_NOTIFICATION_SEL_CHANGED	The selection of the list box has changed.

15.10.3 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 contents to the left.

Key	Reaction
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 contents to the right.
GUI_KEY_DOWN	Moves the selection bar down.
GUI_KEY_UP	Moves the selection bar up.

15.10.4 LISTBOX API

The table below lists the available emWin LISTBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>LISTBOX_AddString()</code>	Adds an item to a list box.
<code>LISTBOX_Create()</code>	Creates the list box. (Obsolete)
<code>LISTBOX_CreateAsChild()</code>	Creates the list box as a child window. (Obsolete)
<code>LISTBOX_CreateEx()</code>	Creates the list box.
<code>LISTBOX_CreateIndirect()</code>	Creates the list box from resource table entry.
<code>LISTBOX_DecSel()</code>	Decrements selection.
<code>LISTBOX_DeleteItem()</code>	Deletes an element.
<code>LISTBOX_GetDefaultBkColor()</code>	Returns the default background color for LISTBOX widgets.
<code>LISTBOX_GetDefaultFont()</code>	Returns the default font for LISTBOX widgets.
<code>LISTBOX_GetDefaultScrollStepH()</code>	Returns the default number of pixels to be scrolled horizontal.
<code>LISTBOX_GetDefaultTextAlign()</code>	Returns the default text alignment for new list boxes.
<code>LISTBOX_GetDefaultTextColor()</code>	Returns the default text color for new list boxes.
<code>LISTBOX_GetFont()</code>	Returns the font of the list box.
<code>LISTBOX_GetItemDisabled()</code>	Returns the disabled state of the given item.
<code>LISTBOX_GetItemSel()</code>	Returns the selection state of a LISTBOX entry.
<code>LISTBOX_GetItemText()</code>	Returns the text of a list box entry.
<code>LISTBOX_GetMulti()</code>	Returns if the multi select mode is active.
<code>LISTBOX_GetNumItems()</code>	Returns the number of items in a list box.
<code>LISTBOX_GetScrollStepH()</code>	Returns the number of pixels to be scrolled horizontal.
<code>LISTBOX_GetSel()</code>	Returns the number of the selected item.
<code>LISTBOX_GetTextAlign()</code>	Returns the text alignment of the LISTBOX.
<code>LISTBOX_IncSel()</code>	Increments selection.
<code>LISTBOX_InsertString()</code>	Inserts an element.
<code>LISTBOX_InvalidateItem()</code>	Invalidates an item of an owner drawn LISTBOX.
<code>LISTBOX_OwnerDraw()</code>	Default function for drawing a LISTBOX entry.
<code>LISTBOX_SetAutoScrollH()</code>	Activates automatic use of a horizontal scrollbar.
<code>LISTBOX_SetAutoScrollV()</code>	Activates automatic use of a vertical scrollbar.
<code>LISTBOX_SetBkColor()</code>	Sets the background color.
<code>LISTBOX_SetDefaultBkColor()</code>	Sets the default background color for LISTBOX widgets.
<code>LISTBOX_SetDefaultFont()</code>	Changes the default font for LISTBOX widgets.
<code>LISTBOX_SetDefaultScrollStepH()</code>	Sets the default number of pixels to be scrolled horizontal.
<code>LISTBOX_SetDefaultTextAlign()</code>	Sets the default text alignment for new LISTBOX widgets.
<code>LISTBOX_SetDefaultTextColor()</code>	Sets the default text color for LISTBOX widgets.
<code>LISTBOX_SetFont()</code>	Selects the font.
<code>LISTBOX_SetItemDisabled()</code>	Sets the disabled state of the given item.
<code>LISTBOX_SetItemSel()</code>	Sets the selection state of the given item.
<code>LISTBOX_SetItemSpacing()</code>	Sets a spacing between the items.
<code>LISTBOX_SetMulti()</code>	Sets the multi selection mode on or off.
<code>LISTBOX_SetOwnerDraw()</code>	Enables the list box to be owner drawn.
<code>LISTBOX_SetScrollbarColor()</code>	Sets the colors of the optional scrollbar.
<code>LISTBOX_SetScrollbarWidth()</code>	Sets the width of the scrollbars used by the LISTBOX.

Routine	Explanation
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_AddString()

Description

Adds an item to an already existing list box.

Prototype

```
void LISTBOX_AddString(LISTBOX_Handle hObj, const char* s);
```

Parameter	Meaning
hObj	Handle of list box.
s	Text to display.

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	Meaning
ppText	Pointer to an array of string pointers containing the elements to be displayed.
x0	Leftmost pixel of the list box (in parent coordinates).
y0	Topmost pixel of the list box (in parent coordinates).
xSize	Horizontal size of the list box (in pixels).
ySize	Vertical size of the list box (in pixels).
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).

Return value

Handle for the created FRAMEWIN widget, 0 on failure.

Add. information

If the parameter [ySize](#) is greater than the required space for drawing the contents 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 hParent, int x0, int y0,
```

```
int xSize, int ySize, int Flags);
```

Parameter	Meaning
<code>ppText</code>	Pointer to an array of string pointers containing the elements to be displayed.
<code>hParent</code>	Handle of parent window.
<code>x0</code>	X-position of the list box relative to the parent window.
<code>y0</code>	Y-position of the list box relative to the parent window.
<code>xSize</code>	Horizontal size of the list box (in pixels).
<code>ySize</code>	Vertical size of the list box (in pixels).
<code>Flags</code>	Window create flags (see <code>LISTBOX_Create()</code>).

Return value

Handle for the created LISTBOX widget; 0 if the routine fails.

Add. information

If the parameter `ySize` is greater than the space required for drawing the contents 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	Meaning
<code>x0</code>	Leftmost pixel of the widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the widget (in pixels).
<code>ysize</code>	Vertical size of the widget (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new HEADER widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	Not used, reserved for future use.
<code>Id</code>	Window ID of the widget.
<code>ppText</code>	Pointer to an array of string pointers containing the elements to be displayed.

Return value

Handle for the created widget; 0 if the routine fails.

LISTBOX_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

LISTBOX_DecSel()

Description

Decrement the list box selection (moves the selection bar of a specified list box up by one item).

Prototypes

```
void LISTBOX_DecSel(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Add. information

Please 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.

Prototypes

```
void LISTBOX_DeleteItem(LISTBOX_Handle hObj, unsigned int Index);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of element to be deleted.

LISTBOX_GetDefaultBkColor()

Description

Returns the default background color for new LISTBOX widgets.

Prototype

```
GUI_COLOR LISTBOX_GetDefaultBkColor(unsigned Index);
```

Parameter	Meaning
Index	Zero based index for background color (see table below)

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_GetDefault TextAlign()**Description**

Returns the default text alignment for new LISTBOX widgets.

Prototype

```
int LISTBOX_GetDefault TextAlign(void);
```

Return value

Default text alignment for new LISTBOX widgets.

Add. information

For more information please refer to `LISTBOX_Set TextAlign()`.

LISTBOX_GetDefaultTextColor()**Description**

Returns the default text color for new LISTBOX widgets.

Prototype

```
GUI_COLOR LISTBOX_GetDefaultTextColor(unsigned Index);
```

Parameter	Meaning
<code>Index</code>	Zero based index for text color (see table below)

Permitted values for parameter <code>Index</code>	
<code>LISTBOX_CI_UNSEL</code>	Unselected element.
<code>LISTBOX_CI_SEL</code>	Selected element, without focus.
<code>LISTBOX_CI_SELFOCUS</code>	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 list box.

Prototype

```
const GUI_FONT * LISTBOX_GetFont(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Return value

Pointer to the font used to display the text of the list box.

LISTBOX_GetItemDisabled()**Description**

Returns if the given list box item has been disabled.

Prototype

```
int LISTBOX_GetItemDisabled(LISTBOX_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of item.

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	Meaning
hObj	Handle of list box.
Index	Zero based index of item.

Return value

1 if item has been selected, 0 if not.

LISTBOX_GetItemText()**Description**

Returns the text of the given list box item.

Prototype

```
void LISTBOX_GetItemText(LISTBOX_Handle hObj,
                        unsigned Index, char * pBuffer, int MaxSize);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of item.
pBuffer	Pointer to buffer to store the item text.
MaxSize	Size of the buffer.

Add. information

The function copies the text of the given list box item into the given buffer.

LISTBOX_GetMulti()

Description

Returns if the multi selection mode of the given list box is active.

Prototype

```
int LISTBOX_GetMulti(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Return value

1 if active, 0 if not.

LISTBOX_GetNumItems()

Description

Returns the number of items in a specified list box.

Prototypes

```
unsigned LISTBOX_GetNumItems(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Return value

Number of items in the list box.

LISTBOX_GetScrollStepH()

Description

Returns the horizontal scroll step of the given list box.

Prototype

```
int LISTBOX_GetScrollStepH(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Return value

Horizontal scroll step of the given list box.

LISTBOX_GetSel()

Description

Returns the zero based index of the currently selected item in a specified list box. In multi selection mode the function returns the index of the focused element.

Prototype

```
int LISTBOX_GetSel(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Return value

Zero based index of the currently selected item.

Add. 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	Meaning
hObj	Handle of widget.

Return value

Text alignment of the given LISTBOX widget.

Add. information

For more information please refer to `LISTBOX_SetTextAlign()`.

LISTBOX_IncSel()**Description**

Increment the list box selection (moves the selection bar of a specified list box down by one item).

Prototypes

```
void LISTBOX_IncSel(LISTBOX_Handle hObj);
```

Parameter	Meaning
hObj	Handle of list box.

Add. information

Please 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.

Prototypes

```
void LISTBOX_InsertString(LISTBOX_Handle hObj, const char* s,
                           unsigned int Index);
```

Parameter	Meaning
hObj	Handle of list box.
s	Pointer to string to be inserted.
Index	Zero based index of element to be inserted.

LISTBOX_InvalidateItem()**Description**

Invalidates an item of a owner drawn listbox.

Prototypes

```
void LISTBOX_InvalidateItem(LISTBOX_Handle hObj, int Index);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of element to be invalidated or LISTBOX_ALL_ITEMS if all items should be invalidated.

Add. information

This function only needs to be called if an item of an owner drawn listbox 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	Meaning
pDrawItemInfo	Pointer to a WIDGET_ITEM_DRAW_INFO structure.

Add. information

This function is usefull if LISTBOX_SetOwnerDrawn 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 sample.

LISTBOX_SetAutoScrollH()

Description

Enables/disables the automatic use of a horizontal scrollbar.

Prototypes

```
void LISTBOX_SetAutoScrollH(LISTBOX_Handle hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of list box.
OnOff	(See table below)

Permitted values for parameter OnOff	
0	Disable automatic use of a horizontal scrollbar.
1	Enable automatic use of a horizontal scrollbar.

Add. information

If enabled the listbox checks if all elements fits into the listbox. If not a horizontal scrollbar will be attached to the window.

LISTBOX_SetAutoScrollV()

Description

Enables/disables the automatic use of a vertical scrollbar.

Prototypes

```
void LISTBOX_SetAutoScrollV(LISTBOX_Handle hObj, int OnOff);
```

Parameter	Meaning
<code>hObj</code>	Handle of list box.
<code>OnOff</code>	(See table below)

Permitted values for parameter <code>OnOff</code>	
0	Disable automatic use of a vertical scrollbar.
1	Enable automatic use of a vertical scrollbar.

Add. information

If enabled the listbox checks if all elements fits into the listbox. If not a vertical scrollbar will be added.

LISTBOX_SetBkColor()

Description

Sets the list box background color.

Prototype

```
void LISTBOX_SetBkColor(LISTBOX_Handle hObj, unsigned int Index,
                        GUI_COLOR Color);
```

Parameter	Meaning
<code>hObj</code>	Handle of list box.
<code>Index</code>	Index for background color (see table below).
<code>Color</code>	Color to be set.

Permitted values for parameter <code>Index</code>	
<code>LISTBOX_CI_UNSEL</code>	Unselected element.
<code>LISTBOX_CI_SEL</code>	Selected element, without focus.
<code>LISTBOX_CI_SELFOCUS</code>	Selected element, with focus.
<code>LISTBOX_CI_DISABLED</code>	Disabled element.

LISTBOX_SetDefaultBkColor()

Description

Sets the default background color for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultBkColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Meaning
<code>Index</code>	Zero based index for background color (see table below)
<code>Color</code>	Desired background color.

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	Meaning
pFont	Pointer to the font.

LISTBOX_SetDefaultScrollStepH()

Description

Sets the default horizontal scroll step used when creating a LISTBOX widget.

Prototype

```
void LISTBOX_SetDefaultScrollStepH(int Value);
```

Parameter	Meaning
Value	Number of pixels to be scrolled.

LISTBOX_SetDefault TextAlign()

Description

Sets the default text alignment for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefault TextAlign(int Align);
```

Parameter	Meaning
Align	Default text alignment for new LISTBOX widgets.

Add. information

For more information please refer to [LISTBOX_SetTextAlign\(\)](#).

LISTBOX_SetDefaultTextColor()

Description

Sets the default text color for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultTextColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Zero based index for text color (see table below)
Color	Desired text color.

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELF_FOCUS	Selected element, with focus.

LISTBOX_SetFont()

Description

Sets the list box font.

Prototype

```
void LISTBOX_SetFont(LISTBOX_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
hObj	Handle of list box.
pFont	Pointer to the font.

LISTBOX_SetItemDisabled()

Description

Modifies the disable state of the given list box item.

Prototype

```
void LISTBOX_SetItemDisabled(LISTBOX_Handle hObj,
                             unsigned Index,
                             int OnOff);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of the listbox item.
OnOff	1 for disabled, 0 for not disabled.

Add. information

When scrolling through a list box disabled items will be skipped. You can not scroll to a disabled list box item.

LISTBOX_SetItemSel()

Description

Modifies the selection state of the given list box item.

Prototype

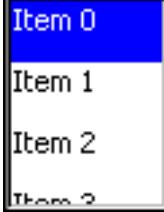
```
void LISTBOX_SetItemSel(LISTBOX_Handle hObj, unsigned Index, int OnOff);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of the listbox item.
OnOff	1 for selected, 0 for not selected.

Add. information

Setting the selection state of a list box item makes only sense when using the multi selection mode. See also [LISTBOX_SetMulti\(\)](#).

LISTBOX_SetItemSpacing()

Before	After
	

Description

Sets an additional spacing below the items of a list box.

Prototype

```
void LISTBOX_SetItemSpacing(LISTBOX_Handle hObj, unsigned Value);
```

Parameter	Meaning
hObj	Handle of list box.
Value	Number of pixels used as additional spacing between the items.

LISTBOX_SetMulti()

Description

Switches the multi selection mode of a LISTBOX on or off.

Prototype

```
void LISTBOX_SetMulti(LISTBOX_Handle hObj, int Mode);
```

Parameter	Meaning
hObj	Handle of list box.
Mode	0 for off, 1 for on.

Add. information

The multi selection mode enables the list box to have more than one selected element. Using the space key would toggle the selection state of a list box item.

LISTBOX_SetOwnerDraw()

Description

Sets the list box to be owner drawn.

Prototype

```
void LISTBOX_SetOwnerDraw(LISTBOX_Handle hObj,
                         WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

Parameter	Meaning
hObj	Handle of list box.
pfDrawItem	Pointer to owner draw function.

Add. information

This function sets a function pointer to a function which will be called by the widget if a list box 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 list box 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 samples as WIDGET_ListBoxOwnerDraw.

LISTBOX_SetScrollbarColor()

Before	After
	

Description

Sets the colors of the optional scrollbar.

Prototype

```
void LISTBOX_SetScrollbarColor(LISTBOX_Handle hObj,
                               unsigned int Index,
```

```
GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index of desired item (see table below).
Color	Color to be used.

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 scrollbars used by the given list box.

Prototype

```
void LISTBOX_SetScrollbarWidth(LISTBOX_Handle hObj, unsigned Width);
```

Parameter	Meaning
hObj	Handle of list box.
Width	Width of the scrollbar(s) used by the given listbox.

LISTBOX_SetScrollStepH()

Description

Sets the horizontal scroll step of the given list box. The horizontal scroll step defines the number of pixels to be scrolled if needed.

Prototype

```
void LISTBOX_SetScrollStepH(LISTBOX_Handle hObj, int Value);
```

Parameter	Meaning
hObj	Handle of list box.
Value	Number of pixels to be scrolled.

LISTBOX_SetSel()

Description

Sets the selected item of a specified list box.

Prototype

```
void LISTBOX_SetSel(LISTBOX_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of list box.
Sel	Element to be selected.

LISTBOX_SetString()

Description

Sets the contents of the given item.

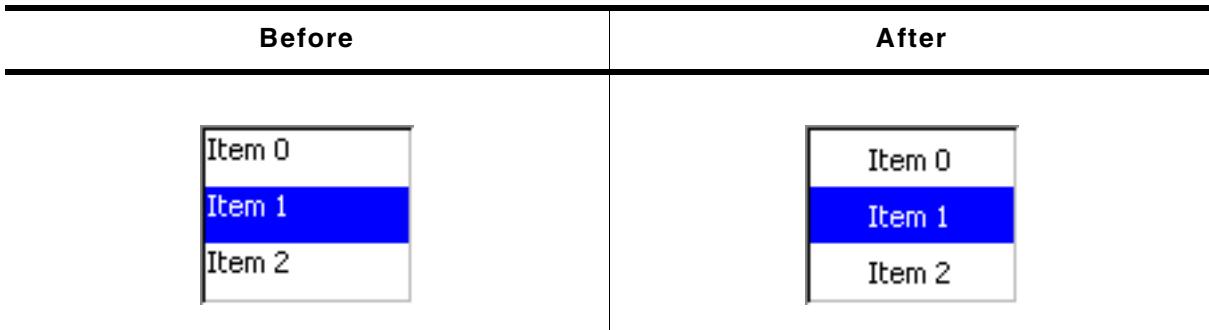
Prototypes

```
void LISTBOX_SetString(LISTBOX_Handle hObj, const char* s,
```

```
unsigned int Index);
```

Parameter	Meaning
hObj	Handle of list box.
s	Pointer to string containing the new contents.
Index	Zero based index of element to be changed.

LISTBOX_SetTextAlign()



Description

The function sets the text alignment used to display each item of the list box.

Prototype

```
void LISTBOX_SetTextAlign(LISTBOX_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of widget.
Align	Text alignment to be used.

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.

Add. 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 list box 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 list box text color.

Prototype

```
void LISTBOX_SetTextColor(LISTBOX_Handle hObj, unsigned int Index,
```

```
    GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Index for text color (see LISTBOX_SetBackColor()).
Color	Color to be set.

15.10.5 Examples

Using the LISTBOX widget

The following example demonstrates the simple use of the LISTBOX widget. It is available in the samples as WIDGET_SimpleListBox.c:

```
/*
 *      SEGGER MICROCONTROLLER SYSTEME GmbH
 *      Solutions for real time microcontroller applications
 */
/*
 *      (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
 *
 *      Internet: www.segger.com      Support: support@segger.com
 */
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
distributed in any way. We appreciate your understanding and fairness.
-----
File      : WIDGET_SimpleListBox.c
Purpose   : Example demonstrating the LISTBOX widget
-----
*/
#include "GUI.h"
#include "LISTBOX.h"
#include <stddef.h>

/*
 *      defines
 */
***** static variables *****
*/
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

/*
 *      static code
 */
***** _DemoListBox *****
*/
void _DemoListBox(void) {
    int i;
    int Entries = countof(_ListBox) - 1;
    LISTBOX_Handle hListBox;
```

```

int ySize = GUI_GetYDistOffFont(&GUI_Font13B_1) * Entries;
/* Create the listbox */
hListBox = LISTBOX_Create(_ListBox, 130, 80, 60, ySize, WM_CF_SHOW);
/* Change current selection of the listbox */
for (i = 0; i < Entries-1; i++) {
    GUI_Delay(500);
    LISTBOX_IncSel(hListBox);
    WM_ExecIdle();
}
for (i = 0; i < Entries-1; i++) {
    GUI_Delay(500);
    LISTBOX_DecSel(hListBox);
    WM_ExecIdle();
}
GUI_Delay(750);
/* Delete listbox widget */
LISTBOX_Delete(hListBox);
GUI_ClearRect(0, 50, 319, 239);
GUI_Delay(750);
}

*****
*
*      MainTask
*
*      Demonstrates LISTBOX widget
*
*****
*/
void MainTask(void) {
    GUI_Init();
    GUI_SetBkColor(GUI_BLUE);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("WIDGET_SimpleListBox - Sample", 160, 5);
    while(1) {
        _DemoListBox();
    }
}

```

Screen shot of above example



Using the LISTBOX widget with and without frames

The following example illustrates the use of the LISTBOX widget with and without a frame window. It is available as `WIDGET_ListBox.c`:

```

*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*****
*
*          (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
*
*          Internet: www.segger.com      Support: support@segger.com
*
*****
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
distributed in any way. We appreciate your understanding and fairness.
-----
File       : WIDGET_ListBox.c
Purpose   : Example demonstrating the LISTBOX widget
-----
*/

```

```

#include "GUI.h"
#include "FRAMEWIN.h"
#include "LISTBOX.h"
#include <stddef.h>

#define SPEED 1000

/*****************
*      static variables
*
*****static code
*/
/*
*      _ShowSeveralFunctions
*/
static void _ShowSeveralFunctions(LISTBOX_Handle hListBox) {
    int NumEntries, i;
    /* Add scrollbar */
    GUI_DispStringAtCEOL("SCROLLBAR_CreateAttached", 5, 55);
    GUI_Delay(SPEED);
    SCROLLBAR_CreateAttached(hListBox, SCROLLBAR_CF_VERTICAL);
    GUI_Delay(SPEED * 0.75);
    /* Add strings */
    GUI_DispStringAtCEOL("LISTBOX_AddString", 5, 55);
    GUI_Delay(SPEED);
    LISTBOX_AddString(hListBox, "Français");
    GUI_Delay(SPEED / 6);
    LISTBOX_AddString(hListBox, "Japanese");
    GUI_Delay(SPEED / 6);
    LISTBOX_AddString(hListBox, "Italiano");
    GUI_Delay(SPEED / 6);
    LISTBOX_AddString(hListBox, "Español");
    GUI_Delay(SPEED / 6);
    LISTBOX_AddString(hListBox, "Other language ...");
    GUI_Delay(SPEED * 0.6);
    /* Set focus */
    GUI_DispStringAtCEOL("WM_SetFocus", 5, 55);
    GUI_Delay(SPEED * 0.9);
    WM_SetFocus(hListBox);
    GUI_Delay(SPEED * 0.7);
    /* Set font */
    GUI_DispStringAtCEOL("LISTBOX_SetFont", 5, 55);
    GUI_Delay(SPEED * 0.9);
    LISTBOX_SetFont(hListBox, &GUI_Font13B_1);
    GUI_Delay(SPEED * 0.7);
    /* Increment selection */
    GUI_DispStringAtCEOL("LISTBOX_IncSel", 5, 55);
    GUI_Delay(SPEED);
    NumEntries = LISTBOX_GetNumItems(hListBox);
    for (i = 0; i < NumEntries - 1; i++) {
        LISTBOX_IncSel(hListBox);
        GUI_Delay(SPEED / 6);
    }
    GUI_Delay(SPEED / 4);
    /* Show automatic scrollbar */
    GUI_DispStringAtCEOL("Optional automatic scrollbar", 5, 55);
    GUI_Delay(SPEED);
    LISTBOX_SetAutoScrollH(hListBox, 1);
    LISTBOX_SetAutoScrollV(hListBox, 1);
    GUI_Delay(SPEED * 0.75);
    /* Set font */
    GUI_DispStringAtCEOL("LISTBOX_SetFont", 5, 55);
    GUI_Delay(SPEED);
    LISTBOX_SetFont(hListBox, &GUI_Font16B_1);
    GUI_Delay(SPEED * 0.75);
    /* Decrement selection */
    GUI_DispStringAtCEOL("LISTBOX_DecSel", 5, 55);
    GUI_Delay(SPEED);
}

```

```

for (i = 0; i < NumEntries - 1; i++) {
    LISTBOX_DecSel(hListBox);
    GUI_Delay(SPEED / 6);
}
GUI_Delay(SPEED / 4);
/* Change width of scrollbar */
GUI_DispStringAtCEOL("Change scrollbar width", 5, 55);
GUI_Delay(SPEED * 0.7);
{
    SCROLLBAR_Handle hScrollH = WM_GetDialogItem(hListBox, GUI_ID_HSCROLL);
    SCROLLBAR_Handle hScrollV = WM_GetDialogItem(hListBox, GUI_ID_VSCROLL);
    SCROLLBAR_SetWidth(hScrollV, 14);
    GUI_Delay(SPEED / 4);
    SCROLLBAR_SetWidth(hScrollH, 14);
    GUI_Delay(SPEED * 0.6);
}
/* Change size of listbox */
GUI_DispStringAtCEOL("Change size of listbox", 5, 55);
GUI_Delay(SPEED * 0.75);
WM_ResizeWindow(hListBox, -15, 0);
GUI_Delay(SPEED / 4);
WM_ResizeWindow(hListBox, 0, -15);
GUI_Delay(SPEED / 4);
WM_ResizeWindow(hListBox, 15, 0);
GUI_Delay(SPEED / 4);
WM_ResizeWindow(hListBox, 0, 15);
GUI_Delay(SPEED / 2);
/* Disable item */
GUI_DispStringAtCEOL("LISTBOX_SetItemDisabled", 5, 55);
GUI_Delay(SPEED);
LISTBOX_SetItemDisabled(hListBox, 4, 1);
GUI_Delay(SPEED * 0.75);
/* Set multi selection mode */
GUI_DispStringAtCEOL("LISTBOX_SetMulti", 5, 55);
GUI_Delay(SPEED);
LISTBOX_SetMulti(hListBox, 1);
GUI_Delay(SPEED * 0.75);
/* Select item */
GUI_DispStringAtCEOL("LISTBOX_SetItemSel", 5, 55);
GUI_Delay(SPEED);
LISTBOX_SetItemSel(hListBox, 0, 1);
GUI_Delay(SPEED / 4);
LISTBOX_SetItemSel(hListBox, 1, 1);
GUI_Delay(SPEED / 4);
LISTBOX_SetItemSel(hListBox, 2, 1);
GUI_Delay(SPEED * 0.8);
/* Delete listbox widget */
GUI_DispStringAtCEOL("LISTBOX_Delete", 5, 55);
GUI_Delay(SPEED * 1.1);
LISTBOX_Delete(hListBox);
GUI_Delay(SPEED * 0.75);
}

*****
*      _DemoListBox
*/
static void _DemoListBox(void) {
    LISTBOX_Handle hListBox;
    /* Display title */
    GUI_SetBkColor(0xB00000);
    GUI_SetColor(0xFFFFF);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("Listbox - Sample", 160, 5);
    GUI_Delay(SPEED / 2);
    /* Create listbox */
    GUI_SetFont(&GUI_Font8x16);
    GUI_SetTextAlign(GUI_TA_LEFT);
    GUI_DispStringAtCEOL("using", 5, 40);
    GUI_DispStringAtCEOL("LISTBOX_Create", 5, 55);
    GUI_Delay(SPEED * 0.9);
    hListBox = LISTBOX_Create(_ListBox, 100, 80, 120, 115, WM_CF_SHOW);
    GUI_Delay(SPEED * 0.75);
    /* Show several functions of listbox */
    _ShowSeveralFunctions(hListBox);
    /* Clear display */
    GUI_Clear();
    GUI_Delay(SPEED * 1.5);
}
*****

```

```

*
*      _DemoListBoxAsChild
*/
static void _DemoListBoxAsChild(void) {
    FRAMEWIN_Handle hFrame;
    LISTBOX_Handle hListBox;
    /* Display title */
    GUI_SetBkColor(0xB00000);
    GUI_SetColor(0xFFFFFFFF);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("Listbox as child - Sample", 160, 5);
    GUI_Delay(SPEED / 2);
    /* Create framewin */
    GUI_SetFont(&GUI_Font8x16);
    GUI_SetTextAlign(GUI_TA_LEFT);
    GUI_DisppStringAtCEOL("using", 5, 40);
    GUI_DisppStringAtCEOL("FRAMEWIN_Create", 5, 55);
    GUI_Delay(SPEED);
    hFrame = FRAMEWIN_Create("List box", NULL, WM_CF_SHOW, 100, 80, 120, 140);
    FRAMEWIN_SetFont(hFrame, &GUI_Font16B_ASCII);
    FRAMEWIN_SetActive(hFrame, 1);
    GUI_Delay(SPEED * 0.75);
    /* Create listbox */
    GUI_DisppStringAtCEOL("LISTBOX_CreateAsChild", 5, 55);
    GUI_Delay(SPEED);
    hListBox = LISTBOX_CreateAsChild(_ListBox, WM_GetClientWindow(hFrame), 0, 0, 0, 0,
    WM_CF_SHOW);
    GUI_Delay(SPEED * 0.75);
    /* Show several functions of listbox */
    _ShowSeveralFunctions(hListBox);
    /* Delete framewin widget */
    GUI_DisppStringAtCEOL("FRAMEWIN_Delete", 5, 55);
    GUI_Delay(SPEED);
    FRAMEWIN_Delete(hFrame);
    GUI_Delay(SPEED * 0.75);
    /* Clear display */
    GUI_Clear();
    GUI_Delay(SPEED * 1.5);
}

*****
*
*      MainTask
*
*      Demonstrates LISTBOX widget
*
*****
*/
void MainTask(void) {
    GUI_Init();
    WM_SetDesktopColor(0xB00000);
    GUI_Exec();
    while(1) {
        _DemoListBox();
        _DemoListBoxAsChild();
    }
}

```

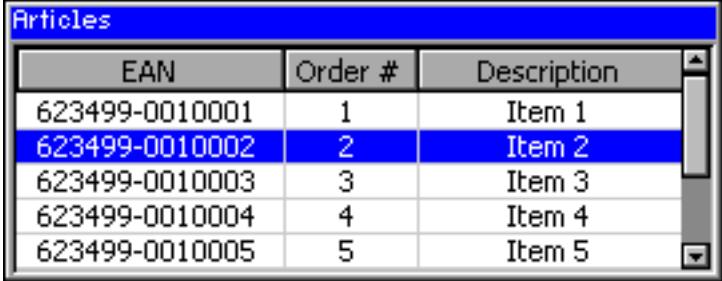
Screen shots of above example



15.11 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:

Explanation	LISTVIEW widget																					
No focus No surrounding FRAMEWIN No SCROLLBAR attached Grid lines not visible	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #cccccc;"> <th style="width: 33%;">EAN</th><th style="width: 33%;">Order #</th><th style="width: 33%;">Description</th></tr> </thead> <tbody> <tr><td>623499-0010001</td><td>1</td><td>Item 1</td></tr> <tr style="background-color: #cccccc;"><td>623499-0010002</td><td>2</td><td>Item 2</td></tr> <tr><td>623499-0010003</td><td>3</td><td>Item 3</td></tr> <tr><td>623499-0010004</td><td>4</td><td>Item 4</td></tr> <tr><td>623499-0010005</td><td>5</td><td>Item 5</td></tr> </tbody> </table>	EAN	Order #	Description	623499-0010001	1	Item 1	623499-0010002	2	Item 2	623499-0010003	3	Item 3	623499-0010004	4	Item 4	623499-0010005	5	Item 5			
EAN	Order #	Description																				
623499-0010001	1	Item 1																				
623499-0010002	2	Item 2																				
623499-0010003	3	Item 3																				
623499-0010004	4	Item 4																				
623499-0010005	5	Item 5																				
Has input focus No surrounding FRAMEWIN No SCROLLBAR attached Grid lines not visible	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #cccccc;"> <th style="width: 33%;">EAN</th><th style="width: 33%;">Order #</th><th style="width: 33%;">Description</th></tr> </thead> <tbody> <tr><td>623499-0010001</td><td>1</td><td>Item 1</td></tr> <tr style="background-color: #0000ff; color: white;"><td>623499-0010002</td><td>2</td><td>Item 2</td></tr> <tr><td>623499-0010003</td><td>3</td><td>Item 3</td></tr> <tr><td>623499-0010004</td><td>4</td><td>Item 4</td></tr> <tr><td>623499-0010005</td><td>5</td><td>Item 5</td></tr> </tbody> </table>	EAN	Order #	Description	623499-0010001	1	Item 1	623499-0010002	2	Item 2	623499-0010003	3	Item 3	623499-0010004	4	Item 4	623499-0010005	5	Item 5			
EAN	Order #	Description																				
623499-0010001	1	Item 1																				
623499-0010002	2	Item 2																				
623499-0010003	3	Item 3																				
623499-0010004	4	Item 4																				
623499-0010005	5	Item 5																				
Has input focus With surrounding FRAMEWIN No SCROLLBAR attached Grid lines not visible	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #0000ff; color: white;"> <th colspan="3" style="padding: 2px;">Articles</th></tr> <tr style="background-color: #cccccc;"> <th style="width: 33%;">EAN</th><th style="width: 33%;">Order #</th><th style="width: 33%;">Description</th></tr> </thead> <tbody> <tr><td>623499-0010001</td><td>1</td><td>Item 1</td></tr> <tr style="background-color: #0000ff; color: white;"><td>623499-0010002</td><td>2</td><td>Item 2</td></tr> <tr><td>623499-0010003</td><td>3</td><td>Item 3</td></tr> <tr><td>623499-0010004</td><td>4</td><td>Item 4</td></tr> <tr><td>623499-0010005</td><td>5</td><td>Item 5</td></tr> </tbody> </table>	Articles			EAN	Order #	Description	623499-0010001	1	Item 1	623499-0010002	2	Item 2	623499-0010003	3	Item 3	623499-0010004	4	Item 4	623499-0010005	5	Item 5
Articles																						
EAN	Order #	Description																				
623499-0010001	1	Item 1																				
623499-0010002	2	Item 2																				
623499-0010003	3	Item 3																				
623499-0010004	4	Item 4																				
623499-0010005	5	Item 5																				
Has input focus With surrounding FRAMEWIN SCROLLBAR attached Grid lines not visible	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #0000ff; color: white;"> <th colspan="3" style="padding: 2px;">Articles</th></tr> <tr style="background-color: #cccccc;"> <th style="width: 33%;">EAN</th><th style="width: 33%;">Order #</th><th style="width: 33%;">Description</th></tr> </thead> <tbody> <tr><td>623499-0010001</td><td>1</td><td>Item 1</td></tr> <tr style="background-color: #0000ff; color: white;"><td>623499-0010002</td><td>2</td><td>Item 2</td></tr> <tr><td>623499-0010003</td><td>3</td><td>Item 3</td></tr> <tr><td>623499-0010004</td><td>4</td><td>Item 4</td></tr> <tr><td>623499-0010005</td><td>5</td><td>Item 5</td></tr> </tbody> </table>	Articles			EAN	Order #	Description	623499-0010001	1	Item 1	623499-0010002	2	Item 2	623499-0010003	3	Item 3	623499-0010004	4	Item 4	623499-0010005	5	Item 5
Articles																						
EAN	Order #	Description																				
623499-0010001	1	Item 1																				
623499-0010002	2	Item 2																				
623499-0010003	3	Item 3																				
623499-0010004	4	Item 4																				
623499-0010005	5	Item 5																				

Explanation	LISTVIEW widget
Has input focus With surrounding FRAMEWIN SCROLLBAR attached Grid lines visible	

15.11.1 Configuration options

Type	Macro	Default	Explanation
S	LISTVIEW_FONT_DEFAULT	&GUI_Font13_1	Default font
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_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	LISTVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color, selected state without focus.
N	LISTVIEW_TEXTCOLOR2_DEFAULT	GUI_WHITE	Text color, selected state with focus.
N	LISTVIEW_GRIDCOLOR_DEFAULT	GUI_LIGHTGRAY	Color of grid lines (if shown)
N	LISTVIEW_ALIGN_DEFAULT	GUI_TA_VCENTER GUI_TA_HCENTER	Default text alignment

15.11.2 Notification codes

The following events are sent from a LISTVIEW widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off of the widget without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scrollbar has been changed.
WM_NOTIFICATION_SEL_CHANGED	The selection of the list box has changed.

15.11.3 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_DOWN	Moves the selection bar down.
GUI_KEY_UP	Moves the selection bar up.

15.11.4 LISTVIEW API

The table below lists the available emWin LISTVIEW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>LISTVIEW_AddColumn()</code>	Adds a column to a LISTVIEW.
<code>LISTVIEW_AddRow()</code>	Adds a row to a LISTVIEW.
<code>LISTVIEW_Create()</code>	Creates a LISTVIEW widget. (Obsolete)
<code>LISTVIEW_CreateAttached()</code>	Creates a LISTVIEW widget attached to a window.
<code>LISTVIEW_CreateEx()</code>	Creates a LISTVIEW widget.
<code>LISTVIEW_CreateIndirect()</code>	Creates a LISTVIEW widget from a resource table entry.
<code>LISTVIEW_DecSel()</code>	Decrements selection.
<code>LISTVIEW_DeleteColumn()</code>	Deletes the given column.
<code>LISTVIEW_DeleteRow()</code>	Deletes the given row.
<code>LISTVIEW_DisableRow()</code>	Sets the state of the given row to disabled.
<code>LISTVIEW_EnableRow()</code>	Sets the state of the given row to enabled.
<code>LISTVIEW_GetBkColor()</code>	Returns the background color of the LISTVIEW.
<code>LISTVIEW_GetFont()</code>	Returns the font of the LISTVIEW.
<code>LISTVIEW_GetHeader()</code>	Returns the handle of the attached HEADER widget.
<code>LISTVIEW_GetItemText()</code>	Returns the text of the given cell.
<code>LISTVIEW_GetNumColumns()</code>	Returns the number of columns.
<code>LISTVIEW_GetNumRows()</code>	Returns the number of rows.
<code>LISTVIEW_GetSel()</code>	Returns the number of the selected item.
<code>LISTVIEW_GetTextColor()</code>	Returns the text color of the LISTVIEW.
<code>LISTVIEW_IncSel()</code>	Increments selection.
<code>LISTVIEW_SetBkColor()</code>	Sets the background color.
<code>LISTVIEW_SetColumnWidth()</code>	Sets the column width.
<code>LISTVIEW_SetDefaultBkColor()</code>	Sets the default background color for HEADER widgets.
<code>LISTVIEW_SetDefaultFont()</code>	Sets the default font for HEADER widgets.
<code>LISTVIEW_SetDefaultGridColor()</code>	Sets the default text color for HEADER widgets.
<code>LISTVIEW_SetDefaultTextColor()</code>	Sets the default color of the grid lines for HEADER widgets.
<code>LISTVIEW_SetFont()</code>	Sets the font of the LISTVIEW.
<code>LISTVIEW_SetGridVis()</code>	Sets the visibility flag of the grid lines.
<code>LISTVIEW_SetItemBkColor()</code>	Sets the background color of a LISTVIEW cell
<code>LISTVIEW_SetItemText()</code>	Sets the text of a LISTVIEW cell.
<code>LISTVIEW_SetItemTextColor()</code>	Sets the text color of a LISTVIEW cell
<code>LISTVIEW_SetLBorder()</code>	Sets the number of pixels used for the left border.
<code>LISTVIEW_SetRBorder()</code>	Sets the number of pixels used for the right border.
<code>LISTVIEW_SetRowHeight()</code>	Sets the row height of the LISTVIEW
<code>LISTVIEW_SetSel()</code>	Sets the current selection.
<code>LISTVIEW_Set TextAlign()</code>	Sets the text alignment of a column.
<code>LISTVIEW_SetTextColor()</code>	Sets the text color.

`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	Meaning
hObj	Handle of 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.

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.

Add. 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	Meaning
hObj	Handle of widget
ppText	Pointer to array containing the text of the LISTVIEW cells

Add. information

The `ppText`-array should contain one item for each column of the HEADER-widget.

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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
SpecialFlags	(Reserved for later use)

Return value

Handle for the created LISTVIEW widget; 0 if the routine 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	Meaning
hObj	Handle of widget
Id	Id of the new LISTVIEW widget
SpecialFlags	(Not used, reserved for later use)

Return value

Handle for the created LISTVIEW widget; 0 if the routine fails.

Add. 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	Meaning
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).

Parameter	Meaning
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	Not used, reserved for future use.
Id	Window ID of the widget.

Return value

Handle for the created widget; 0 if the routine fails.

LISTVIEW_CreateIndirect()**Description**

Prototype explained at the beginning of the chapter.

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	Meaning
hObj	Handle of widget

Add. information

Please 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.

Prototype

```
void LISTVIEW_DeleteColumn(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of widget
Index	Zero based index of column to be deleted.

Add. information

Please 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.

Prototype

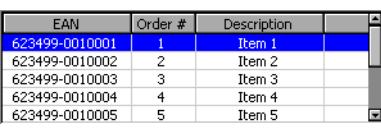
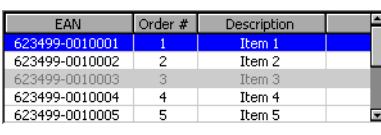
```
void LISTVIEW_DeleteRow(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of widget
Index	Zero based index of row to be deleted.

Add. information

Please note that the numbering of items always starts from the top with a value of 0.

LISTVIEW_DisableRow()

Before	After
	

Description

The function sets the state of the given row to disabled.

Prototype

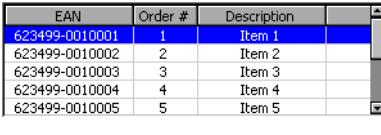
```
void LISTVIEW_DisableRow(LISTVIEW_Handle hObj, unsigned Row);
```

Parameter	Meaning
hObj	Handle of widget
Row	Zero based index of the row to be disabled.

Add. information

When scrolling through a listview disabled items will be skipped. You can not scroll to a disabled listview item.

LISTVIEW_EnableRow()

Before	After
	

Description

The function sets the state of the given row to enabled.

Prototype

```
void LISTVIEW_EnableRow(LISTVIEW_Handle hObj, unsigned Row);
```

Parameter	Meaning
hObj	Handle of widget
Row	Zero based index of the row to be disabled.

Add. information

Please refer to `LISTVIEW_DisableRow()`.

LISTVIEW_GetBkColor()

Description

Returns the background color of the given listview.

Prototype

```
GUI_COLOR LISTVIEW_GetBkColor(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget
<code>Index</code>	Index of color (see table below)

Permitted values for parameter <code>Index</code>	
<code>LISTVIEW_CI_UNSEL</code>	Unselected element.
<code>LISTVIEW_CI_SEL</code>	Selected element, without focus.
<code>LISTVIEW_CI_SELFOCUS</code>	Selected element, with focus.

Return value

Background color of the given listview.

LISTVIEW_GetFont()

Description

Returns a pointer to the font used to display the text of the listview.

Prototype

```
const GUI_FONT * LISTVIEW_GetFont(LISTVIEW_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.

Return value

Pointer to the font used to display the text of the listview.

LISTVIEW_GetHeader()

Description

Returns the handle of the HEADER widget.

Prototype

```
HEADER_Handle LISTVIEW_GetHeader(LISTVIEW_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget

Return value

Handle of the HEADER widget.

Add. 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_GetItemText()

Description

Returns the text of the given listview cell by copying it to the given buffer.

Prototype

```
void LISTVIEW_GetItemText(LISTVIEW_Handle hObj,
                          unsigned Column, unsigned Row,
                          char* pBuffer, unsigned MaxSize);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget
<code>Column</code>	Zero based index of the cell's column.
<code>Row</code>	Zero based index of the cell's row
<code>pBuffer</code>	Pointer to a buffer to be filled by the routine.
<code>MaxSize</code>	Size in bytes of the buffer.

Add. 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	Meaning
<code>hObj</code>	Handle of widget

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	Meaning
<code>hObj</code>	Handle of widget

Return value

Number of rows of the given LISTVIEW widget.

LISTVIEW_GetSel()

Description

Returns the number of the currently selected row in a specified LISTVIEW widget.

Prototype

```
int LISTVIEW_GetSel(LISTVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

Number of the currently selected row.

LISTVIEW_GetTextColor()**Description**

Returns the text color of the given listview.

Prototype

```
GUI_COLOR LISTVIEW_GetTextColor(LISTVIEW_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index of color (see table below)

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.

Return value

Text color of the given listview.

LISTVIEW_IncSel()**Description**

Increment the list box selection (moves the selection bar of a specified LISTVIEW down by one item).

Prototype

```
void LISTVIEW_IncSel(LISTVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

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	Meaning
hObj	Handle of widget
Index	Index for background color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
<code>LISTVIEW_CI_UNSEL</code>	Unselected element.
<code>LISTVIEW_CI_SEL</code>	Selected element, without focus.
<code>LISTVIEW_CI_SELF_FOCUS</code>	Selected element, with focus.
<code>LISTVIEW_CI_DISABLED</code>	Disabled element.

Add. Information

To set the background color for a single cell the function `LISTVIEW_SetItemBkColor()` should be used.

LISTVIEW_SetColumnWidth()

Description

Sets the width of the given column.

Prototype

```
void LISTVIEW_SetColumnWidth(LISTVIEW_Handle hObj,
                             unsigned int Index,
                             int Width);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget
<code>Index</code>	Number of column
<code>Width</code>	New width

LISTVIEW_SetDefaultBkColor()

Description

Sets the default background color for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultBkColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
<code>Index</code>	Index of default background color (see table below)
<code>Color</code>	Color to be set as default

Permitted values for parameter Index	
<code>LISTVIEW_CI_UNSEL</code>	Unselected element.
<code>LISTVIEW_CI_SEL</code>	Selected element, without focus.
<code>LISTVIEW_CI_SELF_FOCUS</code>	Selected element, with focus.
<code>LISTVIEW_CI_DISABLED</code>	Disabled element.

Return value

Previous default value.

LISTVIEW_SetDefaultFont()

Description

Sets the default font for new LISTVIEW widgets.

Prototype

```
const GUI_FONT * LISTVIEW_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Meaning
pFont	Pointer to font used for new LISTVIEW widgets

Return value

Previous default value.

LISTVIEW_SetDefaultGridColor()

Description

Sets the default color of the grid lines for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultGridColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	New default value

Return value

Previous default value

LISTVIEW_SetDefaultTextColor()

Description

Sets the default text color for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultTextColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Index of default text color (see table below)
Color	Color to be set as default

Permitted values for parameter Index	
0	Unselected element.
1	Selected element, without focus.
2	Selected element, with focus.

Return value

Previous default value.

LISTVIEW_SetFont()

Description

Sets the listview font.

Prototype

```
void LISTVIEW_SetFont(LISTVIEW_Handle hObj, const GUI_FONT * pfont);
```

Parameter	Meaning
hObj	Handle of listview.
pFont	Pointer to the font.

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);
```

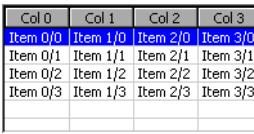
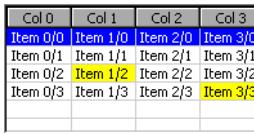
Parameter	Meaning
<code>hObj</code>	Handle of widget
<code>Show</code>	Sets the visibility of the grid lines

Permitted values for parameter <code>Show</code>	
0	Not visible.
1	Visible

Return value

Previous value of the visibility flag.

LISTVIEW_SetItemBkColor()

Before	After
	

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	Meaning
<code>hObj</code>	Handle of widget
<code>Column</code>	Number of column.
<code>Row</code>	Number of row.
<code>Index</code>	Index of background color (see table below).
<code>Color</code>	Color to be used.

Permitted values for parameter <code>Index</code>	
<code>LISTVIEW_CI_UNSEL</code>	Unselected element.
<code>LISTVIEW_CI_SEL</code>	Selected element, without focus.
<code>LISTVIEW_CI_SELFOCUS</code>	Selected element, with focus.

Add. 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	Meaning
hObj	Handle of widget.
Column	Number of column.
Row	Number of row.
s	Text to be displayed in the table cell.

LISTVIEW_SetItemTextColor()

Before	After																																																
<table border="1"> <thead> <tr> <th>Col 0</th><th>Col 1</th><th>Col 2</th><th>Col 3</th></tr> </thead> <tbody> <tr> <td>Item 0/0</td><td>Item 1/0</td><td>Item 2/0</td><td>Item 3/0</td></tr> <tr> <td>Item 0/1</td><td>Item 1/1</td><td>Item 2/1</td><td>Item 3/1</td></tr> <tr> <td>Item 0/2</td><td>Item 1/2</td><td>Item 2/2</td><td>Item 3/2</td></tr> <tr> <td>Item 0/3</td><td>Item 1/3</td><td>Item 2/3</td><td>Item 3/3</td></tr> <tr> <td></td><td></td><td></td><td></td></tr> </tbody> </table>	Col 0	Col 1	Col 2	Col 3	Item 0/0	Item 1/0	Item 2/0	Item 3/0	Item 0/1	Item 1/1	Item 2/1	Item 3/1	Item 0/2	Item 1/2	Item 2/2	Item 3/2	Item 0/3	Item 1/3	Item 2/3	Item 3/3					<table border="1"> <thead> <tr> <th>Col 0</th><th>Col 1</th><th>Col 2</th><th>Col 3</th></tr> </thead> <tbody> <tr> <td>Item 0/0</td><td>Item 1/0</td><td>Item 2/0</td><td>Item 3/0</td></tr> <tr> <td>Item 0/1</td><td>Item 1/1</td><td>Item 2/1</td><td>Item 3/1</td></tr> <tr> <td>Item 0/2</td><td>Item 1/2</td><td>Item 2/2</td><td>Item 3/2</td></tr> <tr> <td>Item 0/3</td><td>Item 1/3</td><td>Item 2/3</td><td>Item 3/3</td></tr> <tr> <td></td><td></td><td></td><td></td></tr> </tbody> </table>	Col 0	Col 1	Col 2	Col 3	Item 0/0	Item 1/0	Item 2/0	Item 3/0	Item 0/1	Item 1/1	Item 2/1	Item 3/1	Item 0/2	Item 1/2	Item 2/2	Item 3/2	Item 0/3	Item 1/3	Item 2/3	Item 3/3				
Col 0	Col 1	Col 2	Col 3																																														
Item 0/0	Item 1/0	Item 2/0	Item 3/0																																														
Item 0/1	Item 1/1	Item 2/1	Item 3/1																																														
Item 0/2	Item 1/2	Item 2/2	Item 3/2																																														
Item 0/3	Item 1/3	Item 2/3	Item 3/3																																														
Col 0	Col 1	Col 2	Col 3																																														
Item 0/0	Item 1/0	Item 2/0	Item 3/0																																														
Item 0/1	Item 1/1	Item 2/1	Item 3/1																																														
Item 0/2	Item 1/2	Item 2/2	Item 3/2																																														
Item 0/3	Item 1/3	Item 2/3	Item 3/3																																														

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	Meaning
hObj	Handle of widget
Column	Number of column.
Row	Number of row.
Index	Index of text color (see table below).
Color	Color to be used.

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.

Add. Information

This function overwrites the default text color for the given cell set by LISTVIEW_SetTextColor().

LISTVIEW_SetLBorder()

Before	After																														
<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Column 0</th> <th>Column 1</th> <th>Column 2</th> </tr> </thead> <tbody> <tr> <td>Item 0/0</td> <td>Item 1/0</td> <td>Item 2/0</td> </tr> <tr> <td>Item 0/1</td> <td>Item 1/1</td> <td>Item 2/1</td> </tr> <tr> <td>Item 0/2</td> <td>Item 1/2</td> <td>Item 2/2</td> </tr> <tr> <td>Item 0/3</td> <td>Item 1/3</td> <td>Item 2/3</td> </tr> </tbody> </table>	Column 0	Column 1	Column 2	Item 0/0	Item 1/0	Item 2/0	Item 0/1	Item 1/1	Item 2/1	Item 0/2	Item 1/2	Item 2/2	Item 0/3	Item 1/3	Item 2/3	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Column 0</th> <th>Column 1</th> <th>Column 2</th> </tr> </thead> <tbody> <tr> <td>Item 0/0</td> <td>Item 1/0</td> <td>Item 2/0</td> </tr> <tr> <td>Item 0/1</td> <td>Item 1/1</td> <td>Item 2/1</td> </tr> <tr> <td>Item 0/2</td> <td>Item 1/2</td> <td>Item 2/2</td> </tr> <tr> <td>Item 0/3</td> <td>Item 1/3</td> <td>Item 2/3</td> </tr> </tbody> </table>	Column 0	Column 1	Column 2	Item 0/0	Item 1/0	Item 2/0	Item 0/1	Item 1/1	Item 2/1	Item 0/2	Item 1/2	Item 2/2	Item 0/3	Item 1/3	Item 2/3
Column 0	Column 1	Column 2																													
Item 0/0	Item 1/0	Item 2/0																													
Item 0/1	Item 1/1	Item 2/1																													
Item 0/2	Item 1/2	Item 2/2																													
Item 0/3	Item 1/3	Item 2/3																													
Column 0	Column 1	Column 2																													
Item 0/0	Item 1/0	Item 2/0																													
Item 0/1	Item 1/1	Item 2/1																													
Item 0/2	Item 1/2	Item 2/2																													
Item 0/3	Item 1/3	Item 2/3																													

Description

Sets the number of pixels used for the left border within each cell of the listview.

Prototype

```
void LISTVIEW_SetLBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

Parameter	Meaning
hObj	Handle of widget.
BorderSize	Number of pixels to be used.

Add. Information

Using this function has no effect to the header widget used by the listview.

LISTVIEW_SetRBorder()

Before	After																														
<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Column 0</th> <th>Column 1</th> <th>Column 2</th> </tr> </thead> <tbody> <tr> <td>Item 0/0</td> <td>Item 1/0</td> <td>Item 2/0</td> </tr> <tr> <td>Item 0/1</td> <td>Item 1/1</td> <td>Item 2/1</td> </tr> <tr> <td>Item 0/2</td> <td>Item 1/2</td> <td>Item 2/2</td> </tr> <tr> <td>Item 0/3</td> <td>Item 1/3</td> <td>Item 2/3</td> </tr> </tbody> </table>	Column 0	Column 1	Column 2	Item 0/0	Item 1/0	Item 2/0	Item 0/1	Item 1/1	Item 2/1	Item 0/2	Item 1/2	Item 2/2	Item 0/3	Item 1/3	Item 2/3	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Column 0</th> <th>Column 1</th> <th>Column 2</th> </tr> </thead> <tbody> <tr> <td>Item 0/0</td> <td>Item 1/0</td> <td>Item 2/0</td> </tr> <tr> <td>Item 0/1</td> <td>Item 1/1</td> <td>Item 2/1</td> </tr> <tr> <td>Item 0/2</td> <td>Item 1/2</td> <td>Item 2/2</td> </tr> <tr> <td>Item 0/3</td> <td>Item 1/3</td> <td>Item 2/3</td> </tr> </tbody> </table>	Column 0	Column 1	Column 2	Item 0/0	Item 1/0	Item 2/0	Item 0/1	Item 1/1	Item 2/1	Item 0/2	Item 1/2	Item 2/2	Item 0/3	Item 1/3	Item 2/3
Column 0	Column 1	Column 2																													
Item 0/0	Item 1/0	Item 2/0																													
Item 0/1	Item 1/1	Item 2/1																													
Item 0/2	Item 1/2	Item 2/2																													
Item 0/3	Item 1/3	Item 2/3																													
Column 0	Column 1	Column 2																													
Item 0/0	Item 1/0	Item 2/0																													
Item 0/1	Item 1/1	Item 2/1																													
Item 0/2	Item 1/2	Item 2/2																													
Item 0/3	Item 1/3	Item 2/3																													

Description

Sets the number of pixels used for the right border within each cell of the listview.

Prototype

```
void LISTVIEW_SetRBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

Parameter	Meaning
hObj	Handle of widget.
BorderSize	Number of pixels to be used.

Add. Information

Using this function has no effect to the header widget used by the listview.

LISTVIEW_SetRowHeight()

Description

Sets the number of pixels used for the height of each row of the listview.

Prototype

```
unsigned LISTVIEW_SetRowHeight(LISTVIEW_Handle hObj, unsigned RowHeight);
```

Parameter	Meaning
hObj	Handle of widget.

Return value

Previous value of the row height set by this function. If the return value is 0, the height of the rows depends on the height of the font used by the widget.

Add. 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	Meaning
hObj	Handle of widget
Sel	Element to be selected.

LISTVIEW_Set.TextAlign()

Description

Sets the alignment for the given column.

Prototype

```
void LISTVIEW_Set.TextAlign(LISTVIEW_Handle hObj,
                           unsigned int Index,
                           int Align);
```

Parameter	Meaning
hObj	Handle of widget
Index	Number of column
Align	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

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

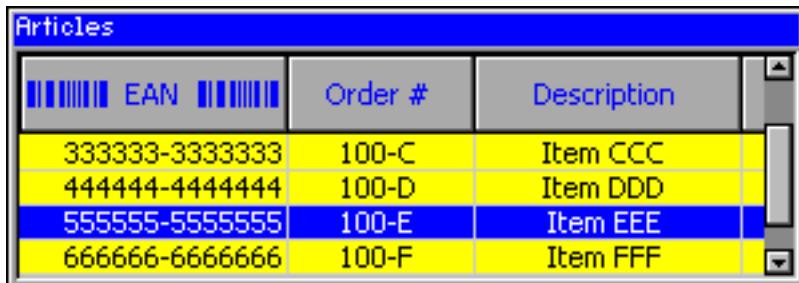
```
void LISTVIEW_SetTextColor(LISTVIEW_Handle hObj,
                           unsigned int Index,
                           GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index for text color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.

15.11.5 Example

The source of the following sample is available as `WIDGET_Header.c` in the samples shipped with emWin:

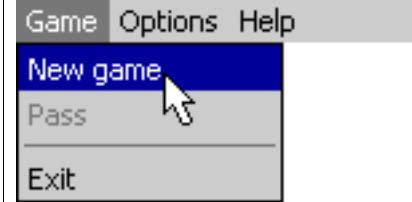
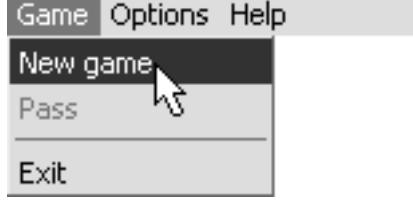
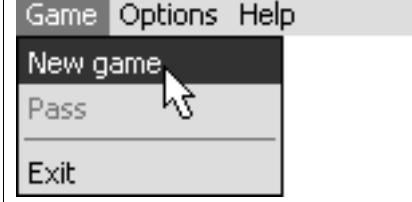
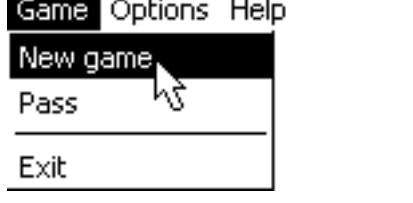
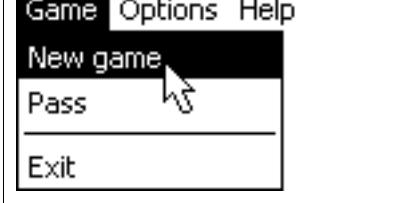


15.12 MENU: Menu widget

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.

The shipment of emWin contains a application sample 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:

Explanation	Menu using WIDGET_Effect_3D1L	Menu using WIDGET_Effect_Simple
Color display (8666 mode)		
Monochrome display (16 gray scales)		
Black/white display		

The table above shows the appearance of the menu widget using its default effect WIDGET_Effect_3D1L and using WIDGET_Effect_Simple. It also works with all other effects.

15.12.1 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 MENU_MSG_DATA

Data type	Element	Meaning
U16	MsgType	(see table below)
U16	ItemId	Id of menu item.

Permitted values for element MsgType	
MENU_ON_INITMENU	This message is send to the owner of menu immediately before the menu opens. This gives the application the chance to modify the menu before it is shown.
MENU_ON_ITEMSELECT	This message is send to the owner of a menu immediately after a menu item is selected. The ItemId element contains the Id of the pressed menu item.

Sample

The following sample 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_INITMENU:
                    ...
                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;
            }
    }
}
```

15.12.2 Data structures

The following shows the menu widget related data structures.

MENU_ITEM_DATA

This structure serves as a container to set or retrieve information about menu items.

Elements of MENU_ITEM_DATA

Data type	Element	Meaning
const char *	pText	Menu item text.
U16	Id	Id of the menu item.
U16	Flags	(see table below)
MENU_Handle	h_submenu	If the item represents a submenu this element contains the handle of the submenu.

Permitted values for element Flags	
MENU_IF_DISABLED	Item is disabled.
MENU_IF_SEPARATOR	Item is a separator.

15.12.3 Configuration options

Type	Macro	Default	Explanation
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 submenu 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 unselected 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.

15.12.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

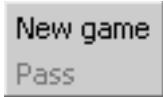
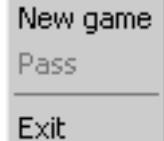
Key	Reaction
GUI_KEY_RIGHT	<ul style="list-style-type: none"> - If the menu is horizontal, the selection moves one item to the right. - If the menu is vertical and the current item is a submenu, the submenu opens and the input focus moves to the submenu. - If the menu is vertical and the current item 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	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> - If the menu is vertical, the selection moves to the previous item. - If the menu is not the top level menu the current menu will be closed and the focus moves to the previous menu.
GUI_KEY_ESCAPE	<ul style="list-style-type: none"> - If the menu is the top level menu, the current menu item becomes unselected.
GUI_KEY_ENTER	<ul style="list-style-type: none"> - 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 send to the owner of the menu.

15.12.5 MENU API

The table below lists the available emWin MENU-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>MENU_AddItem()</code>	Adds an item to an existing menu.
<code>MENU_Attach()</code>	Attaches a menu with the given size at the given position to a specified window.
<code>MENU_CreateEx()</code>	Creates a menu widget.
<code>MENU_DeleteItem()</code>	Deletes the specified menu item.
<code>MENU_DisableItem()</code>	Disables the specified menu item.
<code>MENU_EnableItem()</code>	Enables the specified menu item.
<code>MENU_GetDefaultBkColor()</code>	Returns the default background color for new menus.
<code>MENU_GetDefaultBorderSize()</code>	Returns the default border size for new menus.
<code>MENU_GetDefaultEffect()</code>	Returns the default effect for new menus.
<code>MENU_GetDefaultFont()</code>	Returns a pointer to the default font used to display the menu item text of new menus.
<code>MENU_GetDefaultTextColor()</code>	Returns the default text color for new menus.
<code>MENU_GetItem()</code>	Retrieves information about the given menu item.
<code>MENU_GetItemText()</code>	Returns the text of the given menu item.
<code>MENU_GetNumItems()</code>	Returns the number of items of the given menu.
<code>MENU_InsertItem()</code>	Inserts a menu item.
<code>MENU_SetBkColor()</code>	Sets the background color of the given menu.
<code>MENU_SetBorderSize()</code>	Sets the border size of the given menu.
<code>MENU_SetDefaultBkColor()</code>	Sets the default background color for new menus.
<code>MENU_SetDefaultBorderSize()</code>	Sets the default border size for new menus.
<code>MENU_SetDefaultEffect()</code>	Sets the default effect for new menus.
<code>MENU_SetDefaultFont()</code>	Sets a pointer to the default font used to display the menu item text of new menus.
<code>MENU_SetDefaultTextColor()</code>	Sets the default text color for new menus.
<code>MENU_SetFont()</code>	Sets the font used to display the menu item text of the given menu.
<code>MENU_SetItem()</code>	Changes the information about the given menu item.
<code>MENU_SetOwner()</code>	Sets the window to be informed by the menu.
<code>MENU_SetTextColor()</code>	Sets the text color of the given menu.

`MENU_AddItem()`

Before	After
	

Description

This function adds a new item to the end of the given menu.

Prototype

```
void MENU_AddItem(MENU_Handle hObj, const MENU_ITEM_DATA * pItemData);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>pItemData</code>	Pointer to a MENU_ITEM_DATA structure containing the information of the new item.

Add. Information

If using a menu 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 and no fixed sizes are used the size of the menu will be adapted.

Please refer to the beginning of the menu chapter for details about the MENU_ITEM_INFO data structure.

MENU_Attach()**Description**

Attaches the given menu 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	Meaning
<code>hObj</code>	Handle of widget.
<code>hDestWin</code>	Handle to the window to which the menu should be attached.
<code>x</code>	X position in window coordinates of the menu.
<code>y</code>	Y position in window coordinates of the menu.
<code>xSize</code>	Fixed X size of the menu. For details please refer to MENU_CreateEx().
<code>ySize</code>	Fixed Y size of the menu. For details please refer to MENU_CreateEx().
<code>Flags</code>	Reserved for future use

Add. Information

This function is useful if a menu should be shown in different windows. The typical use is a popup menu which should appear in different windows.

MENU_CreateEx()**Description**

Creates a menu 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	Meaning
<code>x0</code>	Leftmost pixel of the widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the widget (in parent coordinates).
<code>xSize</code>	Fixed horizontal size of the widget (in pixels). 0 if menu should handle the xSize.
<code>ySize</code>	Fixed vertical size of the widget (in pixels). 0 if menu should handle the ySize.

Parameter	Meaning
<code>hParent</code>	Handle of parent window. If 0, the new widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	(see table below)
<code>Id</code>	Window ID of the widget.

Permitted values for parameter <code>ExFlags</code>	
<code>MENU_CF_HORIZONTAL</code>	Creates a horizontal menu.
<code>MENU_CF_VERTICAL</code>	Creates a vertical menu.
<code>MENU_CF_OPEN_ON_POINTEROVER</code>	This flag specifies if the menu should open a submenu if the pointer input device (PID) points on the menu item. The default behavior is opening on clicking, that means submenus are opened only if the PID is clicked and released during pointing on the menu.

Return value

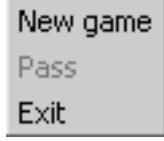
Handle for the created widget; 0 if the routine fails.

Add. Information

The parameters `xSize` and/or `ySize` specifies if a fixed width and/or height should be used for the menu.

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. If items are added or removed the size of the widget will be adapted.

MENU_DeleteItem()

Before	After
	

Description

Deletes a given menu entry from a menu.

Prototype

```
void MENU_DeleteItem(MENU_Handle hObj, U16 ItemId);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>ItemId</code>	Id of the menu item to be deleted.

Add. Information

If the item does not exist the function returns immediately.

When deleting items from a menu and no fixed sizes are used the window size will be adapted.

MENU_DisableItem()

Before	After
	

Description

Disables the given menu item.

Prototype

```
void MENU_DisableItem(MENU_Handle hObj, U16 ItemId);
```

Parameter	Meaning
hObj	Handle of widget.
ItemId	Id of the menu item to be disabled.

Add. Information

If a disabled menu item is selected, the menu widget sends no WM_MENU message to the owner. A disabled submenu item can not be opened.

MENU_EnableItem()

Before	After
	

Description

Enables the given menu item.

Prototype

```
void MENU_EnableItem(MENU_Handle hObj, U16 ItemId);
```

Parameter	Meaning
hObj	Handle of widget.
ItemId	Id of the menu item to be enabled.

Add. Information

For details please refer to function `MENU_DisableItem()`.

MENU_GetDefaultBkColor()

Description

Returns the default background color used to draw new menu items.

Prototype

```
GUI_COLOR MENU_GetDefaultBkColor(unsigned ColorIndex);
```

Parameter	Meaning
ColorIndex	Index of color to be returned (see table below).

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.

Add. Information

For details please refer to function `MENU_SetBkColor()`.

MENU_GetDefaultBorderSize()

Description

Returns the default border size used for new menu widgets.

Prototype

```
U8 MENU_GetDefaultBorderSize(unsigned BorderIndex);
```

Parameter	Meaning
BorderIndex	(see table below)

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.

Add. Information

For details please refer to function `MENU_SetBorderSize()`.

MENU_GetDefaultEffect()

Description

Returns the default effect for new menus.

Prototype

```
const WIDGET_EFFECT * MENU_GetDefaultEffect(void);
```

Return value

The result of the function is a pointer to a `WIDGET_EFFECT` structure.

Add. Information

For more information please refer to the function `WIDGET_SetDefaultEffect()`.

MENU_GetDefaultFont()**Description**

Returns a pointer to the default font used to display the menu item text of new menus.

Prototype

```
const GUI_FONT * MENU_GetDefaultFont(void);
```

Return value

Pointer to the default font used to display the menu item text of new menus.

MENU_GetDefaultTextColor()**Description**

Returns the default text color for new menus.

Prototype

```
GUI_COLOR MENU_GetDefaultTextColor(unsigned ColorIndex);
```

Parameter	Meaning
<code>ColorIndex</code>	Index of color to be returned (see table below)

Permitted values for parameter <code>ColorIndex</code>	
<code>MENU_CI_ACTIVE_SUBMENU</code>	Text color of active submenu items.
<code>MENU_CI_DISABLED</code>	Text color of disabled menu items.
<code>MENU_CI_DISABLED_SEL</code>	Text color of disabled and selected menu items.
<code>MENU_CI_ENABLED</code>	Text color of enabled and not selected menu items.
<code>MENU_CI_SELECTED</code>	Text color of enabled and selected menu items.

Return value

Default text color for new menus.

Add. Information

For details please refer to function `MENU_SetTextColor()`.

MENU_GetItem()**Description**

Retrieves information about the given menu item.

Prototype

```
void MENU_GetItem(MENU_Handle hObj, U16 ItemId, MENU_ITEM_DATA * pItemData);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>ItemId</code>	Id of the requested menu item.
<code>pItemData</code>	Pointer to a <code>MENU_ITEM_DATA</code> structure to be filled by the function.

Add. Information

If using a menu 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.

Please 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	Meaning
<code>hObj</code>	Handle of widget.
<code>ItemId</code>	Id of the requested menu item.
<code>pBuffer</code>	Buffer to be filled by the function.
<code>BufferSize</code>	Maximum number of bytes to be retrieved.

MENU_GetNumItems()

Description

Returns the number of items of the given menu.

Prototype

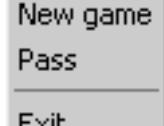
```
unsigned MENU_GetNumItems(MENU_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.

Return value

Number of items of the given menu.

MENU_InsertItem()

Before	After
	

Description

Inserts a menu item at the given position.

Prototype

```
void MENU_InsertItem(MENU_Handle hObj, U16 ItemId,
```

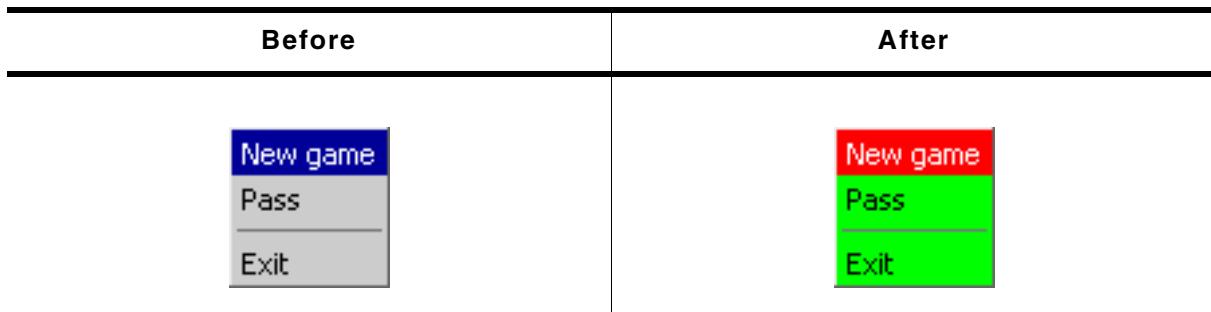
```
const MENU_ITEM_DATA * pItemData);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>ItemId</code>	Id of the menu item the new item should be inserted before.
<code>pItemData</code>	Pointer to a MENU_ITEM_DATA structure containing the information of the new item.

Add. Information

Please refer to the beginning of the menu chapter for details about the MENU_ITEM_INFO data structure.

MENU_SetBkColor()



Description

Sets the background color of the given menu.

Prototype

```
void MENU_SetBkColor(MENU_Handle hObj, unsigned ColorIndex,
                      GUI_COLOR Color);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>ColorIndex</code>	Index of color (see table below)
<code>Color</code>	Color to be used.

Permitted values for parameter <code>ColorIndex</code>	
<code>MENU_CI_ACTIVE_SUBMENU</code>	Background color of active submenu items.
<code>MENU_CI_DISABLED</code>	Background color of disabled menu items.
<code>MENU_CI_DISABLED_SEL</code>	Background color of disabled and selected menu items.
<code>MENU_CI_ENABLED</code>	Background color of enabled and not selected menu items.
<code>MENU_CI_SELECTED</code>	Background color of enabled and selected menu items.

MENU_SetBorderSize()

Before	After
	

The following code is executed between the screenshots above:

```
MENU_SetBorderSize(hMenuGame, MENU_BI_LEFT, 20);
```

Before	After
	

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.

Prototype

```
void MENU_SetBorderSize(MENU_Handle hObj, unsigned BorderIndex,
                       U8 BorderSize);
```

Parameter	Meaning
hObj	Handle of widget.
BorderIndex	(see table below)
BorderSize	Size to be used.

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	Meaning
ColorIndex	Index of color to be returned (see table below).
Color	Color to be used.

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.

Add. Information

For details please refer to function `MENU_SetBkColor()`.

MENU_SetDefaultBorderSize()

Description

Sets the default border size used for new menu widgets.

Prototype

```
void MENU_SetDefaultBorderSize(unsigned BorderIndex, U8 BorderSize);
```

Parameter	Meaning
BorderIndex	(see table below)
BorderSize	Border size to be used.

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.

Add. Information

For details please refer to function `MENU_SetBorderSize()`.

MENU_SetDefaultEffect()

Description

Sets the default effect for new menus.

Prototype

```
void MENU_SetDefaultEffect(const WIDGET_EFFECT* pEffect);
```

Parameter	Meaning
pEffect	Pointer to a WIDGET_EFFECT structure.

Add. Information

For more information please refer to the function `WIDGET_SetDefaultEffect()`.

MENU_SetDefaultFont()

Description

Sets the pointer to the default font used to display the menu item text of new menus.

Prototype

```
void MENU_SetDefaultFont(const GUI_FONT * pFont);
```

Parameter	Meaning
pFont	Pointer to the GUI_FONT structure to be used.

Add. Information

For details please refer to function `MENU_SetFont()`.

MENU_SetDefaultTextColor()

Description

Sets the default text color for new menus.

Prototype

```
void MENU_SetDefaultTextColor(unsigned ColorIndex, GUI_COLOR Color);
```

Parameter	Meaning
ColorIndex	Index of color to be used (see table below)
Color	Color to be used

Permitted values for parameter <code>ColorIndex</code>	
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.

Add. Information

For details please refer to function `MENU_SetTextColor()`.

MENU_SetFont()

Before	After
	

Description

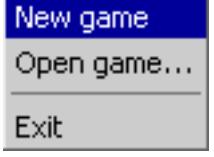
Sets the pointer to the default font used to display the menu item text of new menus.

Prototype

```
void MENU_SetFont(MENU_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Meaning
hObj	Handle of widget.
pFont	Pointer to the GUI_FONT structure to be used.

MENU_SetItem()

Before	After
	

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	Meaning
<code>hObj</code>	Handle of widget.
<code>ItemId</code>	Id of the menu item to be changed.
<code>pItemData</code>	Pointer to a MENU_ITEM_DATA structure containing the new information.

MENU_SetOwner()

Description

Sets the owner of the menu to be informed by the widget.

Prototype

```
void MENU_SetOwner(MENU_Handle hObj, WM_HWIN hOwner);
```

Parameter	Meaning
<code>hObj</code>	Handle of widget.
<code>hOwner</code>	Handle of the owner window which should receive the WM_MENU messages of the menu.

Add. Information

If no owner is set the parent window of the menu will receive WM_MENU messages. In some cases it makes sense to send the messages not to the parent window of the menu. In this case this function can be used to set the recipient for the WM_MENU messages.

MENU_SetTextColor()

Before	After
	

Description

Sets the text color of the given menu.

Prototype

```
void MENU_SetTextColor(MENU_Handle hObj,
                      unsigned ColorIndex, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
ColorIndex	Index of color to be used (see table below)
Color	Color to be used.

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.

15.13 MESSAGEBOX: Message box widget

A MESSAGEBOX widget 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. The table below shows the appearance of the MESSAGEBOX widget:

Simple message box



15.13.1 Configuration options

Type	Macro	Default	Explanation
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.

15.13.2 Keyboard reaction

The widget consists of a frame window, a text and a button widget. When executing a message box the button gains the input focus. So please refer to the keyboard reaction of the button widget for more information.

15.13.3 MESSAGEBOX API

The table below lists the available emWin MESSAGEBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
GUI_MessageBox()	Creates and displays a message box.
MESSAGEBOX_Create()	Creates a message box.

GUI_MessageBox()

Description

Creates and displays a message box.

Prototype

```
int GUI_MessageBox(const char* sMessage, const char* sCaption, int Flags);
```

Parameter	Meaning
sMessage	Message to display.
sCaption	Caption for the title bar of the frame window.
Flags	Reserved for future extensions, value does not matter.

Add. Information

This function gives you the possibility to create and execute a message box with one line of code. The sample folder contains the sample `DIALOG_MessageBox.c` which shows how to use this function.

MESSAGEBOX_Create()

Description

Creates a message box.

Prototype

```
WM_HWIN GUI_MessageBox(const char* sMessage, const char* sCaption, int Flags);
```

Parameter	Meaning
sMessage	Message to display.
sCaption	Caption for the title bar of the frame window.
Flags	(see table below)

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

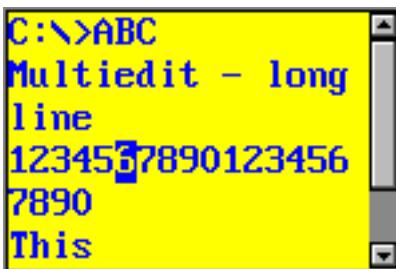
Handle of the message box window.

Add. information

This function gives you the possibility to create a messagebox before it is executed. The advantage of using this function is the possibility of changing the dialog properties or the dialog behaviour by using a user defined callback function. The function `GUI_ExecCreatedDialog()` should be used to execute the message box.

15.14 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 scrollbars. 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:

Explanation	Frame window
edit mode, automatic horizontal scrollbar, non wrapping mode, insert mode,	
edit mode, automatic vertical scrollbar, word wrapping mode, overwrite mode,	
read only mode, word wrapping mode	

15.14.1 Configuration options

Type	Macro	Default	Explanation
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.

15.14.2 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off of the widget without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scrollbar has been changed.
WM_NOTIFICATION_VALUE_CHANGED	The text of the widget has been changed.

15.14.3 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.

15.14.4 MULTIEDIT API

The table below lists the available emWin MULTIEDIT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
MULTIEDIT_AddKey()	Key input routine.
MULTIEDIT_Create()	Creates a multiedit widget. (Obsolete)
MULTIEDIT_CreateEx()	Creates a multiedit widget.
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_SetAutoScrollH()	Activates automatic use of a horizontal scrollbar.

Routine	Explanation
MULTIEDIT_SetAutoScrollV()	Activates automatic use of a vertical scrollbar.
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_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_SetTextColor()	Sets the text color,
MULTIEDIT_SetWrapWord()	Enables/disables word wrapping.
MULTIEDIT_SetWrapNone()	Enables/disables the non wrapping mode.

MULTIEDIT_AddKey()

Description

Adds user input to a specified multiedit widget.

Prototype

```
void MULTIEDIT_AddKey(MULTIEDIT_HANDLE hObj, int Key);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
Key	Character to be added.

Add. 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_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	Meaning
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.

Parameter	Meaning
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" 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.

Permitted values for parameter ExFlags	
MULTIEDIT_CF_AUTOSCROLLBAR_H	Automatic use of a horizontal scrollbar.
MULTIEDIT_CF_AUTOSCROLLBAR_V	Automatic use of a vertical scrollbar.
MULTIEDIT_CF_INSERT	Enables insert mode.
MULTIEDIT_CF_READONLY	Enables read only mode.

Return value

Handle for the created MULTIEDIT widget; 0 if the routine 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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" 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.

Permitted values for parameter ExFlags	
MULTIEDIT_CF_AUTOSCROLLBAR_H	Automatic use of a horizontal scrollbar.
MULTIEDIT_CF_AUTOSCROLLBAR_V	Automatic use of a vertical scrollbar.
MULTIEDIT_CF_INSERT	Enables insert mode.
MULTIEDIT_CF_READONLY	Enables read only mode.

Return value

Handle for the created widget; 0 if the routine fails.

MULTIEDIT_GetPrompt()

Description

Returns the current prompt text.

Prototype

```
void MULTIEDIT_GetPrompt(MULTIEDIT_HANDLE hObj, char * sDest, int MaxLen);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
sDest	Buffer for the prompt text to be returned.
MaxLen	Maximum number of bytes to be copied to sDest.

Add. 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	Meaning
hObj	Handle of multiedit widget.
sDest	Buffer for the text to be returned.
MaxLen	Maximum number of bytes to be copied to sDest.

Add. 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	Meaning
hObj	Handle of multiedit widget.

Return value

Buffer size used to store the current text (and prompt).

MULTIEDIT_SetAutoScrollH()

Description

Enables/disables the automatic use of a horizontal scrollbar.

Prototype

```
void MULTIEDIT_SetAutoScrollH(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
OnOff	(See table below)

Permitted values for parameter OnOff	
0	Disables automatic use of a horizontal scrollbar.
1	Enables automatic use of a horizontal scrollbar.

Add. information

Enabling the use of a automatic horizontal scrollbar makes only 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 scrollbar will be attached to the window.

MULTIEDIT_SetAutoScrollV()

Description

Enables/disables the automatic use of a vertical scrollbar.

Prototype

```
void MULTIEDIT_SetAutoScrollV(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
OnOff	(See table below)

Permitted values for parameter OnOff	
0	Disables automatic use of a vertical scrollbar.
1	Enables automatic use of a vertical scrollbar.

Add. information

If enabled the multiedit widget checks if the height of the text fits into the client area. If not a vertical scrollbar 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	Meaning
hObj	Handle of multiedit widget.
Index	(See table below)
Color	Background color to be used.

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	Meaning
hObj	Handle of multiedit widget.
BufferSize	Maximum number of bytes.

Add. information

The function clears the current contents 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	Meaning
hObj	Handle of multiedit widget.
Offset	New cursor position.

Add. 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_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	Meaning
hObj	Handle of multiedit widget.
pFont	Pointer to font to be used.

MULTIEDIT_SetInsertMode()

Description

Enables/disables the insert mode. The default behaviour is overwrite mode.

Prototype

```
void MULTIEDIT_SetInsertMode(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
OnOff	(See table below)

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	Meaning
hObj	Handle of multiedit widget.
MaxNumChars	Maximum number of characters.

MULTIEDIT_SetPasswordMode()

Description

Enables/disables the password mode.

Prototype

```
void MULTIEDIT_SetPasswordMode(MULTIEDIT_HANDLE hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
OnOff	(See table below)

Permitted values for parameter OnOff	
0	Disables password mode.
1	Enables password mode.

Add. 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	Meaning
hObj	Handle of multiedit widget.
sPrompt	Pointer to the new prompt text.

Add. 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	Meaning
hObj	Handle of multiedit widget.
OnOff	(see table below)

Permitted values for parameter OnOff	
0	Disables read only mode.
1	Enables read only mode.

Add. 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 widget.

Prototype

```
void MULTIEDIT_SetText(MULTIEDIT_HANDLE hObj, const char * s);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
s	Pointer to the text to be handled by the multiedit widget.

Add. 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_SetTextColor()

Description

Sets the text color.

Prototype

```
void MULTIEDIT_SetTextColor(MULTIEDIT_HANDLE hObj,
                           unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
Index	(See table below)
Color	Text color to be used.

Permitted values for parameter Index	
MULTIEDIT_CI_EDIT	Edit mode.
MULTIEDIT_CI_READONLY	Read only mode.

MULTIEDIT_SetWrapWord()

Description

Enables the word wrapping mode.

Prototype

```
void MULTIEDIT_SetWrapWord(MULTIEDIT_HANDLE hObj);
```

Parameter	Meaning
hObj	Handle of multiedit widget.

Add. 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	Meaning
hObj	Handle of multiedit widget.

Add. 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.

15.15 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:

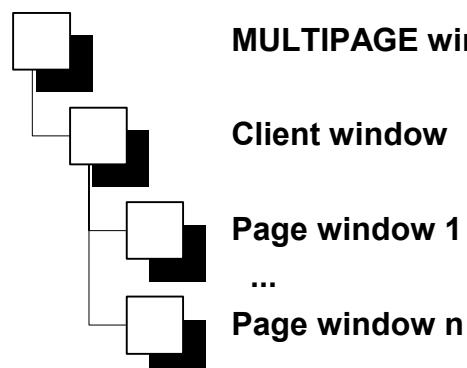
Explanation	MULTIPAGE widget
MULTIPAGE widget with 3 pages, alignment top/left.	
MULTIPAGE widget with 6 pages, alignment bottom/right.	

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.



15.15.1 Configuration options

Type	Macro	Default	Explanation
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.

15.15.2 Notification codes

The following events are sent from the widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off of the widget without releasing.
WM_NOTIFICATION_VALUE_CHANGED	The text of the widget has been changed.

15.15.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

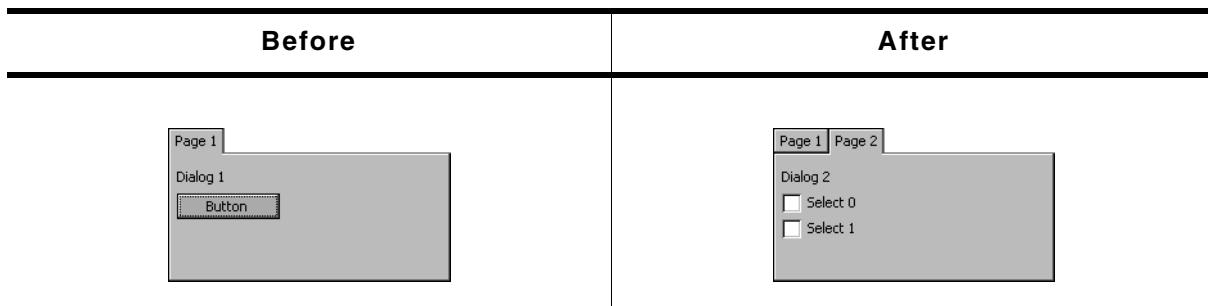
15.15.4 MULTIPAGE API

The table below lists the available emWin MULTIPAGE-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
MULTIPAGE_AddPage()	Adds a page to a MULTIPAGE widget.
MULTIPAGE_CreateEx()	Creates a new MULTIPAGE widget.
MULTIPAGE_CreateIndirect()	Creates the MULTIPAGE widget from a resource table entry.
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_GetDefaultAlign()	Returns the default alignment for new MULTIPAGE widgets.
MULTIPAGE_GetDefaultBkColor()	Returns the default background color for new MULTIPAGE widgets.
MULTIPAGE_GetDefaultFont()	Returns the default font used for new MULTIPAGE widgets.
MULTIPAGE_GetDefaultTextColor()	Returns the default text color used for new MULTIPAGE widgets.
MULTIPAGE_GetSelection()	Returns the current selection.
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_SetBkColor()	Sets the background color.

Routine	Explanation
<code>MULTIPAGE_SetDefaultAlign()</code>	Sets the default alignment for new MULTIPAGE widgets.
<code>MULTIPAGE_SetDefaultBkColor()</code>	Sets the default background color for new MULTIPAGE widgets.
<code>MULTIPAGE_SetDefaultFont()</code>	Sets the default font used by new MULTIPAGE widgets.
<code>MULTIPAGE_SetDefaultTextColor()</code>	Sets the default text color used by new MULTIPAGE widgets.
<code>MULTIPAGE_SetFont()</code>	Selects the font for the widget.
<code>MULTIPAGE_SetText()</code>	Sets the text displayed in a tab of a MULTIPAGE widget.
<code>MULTIPAGE_SetTextColor()</code>	Sets the text color.

MULTIPAGE_AddPage()



Description

Adds a new page to a given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_AddPage(MULTIPAGE_Handle hObj,
                        WM_HWIN hWin ,const char* pText);
```

Parameter	Meaning
<code>hObj</code>	Handle of MULTIPAGE widget.
<code>hWin</code>	Handle of window to be shown in the given page.
<code>pText</code>	Pointer to text to be displayed in the tab of the page.

Add. 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 new 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	Meaning
<code>x0</code>	X-position of the widget (in parent coordinates).
<code>y0</code>	Y-position of the widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the widget (in pixels).
<code>ysize</code>	Vertical size of the widget (in pixels).

Parameter	Meaning
<code>hParent</code>	Handle of parent window. If 0, the new widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 13: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	Not used yet, reserved for future use.
<code>Id</code>	Window ID of the widget.

Return value

Handle of the new widget.

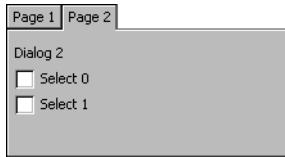
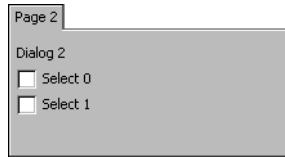
Add. information

The size of the tabs depends on the size of the font used for the MULTIPAGE widget.

MULTIPAGE_CreateIndirect()

Prototype explained at the beginning of the chapter.

MULTIPAGE_DeletePage()

Before	After
	

Description

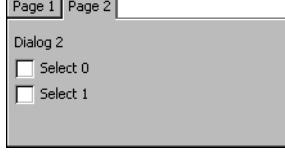
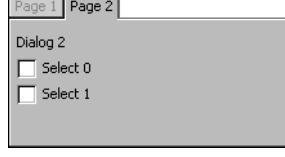
Removes a page from a MULTIPAGE widget and optional deletes the window.

Prototype

```
void MULTIPAGE_DeletePage(MULTIPAGE_Handle hObj,
                           unsigned Index, int Delete);
```

Parameter	Meaning
<code>hObj</code>	Handle of MULTIPAGE widget.
<code>Index</code>	Zero based index of the page to be removed from the MULTIPAGE widget.
<code>Delete</code>	If >0 the window attached to the page will be deleted.

MULTIPAGE_DisablePage()

Before	After
	

Description

Disables a page from a MULTIPAGE widget.

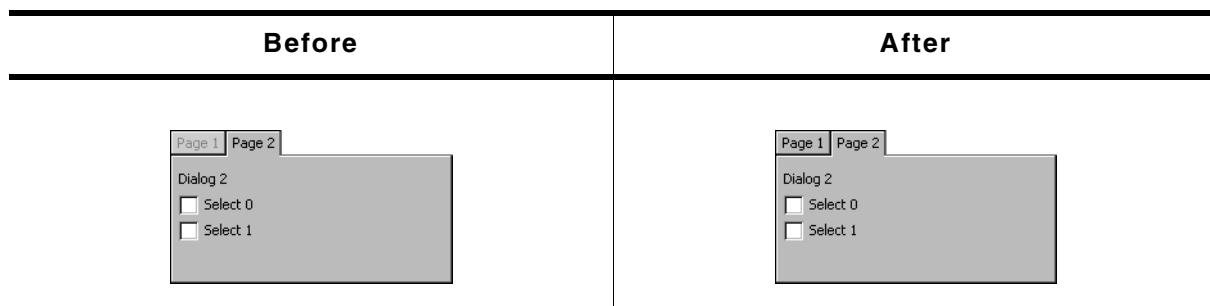
Prototype

```
void MULTIPAGE_DisablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of the page to be disabled.

Add. 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()**Description**

Enables a page of a MULTIPAGE widget.

Prototype

```
void MULTIPAGE_EnablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.

Add. information

The default state of MULTIEDIT pages is 'enabled'.

MULTIPAGE_GetDefaultAlign()**Description**

Returns the default tab alignment for new MULTIPAGE widgets.

Prototype

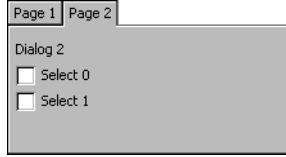
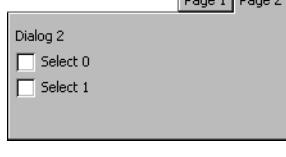
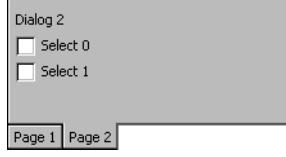
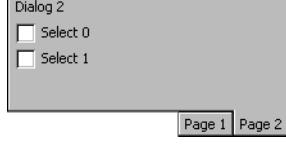
```
unsigned MULTIPAGE_GetDefaultAlign(void);
```

Return value

Default tab alignment for new MULTIPAGE widgets.

Add. 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	

MULTIPAGE_GetDefaultBkColor()

Description

Returns the default background color for new MULTIPAGE widgets.

Prototype

```
GUI_COLOR MULTIPAGE_GetDefaultBkColor(unsigned Index);
```

Parameter	Meaning
Index	See table below.

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	Meaning
Index	See table below.

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_GetSelection()

Description

Retruns the zero based index of the currently selected page of a MULTIPAGE widget.

Prototype

```
int MULTIPAGE_GetSelection(MULTIPAGE_Handle hObj);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.

Return value

Zero based index of the currently selected page of a MULTIPAGE widget.

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	Meaning
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of page.

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.

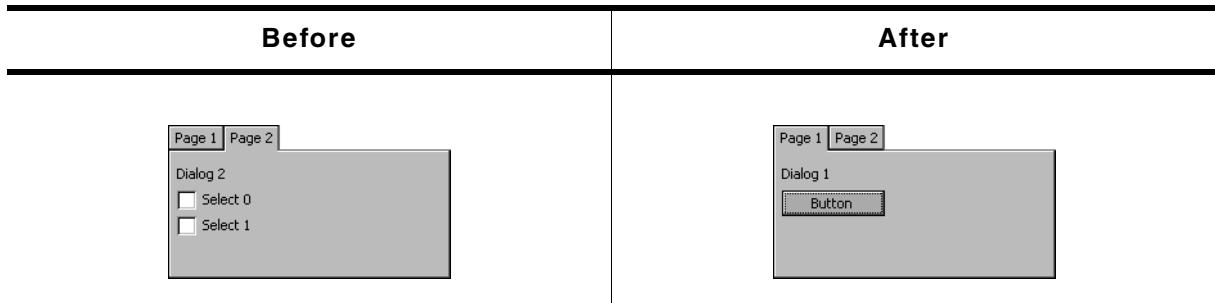
Prototype

```
int MULTIPAGE_IsPageEnabled (MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of requested page.

Return value

1 if the given page is enabled, otherwise 0.

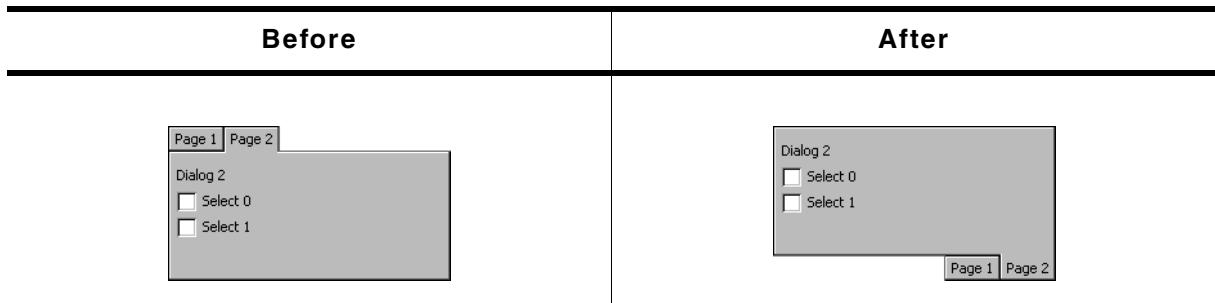
MULTIPAGE_SelectPage()**Description**

Sets the currently selected page of a MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SelectPage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of page to be selected.

MULTIPAGE_SetAlign()**Description**

Sets the tab alignment for the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetAlign(MULTIPAGE_Handle hObj, unsigned Align);
```

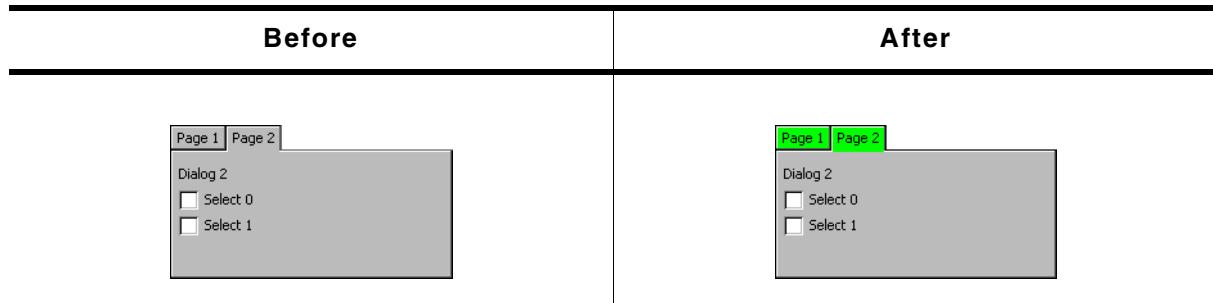
Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Align	See table below.

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..

Add. information

For more information please refer to the function [MULTIPAGE_GetDefaultAlign\(\)](#).

MULTIPAGE_SetBkColor()



Description

Sets the background color of the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetBkColor(MULTIPAGE_Handle hObj,
                           GUI_COLOR Color,
                           unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Color	Color to be used.
Index	See table below.

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.

Add. 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	Meaning
Align	Tab alignment used for new MULTIPAGE widgets.

Add. information

For more informations about the tab alignment please refer to the functions `MULTIPAGE_GetDefaultAlign()` and `MULTIPAGE_SetAlign()`.

MULTIPAGE_SetDefaultBkColor()

Description

Sets the default background color used for new MULTIPAGE widgets.

Prototype

```
void MULTIPAGE_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

Parameter	Meaning
Color	Color to be used.
Index	See table below.

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_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	Meaning
pFont	Pointer to GUI_FONT structure to be used.

Add. 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	Meaning
Color	Color to be used.
Index	See table below.

MULTIPAGE_SetFont()

Before	After

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	Meaning
hObj	Handle of MULTIPAGE widget.
pFont	Pointer to GUI_FONT structure used to display the text in the tabs.

Add. 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_SetText()

Before	After

Description

Sets the text displayed in the tab of a given page.

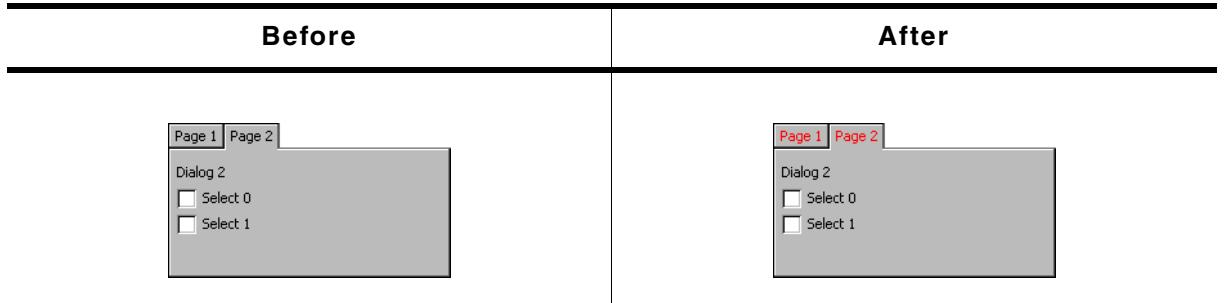
Prototype

```
void MULTIPAGE_SetText(MULTIPAGE_Handle hObj,
```

```
const char* pText, unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
pText	Pointer to the text to be displayed.
Index	Zero based index of the page.

MULTIPAGE_SetTextColor()



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	Meaning
hObj	Handle of MULTIPAGE widget.
Color	Color to be used.
Index	See table below.

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.

15.16 PROGBAR: Progress bar widget

Progress bars are commonly used in applications for visualization; for example, a tank fill-level indicator or an oil-pressure indicator. Sample 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.

15.16.1 Configuration options

Type	Macro	Default	Explanation
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	0xFFFFF	Text color, left bar.
N	PROGBAR_DEFAULT_TEXTCOLOR1	0x000000	Text color, right bar.

15.16.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.16.3 PROGBAR API

The table below lists the available emWin PROGBAR-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
PROGBAR_Create()	Create the progress bar. (Obsolete)
PROGBAR_CreateAsChild()	Create the progress bar as a child window. (Obsolete)
PROGBAR_CreateEx()	Create the progress bar.
PROGBAR_CreateIndirect()	Create the progress bar from resource table entry.
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.
PROGBAR_SetTextPos()	Set the text position (default 0,0).
PROGBAR_SetValue()	Set the value for the bar graph (and percentage if no text has been assigned).

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 xsiz, int ysize, int
Flags);
```

Parameter	Meaning
x0	Leftmost pixel of the progress bar (in parent coordinates).
y0	Topmost pixel of the progress bar (in parent coordinates).

Parameter	Meaning
xsize	Horizontal size of the progress bar (in pixels).
ysize	Vertical size of the progress bar (in pixels).
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).

Return value

Handle for the created PROGBAR widget; 0 if the routine 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	Meaning
x0	X-position of the progress bar relative to the parent window.
y0	Y-position of the progress bar relative to the parent window.
xsize	Horizontal size of the progress bar (in pixels).
ysize	Vertical size of the progress bar (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
Flags	Window create flags (see PROGBAR_Create()).

Return value

Handle for the created PROGBAR widget; 0 if the routine 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	Meaning
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 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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	(see table below)
Id	Window ID of the widget.

Permitted values for parameter ExFlags	
PROGBAR_CF_VERTICAL	A vertical progress bar will be created.
PROGBAR_CF_HORIZONTAL	A horizontal progress bar will be created.

Return value

Handle for the created widget; 0 if the routine fails.

PROGBAR_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

PROGBAR_SetBarColor()

Description

Sets the color(s) of the progress bar.

Prototype

```
void PROGBAR_SetBarColor(PROGBAR_Handle hObj,
                         unsigned int Index,
                         GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of progress bar.
Index	See table below. Other values are not permitted.
Color	Color to set (24-bit RGB value).

Permitted values for parameter Index	
0	Left portion of the progress bar.
1	Right portion of the progress bar.

PROGBAR_SetFont()

Description

Selects the font for the text display inside the progress bar.

Prototype

```
void PROGBAR_SetFont(PROGBAR_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
hObj	Handle of progress bar.
pFont	Pointer to the font.

Add. information

If this function is not called, the default font for progress bars (the GUI default font) will be used. However, the progress bar default font 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 progress bar.

Prototype

```
void PROGBAR_SetMinMax(PROGBAR_Handle hObj, int Min, int Max);
```

Parameter	Meaning
hObj	Handle of progress bar.
Min	Minimum value Range: -16383 < Min <= 16383.
Max	Maximum value Range: -16383 < Max <= 16383.

Add. 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 progress bar.

Prototype

```
void PROGBAR_SetText(PROGBAR_Handle hObj, const char* s);
```

Parameter	Meaning
hObj	Handle of progress bar.
s	Text to display. A NULL pointer is permitted; in this case a percentage value will be displayed.

Add. 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	Meaning
hObj	Handle of progress bar.
Align	Horizontal alignment attribute for the text (see table below).

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.

Add. information

If this function is not called, the default behavior is to display the text centered.

PROGBAR_SetTextPos()

Description

Sets the text position in pixels.

Prototype

```
void PROGBAR_SetTextPos(PROGBAR_Handle hObj, int XOff, int YOff);
```

Parameter	Meaning
hObj	Handle of progress bar.
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.

Add. information

The values move the text the specified number of pixels within the widget. Normally, the default of (0,0) should be sufficient.

PROGBAR_SetValue()

Description

Sets the value of the progress bar.

Prototype

```
void PROGBAR_SetValue(PROGBAR_Handle hObj, int v);
```

Parameter	Meaning
hObj	Handle of progress bar.
v	Value to set.

Add. 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:

$$p = 100\% * (v-\text{Min})/(\text{Max}-\text{Min})$$

The default value after creation of the widget is 0.

15.16.4 Examples

Using the PROGBAR widget

The following simple example demonstrates the use of the PROGBAR widget. It is available in the samples as `WIDGET_SimpleProgbar.c`:

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*****
*
*      (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
*
*      Internet: www.segger.com      Support: support@segger.com
*
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
distributed in any way. We appreciate your understanding and fairness.
-----
File       : WIDGET_SimpleProgbar.c
Purpose    : Demonstrates the use of the PROGBAR widget
-----
*/
#include "GUI.h"
#include "PROGBAR.h"
```

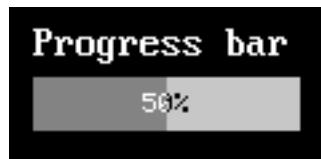
```

*****
*
*      static code
*
*****
*/
static void _DemoProgBar(void) {
    PROGBAR_Handle ahProgBar;
    int i;
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispatchString("Progress bar", 100, 80);
    /* Create progress bar */
    ahProgBar = PROGBAR_Create(100, 100, 100, 20, WM_CF_SHOW);
    GUI_Delay(500);
    /* Modify progress bar */
    for (i = 0; i <= 100; i++) {
        PROGBAR_SetValue(ahProgBar, i);
        GUI_Delay(10);
    }
    GUI_Delay(400);
    /* Delete progress bar */
    PROGBAR_Delete(ahProgBar);
    GUI_ClearRect(0, 50, 319, 239);
    GUI_Delay(750);
}

*****
*
*      MainTask
*
*      Shows the use of progress bars
*
*****
*/
void MainTask(void) {
    GUI_Init();
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispatchStringHCenterAt("WIDGET_SimpleProgbar - Sample", 160, 5);
    while(1) {
        _DemoProgBar();
    }
}

```

Screen shot of above example



Advanced use of the PROGBAR widget

This more advanced example creates a tank fill-level indicator. It is available as WIDGET_Progbar.c:

```

*****
*      SEGGER MICROCONTROLLER SYSTEME GmbH
*      Solutions for real time microcontroller applications
*****
*
*      (c) 1996 - 2004 SEGGER Microcontroller Systeme GmbH
*
*      Internet: www.segger.com      Support: support@segger.com
*
*****

```

```
***** emWin - Graphical user interface for embedded applications *****
emWin is protected by international copyright laws. Knowledge of the
source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be re-
distributed in any way. We appreciate your understanding and fairness.
-----
File      : WIDGET_Progbar.c
Purpose   : Simple demo shows the use of the PROGBAR widget
-----
*/
#include "GUI.h"
#include "PROGBAR.h"
#include <stddef.h>

/*****
*
*      static code
*
*****
*/
static void _DemoProgBar(void) {
    int i;
    PROGBAR_Handle ahProgBar[2];
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font24_ASCII);
    GUI_DispStringHCenterAt("WIDGET_Progbar - Sample", 160, 5);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringAt("Progress bar", 100,80);
    /* Create the progbars */
    ahProgBar[0] = PROGBAR_Create(100,100,100,20, WM_CF_SHOW);
    ahProgBar[1] = PROGBAR_Create( 80,150,140,10, WM_CF_SHOW);
    /* Use memory device (optional, for better looks) */
    PROGBAR_EnableMemdev(ahProgBar[0]);
    PROGBAR_EnableMemdev(ahProgBar[1]);
    PROGBAR_SetMinMax(ahProgBar[1], 0, 500);
    PROGBAR_SetFont(ahProgBar[0], &GUI_Font8x16);
    GUI_Delay(500);
    while(1) {
        PROGBAR_SetFont(ahProgBar[0], &GUI_Font8x16);
        if (LCD_GetDevCap(LCD_DEVCAP_BITSPERPIXEL) <= 4) {
            PROGBAR_SetBarColor(ahProgBar[0], 0, GUI_DARKGRAY);
            PROGBAR_SetBarColor(ahProgBar[0], 1, GUI_LIGHTGRAY);
        } else {
            PROGBAR_SetBarColor(ahProgBar[0], 0, GUI_GREEN);
            PROGBAR_SetBarColor(ahProgBar[0], 1, GUI_RED);
        }
        PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_HCENTER);
        PROGBAR_SetText(ahProgBar[0], NULL);
        for (i=0; i<=100; i++) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], i);
            GUI_Delay(5);
        }
        PROGBAR_SetText(ahProgBar[0], "Tank empty");
        for ( ; i>=0; i--) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 200-i);
            GUI_Delay(5);
        }
        PROGBAR_SetText(ahProgBar[0], "Any text... ");
        PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_LEFT);
        for ( ; i<=100; i++) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 200+i);
            GUI_Delay(5);
        }
        PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_RIGHT);
        for ( ; i>=0; i--) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 400-i);
            GUI_Delay(5);
        }
        PROGBAR_SetFont(ahProgBar[0], &GUI_FontComic18B_1);
    }
}
```

```
PROGBAR_SetText(ahProgBar[0], "Any font...");  
for ( ; i<=100; i++) {  
    PROGBAR_SetValue(ahProgBar[0], i);  
    PROGBAR_SetValue(ahProgBar[1], 400+i);  
    GUI_Delay(5);  
}  
GUI_Delay(500);  
}  
  
/* ***** */  
*  
*      MainTask  
*  
*      Demonstrates the use of the PROGBAR widget  
*  
***** */  
  
void MainTask(void) {  
    GUI_Init();  
    while (1) {  
        _DemoProgBar();  
    }  
}
```

Screen shot of above example



15.17 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		

15.17.1 Configuration options

Type	Macro	Default	Explanation
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 radio button text.
N	RADIO_DEFAULT_TEXT_COLOR	GUI_BLACK	Default text color of radio button text.
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.

15.17.2 Notification codes

The following events are sent from a radio button widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
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 off the button without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Value (selection) of the radio button widget has changed.

15.17.3 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.

15.17.4 RADIO API

The table below lists the available emWin RADIO-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>RADIO_Create()</code>	Create a group of radio buttons. (Obsolete)
<code>RADIO_CreateEx()</code>	Create a group of radio buttons.
<code>RADIO_CreateIndirect()</code>	Create a group of radio buttons from resource table entry.
<code>RADIO_Dec()</code>	Decrement the button selection by a value of 1.
<code>RADIO_GetDefaultFont()</code>	Returns the default font used to show the text of new radio buttons.
<code>RADIO_GetDefaultTextColor()</code>	Returns the default text color used to show the text of new radio buttons.
<code>RADIO_GetText()</code>	Returns the text of a radio button item.
<code>RADIO_GetValue()</code>	Return the current button selection.
<code>RADIO_Inc()</code>	Increment the button selection by a value of 1.
<code>RADIO_SetBkColor()</code>	Sets the background color of the radio button.
<code>RADIO_SetDefaultFocusColor()</code>	Sets the default focus rectangle color for new radio buttons.
<code>RADIO_SetDefaultFont()</code>	Sets the default font used to show the text of new radio buttons.
<code>RADIO_SetDefaultImage()</code>	Sets the images to be used for new radio buttons.
<code>RADIO_SetDefaultTextColor()</code>	Sets the default text color used to show the text of new radio buttons.
<code>RADIO_SetFocusColor()</code>	Sets the color of the focus rectangle.
<code>RADIO_SetFont()</code>	Sets the font used to show the text of the radio button.
<code>RADIO_SetGroupId()</code>	Sets the group Id of the given radio widget.
<code>RADIO_SetImage()</code>	Sets the images used to display the radio button.
<code>RADIO_SetText()</code>	Sets the text
<code>RADIO_SetTextColor()</code>	Sets the text color used to show the text of radio button.
<code>RADIO_SetValue()</code>	Set the button selection.

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	Meaning
<code>x0</code>	Leftmost pixel of the radio button widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the radio button widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the radio button widget (in pixels).

Parameter	Meaning
<code>ysize</code>	Vertical size of the radio button widget (in pixels).
<code>hParent</code>	Handle of parent window.
<code>Id</code>	ID to be returned.
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>Para</code>	Number of buttons in the group.

Return value

Handle for the created RADIO widget; 0 if the routine 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	Meaning
<code>x0</code>	Leftmost pixel of the widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the widget (in pixels).
<code>ysize</code>	Vertical size of the widget (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new RADIO widget will be a child of the desktop (top-level window).
<code>WinFlags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>ExFlags</code>	Not used, reserved for future use.
<code>Id</code>	Window ID of the widget.
<code>NumItems</code>	Number of items of the radio widget. (default is 2)
<code>Spacing</code>	Number of vertical pixels used for each item of the radio widget.

Return value

Handle for the created widget; 0 if the routine fails.

Add. 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()

Prototype explained at the beginning of the chapter. The element `Flags` of the resource passed as parameter is not used.

The following table shows the use of the resource element `Para`:

Bits	Meaning
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.

RADIO_Dec()

Before	After

Description

Decrements the selection by a value of 1.

Prototype

```
void RADIO_Dec(RADIO_Handle hobj);
```

Parameter	Meaning
hObj	Handle of radio button widget.

Add. information

Please 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.

Add. information

For information about how to add text to a radio widget please refer to the function [RADIO_SetText\(\)](#).

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.

Add. information

For information about how to add text to a radio widget please refer to the function [RADIO_SetText\(\)](#).

RADIO_GetText()

Description

Returns the optional text of the given radio button.

Prototype

```
int RADIO_GetText(RADIO_Handle hObj, unsigned Index,
                  char * pBuffer, int MaxLen);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Index of the desired item.
pBuffer	Pointer to buffer to which the text will be copied.
MaxLen	Buffer size in bytes.

Return value

Length of the text copied into the buffer.

Add. information

If the desired item of the radio button contains no text the function returns 0 and the buffer remains unchanged.

RADIO_GetValue()

Description

Returns the current button selection.

Prototype

```
void RADIO_GetValue(RADIO_Handle hObj);
```

Parameter	Meaning
hObj	Handle of radio button widget.

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.

Add. information

For information about how to use groups of radio buttons please refer to the function `RADIO_SetGroupID()`.

RADIO_Inc()

Before	After

Description

Increments the selection by a value of 1.

Prototype

```
void RADIO_Inc(RADIO_Handle hObj);
```

Parameter	Meaning
hObj	Handle of radio button widget.

Add. information

Please 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

Description

Sets the background color of the radio widget.

Prototype

```
void RADIO_SetBkColor(RADIO_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of radio button widget.
Color	Color to be used for the background. (range 0x000000 and 0xFFFF or a valid color define) GUI_INVALID_COLOR to make background transparent

Add. 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 contents 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.

RADIO_SetDefaultFocusColor()

Description

Sets the default focus rectangle color for new radio buttons.

Prototype

```
GUI_COLOR RADIO_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Default color to be used for new radio buttons.

Return value

Previous default focus rectangle color.

Add. information

For more information please refer to the function `RADIO_SetFocusColor()`.

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	Meaning
<code>pFont</code>	Pointer to <code>GUI_FONT</code> structure used to show the text of new radio widgets.

Add. information

For information about how to add text to a radio widget please refer to the function `RADIO_SetText()`.

RADIO_SetDefaultImage()

Description

Sets the images used to draw new radio buttons.

Prototype

```
void RADIO_SetDefaultImage(const GUI_BITMAP * pBitmap, unsigned int Index);
```

Parameter	Meaning
<code>pBitmap</code>	Pointer to the bitmap.
<code>Index</code>	(see table below)

Permitted values for parameter <code>Index</code>	
<code>RADIO_BI_INACTIV</code>	Outer image used to show a disabled radio button.
<code>RADIO_BI_ACTIV</code>	Outer image used to show a enabled radio button.
<code>RADIO_BI_CHECK</code>	Inner image used to mark the selected item.

Add. 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_INACTIV`. 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	Meaning
<code>TextColor</code>	New color to be used.

Add. information

For information about how to add text to a radio widget please refer to the function `RADIO_SetText()`.

RADIO_SetFocusColor()

Before	After

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	Meaning
<code>hObj</code>	Handle of widget.
<code>Color</code>	Color to be used for the focus rectangle.

Return value

Previous color of the focus rectangle.

Add. information

The focus rectangle is only visible if the widget has the input focus.

RADIO_SetFont()

Before	After

Description

Sets the font used to display the optional text next to the radio button.

Prototype

```
void RADIO_SetFont(RADIO_Handle hObj, const GUI_FONT * pFont);
```

Parameter	Meaning
<code>hObj</code>	Handle of radio button widget.
<code>pFont</code>	Pointer to <code>GUI_FONT</code> structure to be used to display the text.

Add. information

For information about how to add text to a radio widget please refer to the function `RADIO_SetText()`.

RADIO_SetGroupID()

Before	After
<input checked="" type="radio"/> Red <input checked="" type="radio"/> Magenta <input type="radio"/> Green <input type="radio"/> Cyan <input type="radio"/> Blue <input type="radio"/> Yellow	<input checked="" type="radio"/> Red <input type="radio"/> Magenta <input type="radio"/> Green <input type="radio"/> Cyan <input type="radio"/> Blue <input type="radio"/> Yellow

Description

Sets the group ID of the radio widget.

Prototype

```
void RADIO_SetGroupID(RADIO_Handle hObj, U8 GroupID);
```

Parameter	Meaning
<code>hObj</code>	Handle of radio button widget.
<code>GroupID</code>	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.

Add. 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 sample 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	Meaning
<code>hObj</code>	Handle of radio button widget.
<code>pBitmap</code>	Pointer to the bitmap. (see table shown under RADIO_SetDefaultImage)

Add. information

(see `RADIO_SetDefaultImage`).

`RADIO_SetText()`

Before	After
	

Description

Sets the optional text shown next to the radio buttons.

Prototype

```
void RADIO_SetText(RADIO_Handle hObj, const char * pText, unsigned Index);
```

Parameter	Meaning
<code>hObj</code>	Handle of radio button widget.
<code>pText</code>	Pointer to the text to be shown next to the specified radio button.
<code>Index</code>	Zero based index of the radio button.

Add. information

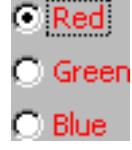
If using a `RADIO` widget without text (old style) the focus rectangle is drawn around the buttons of the widget. If using radio button text the focus rectangle is shown around the text of the currently selected radio button of the widget.

Example

The following sample 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
	

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	Meaning
hObj	Handle of radio button widget.
Color	Color used to show the text.

Add. information

For information about how to add text to a radio widget please refer to the function `RADIO_SetText()`.

RADIO_SetValue()

Description

Sets the current button selection.

Prototype

```
void RADIO_SetValue(RADIO_Handle hObj, int v);
```

Parameter	Meaning
hObj	Handle of radio button widget.
v	Value to be set.

Add. 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.

15.18 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.

15.18.1 Configuration options

Type	Macro	Default	Explanation
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.

15.18.2 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	Explanation
WM_NOTIFICATION_CLICKED	Scrollbar has been clicked.
WM_NOTIFICATION_RELEASED	Scrollbar has been released.
WM_NOTIFICATION_SCROLLBAR_ADDED	Scroll bar 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 scroll bar has changed, either by moving the thumb or by pressing the arrow buttons.

15.18.3 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 scroll bar by a value of 1.
GUI_KEY_DOWN	Increments the current value of the scroll bar by a value of 1.
GUI_KEY_LEFT	Decrements the current value of the scroll bar by a value of 1.
GUI_KEY_UP	Decrements the current value of the scroll bar by a value of 1.

15.18.4 SCROLLBAR API

The table below lists the available emWin SCROLLBAR-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
SCROLLBAR_AddValue()	Increment or decrement the value of the scroll bar by a specified value.
SCROLLBAR_Create()	Create the scroll bar. (Obsolete)
SCROLLBAR_CreateAttached()	Create a scroll bar attached to a window.
SCROLLBAR_CreateEx()	Create the scroll bar.

Routine	Explanation
<code>SCROLLBAR_CreateIndirect()</code>	Creates the scroll bar from resource table entry.
<code>SCROLLBAR_Dec()</code>	Decrements the value of the scroll bar by a value of 1.
<code>SCROLLBAR_GetDefaultWidth()</code>	Returns the default width of a scroll bar.
<code>SCROLLBAR_GetValue()</code>	Returns the current item value.
<code>SCROLLBAR_Inc()</code>	Increments the value of the scroll bar by a value of 1.
<code>SCROLLBAR_SetColor()</code>	Sets the color of a scroll bar.
<code>SCROLLBAR_SetDefaultColor()</code>	Sets the default colors for new scroll bars.
<code>SCROLLBAR_SetDefaultWidth()</code>	Sets the default width of a scroll bar.
<code>SCROLLBAR_SetNumItems()</code>	Sets the number of items for scrolling.
<code>SCROLLBAR_SetPageSize()</code>	Sets the page size (in number of items).
<code>SCROLLBAR_SetState()</code>	Sets the state of a scroll bar.
<code>SCROLLBAR_SetValue()</code>	Sets the current value of the scroll bar.
<code>SCROLLBAR_SetWidth()</code>	Sets the width of the scroll bar.

SCROLLBAR_AddValue()

Definition

Increments or decrements the value of the scroll bar by a specified value.

Prototype

```
void SCROLLBAR_AddValue(SCROLLBAR_Handle hObj, int Add);
```

Parameter	Meaning
<code>hObj</code>	Handle of scroll bar.
<code>Add</code>	Number of items to increment or decrement at one time.

Add. information

The scroll bar 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	Meaning
<code>x0</code>	Leftmost pixel of the scroll bar (in parent coordinates).
<code>y0</code>	Topmost pixel of the scroll bar (in parent coordinates).
<code>xsize</code>	Horizontal size of the scroll bar (in pixels).
<code>ysize</code>	Vertical size of the scroll bar (in pixels).
<code>hParent</code>	Handle of parent window.

Parameter	Meaning
<code>Id</code>	ID to be returned.
<code>WinFlags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>SpecialFlags</code>	Special creation flags (see indirect creation flags under <code>SCROLLBAR_CreateIndirect()</code>).

Return value

Handle for the created SCROLLBAR widget; 0 if the routine 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	Meaning
<code>hParent</code>	Handle of parent window.
<code>SpecialFlags</code>	Special creation flags (see indirect creation flags under <code>SCROLLBAR_CreateIndirect()</code>).

Return value

Handle for the created scrollbar; 0 if the routine fails.

Add. information

An attached scroll bar is essentially a child window which will position itself on the parent window and operate accordingly.

Vertical attached scrollbars will be automatically placed on the right side of the parent window; horizontal scrollbars on the bottom. Since no more than one horizontal and one vertical scroll bar 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 scroll bar is created:

`GUI_ID_HSCROLL` for a horizontal scroll bar, and
`GUI_ID_VSCROLL` for a vertical scroll bar.

Example

Creates a list box with an attached scrollbar:

```
LISTBOX_Handle hListBox;
hListBox = LISTBOX_Create(ListBox, 50, 50, 100, 100, WM_CF_SHOW);
SCROLLBAR_CreateAttached(hListBox, SCROLLBAR_CF_VERTICAL);
```

Screen shots 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 scrollbar:



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	Meaning
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 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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	Special creation flags (see indirect creation flags under SCROLLBAR_CreateIndirect()).
Id	Window ID of the widget.

Return value

Handle for the created widget; 0 if the routine fails.

SCROLLBAR_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flags may be used as the Flags element of the resource passed as parameter:

Permitted indirect creation flags ("OR" combinable)	
SCROLLBAR_CF_VERTICAL	Creates a vertical scroll bar (default is horizontal).
SCROLLBAR_CF_FOCUSSABLE	Gives scroll bar the input focus.

The Para element is not used in the resource table.

SCROLLBAR_Dec()**Description**

Decrements the current value of the scroll bar by a value of 1.

Prototype

```
void SCROLLBAR_Dec(SCROLLBAR_Handle hObj);
```

Parameter	Meaning
hObj	Handle of scroll bar.

Add. 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.

Prototype

```
int SCROLLBAR_GetDefaultWidth(void);
```

Return value

Default width used to create a scrollbar.

SCROLLBAR_GetValue()**Description**

Return the value of the current item.

Prototype

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

Parameter	Meaning
hObj	Handle of scroll bar.

Return value

The value of the current item.

SCROLLBAR_Inc()**Description**

Increments the current value of the scroll bar by a value of 1.

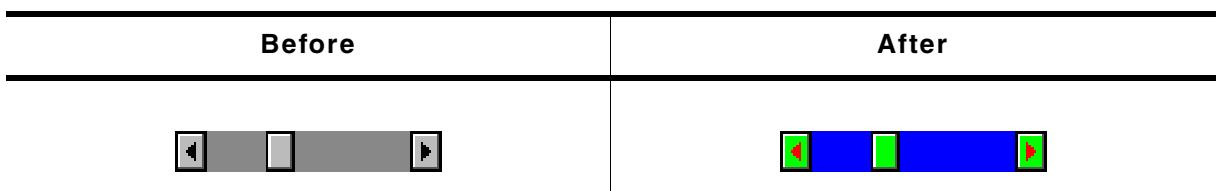
Prototype

```
void SCROLLBAR_Inc(SCROLLBAR_Handle hObj);
```

Parameter	Meaning
hObj	Handle of scroll bar.

Add. 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()**Description**

Sets the given color attribute of the scroll bar.

Prototype

```
GUI_COLOR SCROLLBAR_SetColor(SCROLLBAR_Handle hObj,
                               int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of scroll bar.
Index	(see table below)
Color	Color to be used.

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 scroll bars.

Prototype

```
GUI_COLOR SCROLLBAR_SetDefaultColor(GUI_COLOR Color, unsigned int Index);
```

Parameter	Meaning
Color	Color used as default for new scroll bars.
Index	(see table under SCROLLBAR_SetColor())

Return value

Previous default color.

SCROLLBAR_SetDefaultWidth()

Description

Sets the default width used to create a scrollbar.

Prototype

```
int SCROLLBAR_SetDefaultWidth(int DefaultWidth);
```

Parameter	Meaning

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	Meaning
hObj	Handle of scroll bar.
NumItems	Number of items to be set.

Add. information

The definition of an "item" is application-specific, although in most cases it is equal to one line.

The number of items specified is the maximum value; the scroll bar cannot go beyond this value.

SCROLLBAR_SetPageSize()

Description

Sets the page size.

Prototype

```
void SCROLLBAR_SetPageSize(SCROLLBAR_Handle hObj, int PageSize);
```

Parameter	Meaning
hObj	Handle of scroll bar.
PageSize	Page size (in number of items).

Add. 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 scroll bar area, the window will be scrolled up or down by the number of items specified to be one page.

SCROLLBAR_SetState()

Description

Sets the state of a scroll bar.

Prototype

```
void SCROLLBAR_SetState(SCROLLBAR_Handle hObj, const WM_SCROLL_STATE*
                        pState);
```

Parameter	Meaning
hObj	Handle of scroll bar.
pState	Pointer to a data structure of type WM_SCROLL_STATE.

Add. information

The data structure is defined as follows:

```
typedef struct {
    int NumItems;
    int v;
    int PageSize;
} WM_SCROLL_STATE;
```

SCROLLBAR_SetValue()

Description

Sets the current value of a scroll bar.

Prototype

```
void SCROLLBAR_SetValue(SCROLLBAR_Handle hObj, int v);
```

Parameter	Meaning
hObj	Handle of scroll bar.
v	Value to be set.

SCROLLBAR_SetWidth()

Description

Sets the width of the scroll bar.

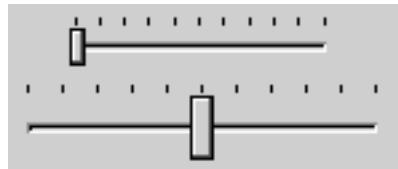
Prototype

```
void SCROLLBAR_SetWidth(SCROLLBAR_Handle hObj, int Width);
```

Parameter	Meaning
hObj	Handle of scroll bar.
Width	Width to be set.

15.19 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`.

15.19.1 Configuration options

Type	Macro	Default	Explanation
N	<code>SLIDER_BKCOLOR0_DEFAULT</code>	0xc0c0c0	Background color.
N	<code>SLIDER_COLOR0_DEFAULT</code>	0xc0c0c0	Slider (thumb) color.
N	<code>SLIDER_FOCUSCOLOR_DEFAULT</code>	GUI_BLACK	Default color for rendering the focus rectangle.

15.19.2 Notification codes

The following events are sent from a slider widget to its parent window as part of a `WM_NOTIFY_PARENT` message:

Message	Explanation
<code>WM_NOTIFICATION_CLICKED</code>	Slider widget has been clicked.
<code>WM_NOTIFICATION_RELEASED</code>	Slider widget has been released.
<code>WM_NOTIFICATION_VALUE_CHANGED</code>	Value of the slider widget has changed by moving the thumb.

15.19.3 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
<code>GUI_KEY_RIGHT</code>	Increments the current value of the slider bar by one item.
<code>GUI_KEY_LEFT</code>	Decrements the current value of the slider bar by one item.

15.19.4 SLIDER API

The table below lists the available emWin SLIDER-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>SLIDER_Create()</code>	Create the slider. (Obsolete)
<code>SLIDER_CreateEx()</code>	Create the slider.
<code>SLIDER_CreateIndirect()</code>	Create the slider from resource table entry.
<code>SLIDER_Dec()</code>	Decrement the value of the slider bar.
<code>SLIDER_GetValue()</code>	Return the current value of the slider bar.
<code>SLIDER_Inc()</code>	Increment the value of the slider bar.
<code>SLIDER_SetBkColor()</code>	Sets the background color of the slider bar.
<code>SLIDER_SetDefaultFocusColor()</code>	Sets the default focus rectangle color for new slider bars.
<code>SLIDER_SetFocusColor()</code>	Sets the color of the focus rectangle.

Routine	Explanation
<code>SLIDER_SetNumTicks()</code>	Sets the number of tick marks of the slider bar.
<code>SLIDER_SetRange()</code>	Set the range of the slider value.
<code>SLIDER_SetValue()</code>	Set the current value of the slider bar.
<code>SLIDER_SetWidth()</code>	Set the width of the slider bar.

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	Meaning
<code>x0</code>	Leftmost pixel of the slider (in parent coordinates).
<code>y0</code>	Topmost pixel of the slider (in parent coordinates).
<code>xsize</code>	Horizontal size of the slider (in pixels).
<code>ysize</code>	Vertical size of the slider (in pixels).
<code>hParent</code>	Handle of the parent window.
<code>Id</code>	Id to be returned
<code>WinFlags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 14: "The Window Manager" for a list of available parameter values).
<code>SpecialFlags</code>	Special creation flag (see indirect creation flag under <code>SLIDER_CreateIndirect()</code>).

Return value

Handle for the created SLIDER widget; 0 if the routine 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	Meaning
<code>x0</code>	Leftmost pixel of the widget (in parent coordinates).
<code>y0</code>	Topmost pixel of the widget (in parent coordinates).
<code>xsize</code>	Horizontal size of the widget (in pixels).
<code>ysize</code>	Vertical size of the widget (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new SLIDER widget will be a child of the desktop (top-level window).

Parameter	Meaning
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	Special creation flags (see indirect creation flags under SLIDER_CreateIndirect()).
Id	Window ID of the widget.

Return value

Handle for the created widget; 0 if the routine fails.

SLIDER_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flag may be used as the Flags element of the resource passed as parameter:

Permitted indirect creation flag	
SLIDER_CF_VERTICAL	Create a vertical slider (default is horizontal).

The Para element is not used in the resource table.

SLIDER_Dec()

Description

Decrements the current value of the slider bar by one item.

Prototype

```
void SLIDER_Dec(SLIDER_Handle hObj);
```

Parameter	Meaning
hObj	Handle of slider widget.

SLIDER_GetValue()

Description

Returns the current value of the slider bar.

Prototype

```
int SLIDER_GetValue(SLIDER_Handle hObj);
```

Parameter	Meaning
hObj	Handle of slider widget.

Return value

The current value of the slider.

SLIDER_Inc()

Description

Increments the current value of the slider bar by one item.

Prototype

```
void SLIDER_Inc(SLIDER_Handle hObj);
```

Parameter	Meaning
hObj	Handle of slider widget.

SLIDER_SetBkColor()

Description

Sets the background color of the slider.

Prototype

```
void SLIDER_SetBkColor(SLIDER_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of slider widget.
Color	Color to be used for the background. (range 0x000000 and 0xFFFF or a valid color define) GUI_INVALID_COLOR to make background transparent

Add. 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 contents 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 bars.

Prototype

```
GUI_COLOR SLIDER_SetDefaultFocusColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Default color to be used for new slider bars.

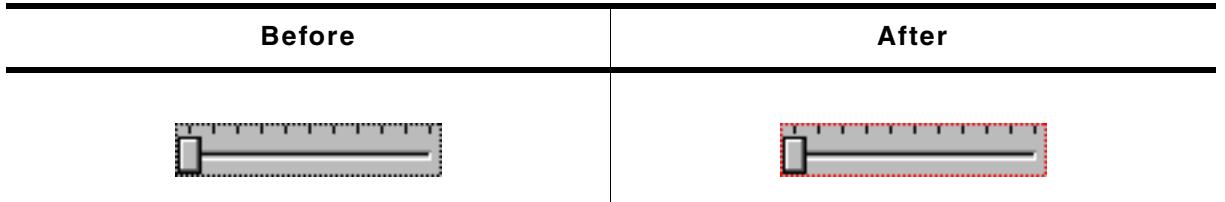
Return value

Previous default focus rectangle color.

Add. information

For more information please refer to the function `SLIDER_SetFocusColor()`.

SLIDER_SetFocusColor()



Description

Sets the color used to render the focus rectangle of the slider bar.

Prototype

```
GUI_COLOR SLIDER_SetFocusColor(SLIDER_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Color	Color to be used for the focus rectangle.

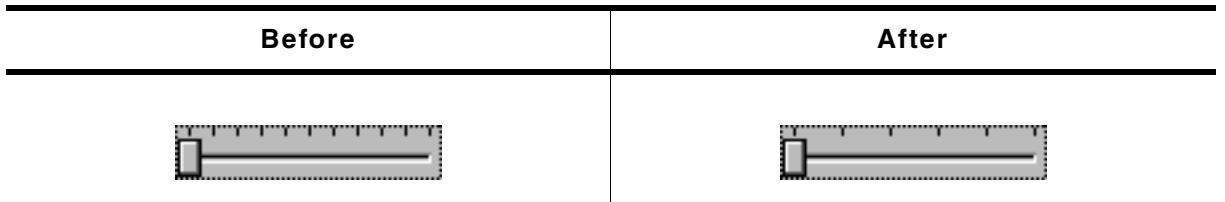
Return value

Previous color of the focus rectangle.

Add. information

The focus rectangle is only visible if the widget has the input focus.

SLIDER_SetNumTicks()



Description

Sets the number of tick marks of the slider bar.

Prototype

```
void SLIDER_SetNumTicks(SLIDER_Handle hObj, int NumTicks);
```

Parameter	Meaning
hObj	Handle of slider widget.
NumTicks	Number of tick marks drawn.

Add. 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.

Prototype

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

Parameter	Meaning
hObj	Handle of slider widget.
Min	Minimum value.
Max	Maximum value.

Add. 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_SetValue()

Description

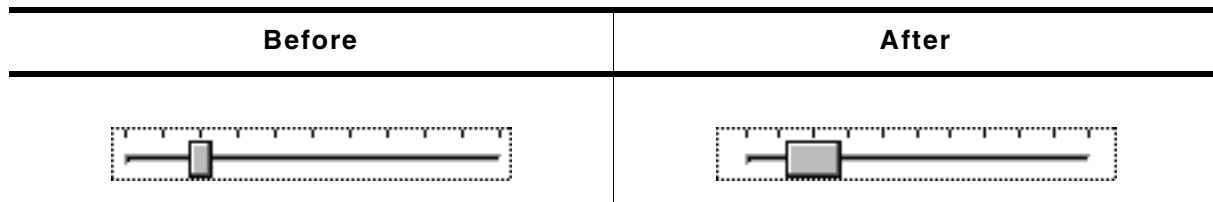
Sets the current value of the slider bar.

Prototype

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

Parameter	Meaning
hObj	Handle of slider widget.
v	Value to be set.

SLIDER_SetWidth()



Description

Sets the width of the slider bar.

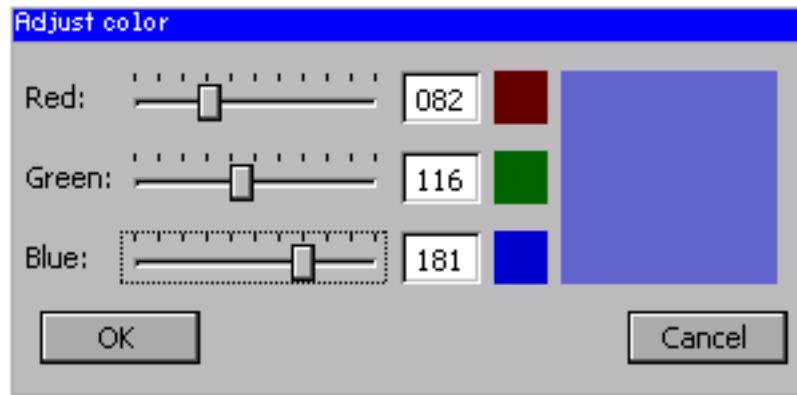
Prototype

```
void SLIDER_SetWidth(SLIDER_Handle hObj, int Width);
```

Parameter	Meaning
hObj	Handle of slider widget.
Width	Width to be set.

15.19.5 Example

The source of the following sample is available as DIALOG_SliderColor.c in the samples shipped with emWin:

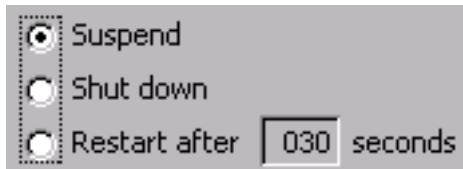


15.20 TEXT: Text widget

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`.

15.20.1 Configuration options

Type	Macro	Default	Explanation
S	<code>TEXT_FONT_DEFAULT</code>	&GUI_Font13_1	Font used.

15.20.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.20.3 TEXT API

The table below lists the available emWin TEXT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>TEXT_Create()</code>	Creates the text widget. (Obsolete)
<code>TEXT_CreateAsChild()</code>	Creates the text widget as a child window. (Obsolete)
<code>TEXT_CreateEx()</code>	Creates the text widget.
<code>TEXT_CreateIndirect()</code>	Creates the text widget from resource table entry.
<code>TEXT_GetDefaultFont()</code>	Returns the default font used for text.
<code>TEXT_SetBkColor()</code>	Sets the background color for the text.
<code>TEXT_SetDefaultFont()</code>	Sets the default font used for text.
<code>TEXT_SetDefaultTextColor()</code>	Sets the default text color used for text.
<code>TEXT_SetFont()</code>	Sets the font used for a specified text widget.
<code>TEXT_SetText()</code>	Sets the text for a specified text widget.
<code>TEXT_SetTextAlign()</code>	Sets the text alignment of a specified text widget.
<code>TEXT_SetTextColor()</code>	Sets the text color of the given widget.

`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	Meaning
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.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" 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()).

Return value

Handle for the created TEXT widget; 0 if the routine 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	Meaning
x0	X-position of the progress bar relative to the parent window.
y0	Y-position of the progress bar 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()).

Return value

Handle for the created TEXT widget; 0 if the routine 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	Meaning
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 (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
ExFlags	Alignment attribute for the text (see indirect creation flags under TEXT_CreateIndirect()).
Id	Window ID of the widget.
pText	Pointer to the text to be displayed.

Return value

Handle for the created widget; 0 if the routine fails.

TEXT_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flags may be used as the Flags element of the resource passed as parameter:

Permitted indirect creation flags ("OR" combinable)	
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

The Para element is not used in the resource table.

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_SetBkColor()

Description

Sets the background color of the text widget.

Prototype

```
void TEXT_SetBkColor(TEXT_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of text widget.
Color	Color to be used for the background. (range 0x000000 and 0xFFFF or a valid color define) GUI_INVALID_COLOR to make background transparent

Add. 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 contents 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	Meaning
pFont	Pointer to the font to be set as default.

TEXT_SetDefaultTextColor()

Description

Sets the default text color used for text widgets.

Prototype

```
void TEXT_SetDefaultTextColor(GUI_COLOR Color);
```

Parameter	Meaning
Color	Color to be used.

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	Meaning
hObj	Handle of text widget.
pFont	Pointer to the font to be used.

TEXT_SetText()

Description

Sets the text to be used for a specified text widget.

Prototype

```
void TEXT_SetText(TEXT_Handle hObj, const char* s);
```

Parameter	Meaning
hObj	Handle of text widget.
s	Text to be displayed.

TEXT_SetTextAlign()**Description**

Sets the text alignment of a specified text widget.

Prototype

```
void TEXT_SetTextAlign(TEXT_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of text widget.
Align	Text alignment (see TEXT_Create()).

TEXT_SetTextColor()**Description**

Sets the text color of a specified text widget.

Prototype

```
void TEXT_SetTextColor(TEXT_Handle pObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of text widget.
Color	New text color.

15.21 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.

15.21.1 Configuration options

Type	Macro	Default	Explanation
S	WINDOW_BKCOLOR_DEFAULT	0xC0C0C0	Default background color for new WINDOW widgets

15.21.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.21.3 WINDOW API

The table below lists the available emWin WINDOW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
WINDOW_CreateIndirect()	Creates the window widget from a resource table entry.
WINDOW_SetDefaultBkColor()	Sets the default background color for new WINDOW widgets.

WINDOW_CreateIndirect()

Prototype explained at the beginning of the chapter. The sample folder contains the file `WIDGET_Window.c` which shows how to use the WINDOW widget in a dialog resource.

WINDOW_SetDefaultBkColor()

Description

Sets the default background color used for WINDOW widgets.

Prototype

```
void WINDOW_SetDefaultBkColor(GUI_COLOR Color);
```

Parameter	Meaning
<code>Color</code>	Color to be used.

Chapter 16

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 to the user) and an "OK" button.

16.1 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.

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.

Dialog procedure

A dialog box is a window, and it receives messages just like all other windows in the system do. Most messages are handled by the window callback routine of the dialog box automatically; the others are passed to the callback routine specified upon creation of the dialog box, which is known as the dialog procedure.

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.

16.2 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.

16.2.1 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. This particular sample was created manually, although it could also be done by a GUI-builder:

```

static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
{ FRAMEWIN_CreateIndirect, "Dialog", 0, 10, 10, 180, 230, FRAMEWIN_CF_MOVEABLE, 0 },
{ BUTTON_CreateIndirect, "OK", GUI_ID_OK, 100, 5, 60, 20 },
{ BUTTON_CreateIndirect, "Cancel", GUI_ID_CANCEL, 100, 30, 60, 20 },
{ TEXT_CreateIndirect, "LText", 0, 10, 55, 48, 15, TEXT_CF_LEFT },
{ TEXT_CreateIndirect, "RText", 0, 10, 80, 48, 15, TEXT_CF_RIGHT },
{ 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 },
{ EDIT_CreateIndirect, NULL, GUI_ID_EDIT2, 60, 100, 100, 15, 0, 6 },
{ TEXT_CreateIndirect, "Bin", 0, 10, 120, 48, 15, TEXT_CF_RIGHT },
{ EDIT_CreateIndirect, NULL, GUI_ID_EDIT3, 60, 120, 100, 15 },
{ LISTBOX_CreateIndirect, NULL, GUI_ID_LISTBOX0, 10, 10, 48, 40 },
{ CHECKBOX_CreateIndirect, NULL, GUI_ID_CHECK0, 10, 140, 0, 0 },
{ CHECKBOX_CreateIndirect, NULL, GUI_ID_CHECK1, 30, 140, 0, 0 },
{ SLIDER_CreateIndirect, NULL, GUI_ID_SLIDER0, 60, 140, 100, 20 },
{ SLIDER_CreateIndirect, NULL, GUI_ID_SLIDER1, 10, 170, 150, 30 }
};

```

Any widget to be included in a dialog box must be created indirectly with the <WIDGET>_CreateIndirect() function. For more information, please refer to Chapter 15: "Window Objects (Widgets)".

16.2.2 Dialog procedure

The sample 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:

```

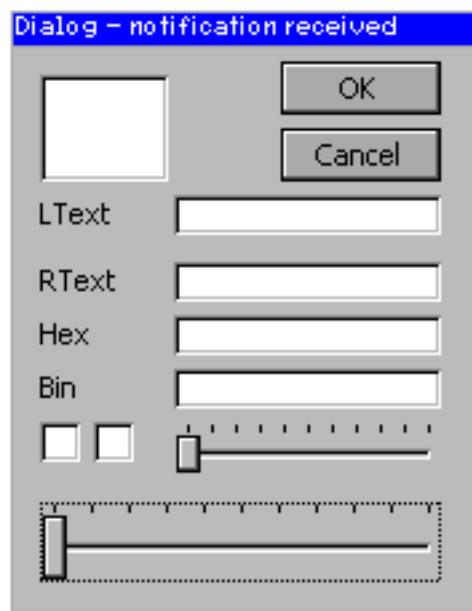
*****
*
*      Dialog procedure
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
    switch (pMsg->MsgId) {
        default:
            WM_DefaultProc(pMsg);
    }
}

```

For this sample, 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):



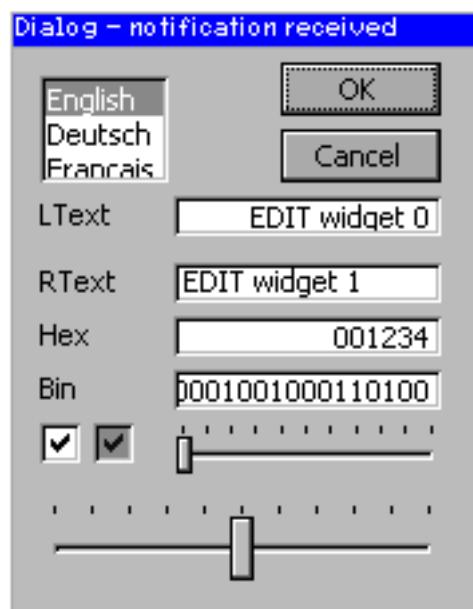
After creation of the dialog box, all widgets included in the resource table will be visible, although as can be seen in the previous screen shot, they will appear "empty". This is because the dialog procedure does not yet contain code that initializes the individual elements. The initial values of the widgets, the actions caused by them, and the interactions between them need to be defined in the dialog procedure.

16.2.2.1 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) {
    int NCode, Id;
    WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
    WM_HWIN 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;
        default:
            WM_DefaultProc(pMsg);
    }
}
```

The initialized dialog box now appears as follows, with all widgets containing their initial values:



16.2.2.2 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) {
    int NCode, Id;
    WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
    WM_HWIN 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);
    }
}
```

For further details, this entire example is available as `Dialog.c` in the samples shipped with emWin.

16.3 Dialog API

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

Routine	Explanation
Dialog boxes	
GUI_CreateDialogBox()	Create a non-blocking dialog.
GUI_ExecCreatedDialog()	Executes an already created dialog.
GUI_ExecDialogBox()	Create and execute a dialog.
GUI_EndDialog()	End a dialog box.
Message boxes	
GUI_MessageBox()	Creates and executes a message box.
MESSAGEBOX_Create()	Creates a message box.

16.4 Dialog boxes

GUI_CreateDialogBox()

Description

Creates a non-blocking 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	Meaning
<code>paWidget</code>	Pointer to resource table defining the widgets to be included in the dialog.
<code>NumWidgets</code>	Total number of widgets included in the dialog.
<code>cb</code>	Pointer to an application-specific callback function (dialog procedure).
<code>hParent</code>	Handle of parent window (0 = no parent window).
<code>x0</code>	X-position of the dialog relative to parent window.
<code>y0</code>	Y-position of the dialog relative to parent window.

GUI_ExecCreatedDialog()

Description

Executes an already created dialog box.

Prototype

```
int GUI_ExecCreatedDialog(WM_HWIN hDialog);
```

Parameter	Meaning
<code>hDialog</code>	Handle to dialog box.

Return value

Value returned from `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	Meaning
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.

Return value

Value returned from GUI_EndDialog.

GUI_EndDialog()

Description

Ends (closes) a dialog box.

Prototype

```
void GUI_EndDialog(WM_HWIN hDialog, int r);
```

Parameter	Meaning
hDialog	Handle to dialog box.
r	Value to be returned by GUI_ExecDialogBox.

Return value

Specifies the value to be returned to the calling thread from the function that created the dialog box (typically only relevant with GUI_ExecDialogBox()).

With non-blocking dialogs, there is no application thread waiting and the return value is ignored.

Chapter 17

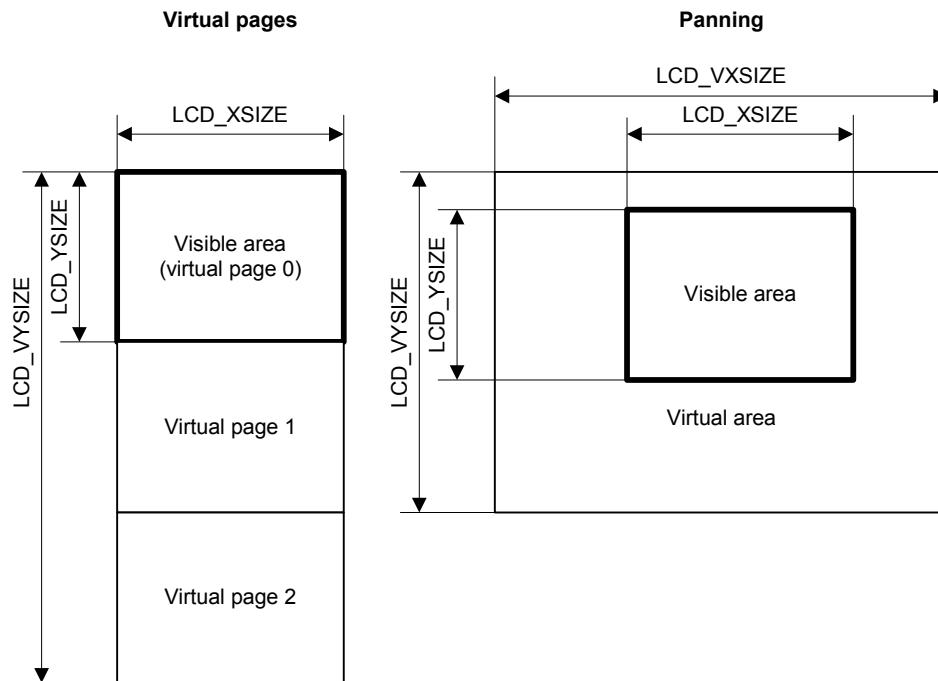
Virtual screen / Virtual pages

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.

17.1 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.

17.2 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 display start address or it has a command to set the upper left display start position.

17.3 Configuration

The virtual screen support configuration should be done in the file `LCDConf.h`. The table below shows all available configuration macros:

Type	Macro	Default	Explanation
F	<code>LCD_SET_ORG</code>	---	Macro used to set the display start position of the upper left corner.
N	<code>LCD_VXSIZE</code>	<code>LCD_XSIZE</code>	Horizontal resolution of virtual display.
N	<code>LCD_VYSIZE</code>	<code>LCD_YSIZE</code>	Vertical resolution of virtual display.

`LCD_SET_ORG`

Description

This macro is used by the display driver to set the display start position of the upper left corner of the display.

Type

Function replacement.

Prototype

```
#define LCD_SET_ORG(x, y)
```

Parameter	Meaning
<code>x</code>	X position of the visible area.
<code>y</code>	Y position of the visible area.

Example

```
#define LCD_SET_ORG(x, y) SetDisplayOrigin(x, y) /* Function call for setting the display start position */
```

`LCD_VXSIZE, LCD_VYSIZE`

Description

The virtual screen size is configured by the macros `LCD_VXSIZE` and `LCD_VYSIZE`. `LCD_VXSIZE` always should be > `LCD_XSIZE` and `LCD_VYSIZE` should be > `LCD_YSIZE`. If a virtual area is configured the clipping area of emWin depends on the virtual screen and not on the display size. Drawing operations outside of `LCD_XSIZE` and `LCD_YSIZE` but inside the virtual screen are performed.

Type

Numerical values.

17.3.1 Sample configuration

The following excerpt of the file `LCDConf.h` shows how to configure emWin for using a virtual area of 640x480 pixels on a QVGA display with 320x240 pixels:

```
#define LCD_SET_ORG(x, y) SetDisplayOrigin(x, y) /* Function call for setting the display start position */
#define LCD_XSIZE          320 /* X-resolution of LCD */
#define LCD_YSIZE          240 /* Y-resolution of LCD */
#define LCD_VXSIZE         640 /* Virtual X-resolution */
#define LCD_VYSIZE         480 /* Virtual Y-resolution */
```

17.4 Samples

In the following a few samples are shown to make clear how to use virtual screens with emWin.

17.4.1 Basic sample

The following sample shows how to use a virtual screen of 128x192 and a display of 128x64 for instantaneous switching between 3 different screens.

Configuration

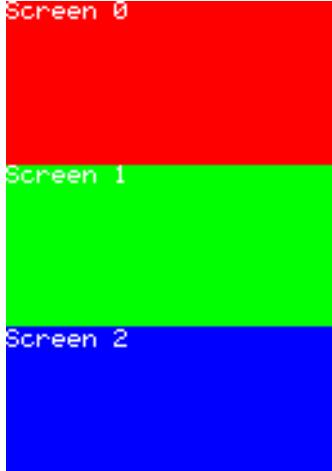
```
#define LCD_XSIZE 128
#define LCD_YSIZE 64
#define LCD_VYSIZE 192
```

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, 127, 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 */
```

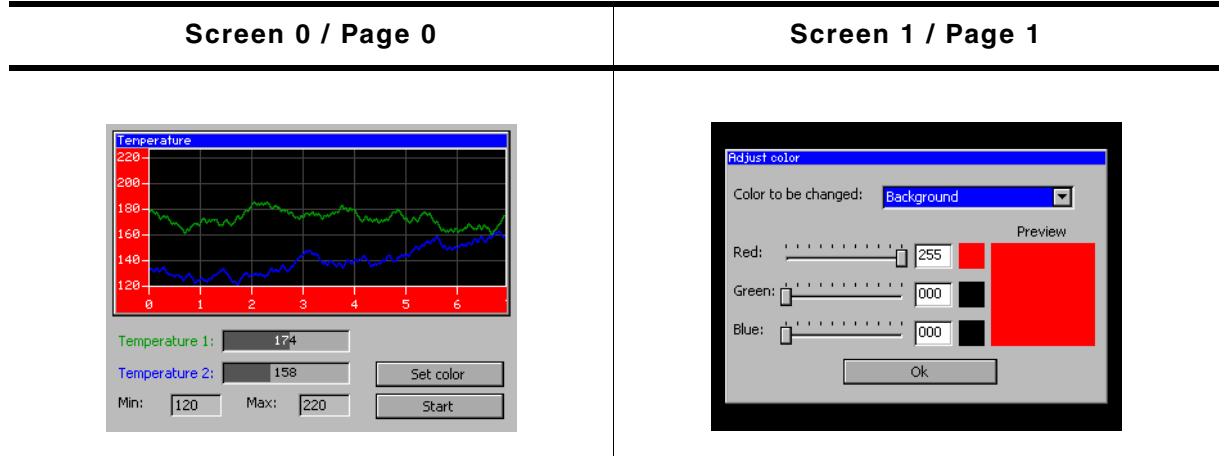
Output

The table below shows the output of the display:

Explanation	Display output	Contents of virtual area
Before executing GUI_SetOrg(0, 240)		
After executing GUI_SetOrg(0, 240)		
After executing GUI_SetOrg(0, 480)		

17.4.2 Real time sample using the window manager

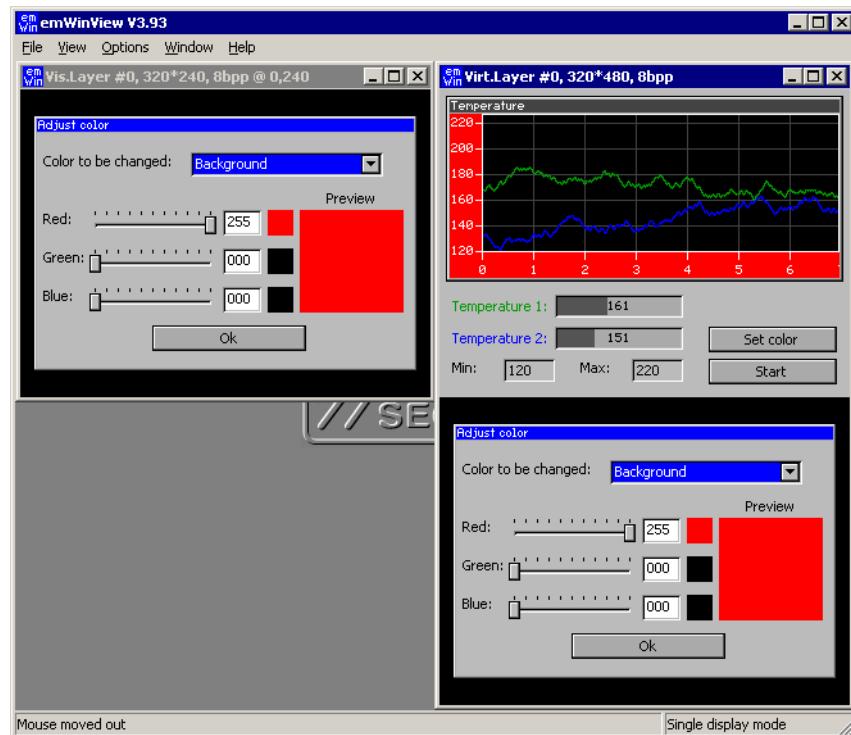
The shipment of emWin contains a sample which shows how to use virtual screens in a real time application. It can be found under Sample\GUI\VSCREEN_RealTime.c:



After showing a short introduction the sample 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, please take a look at the source code of the sample.

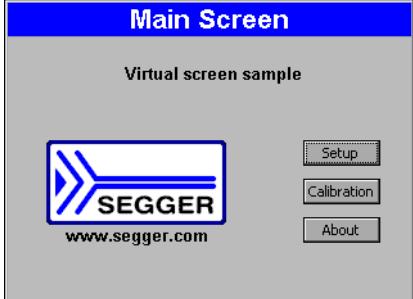
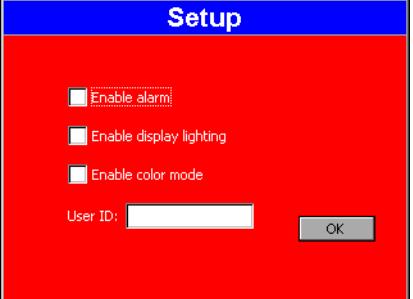
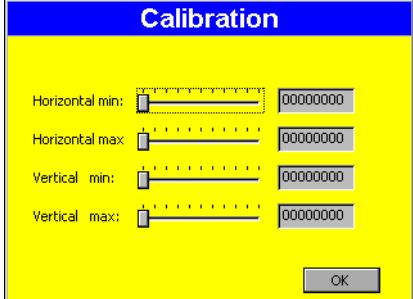
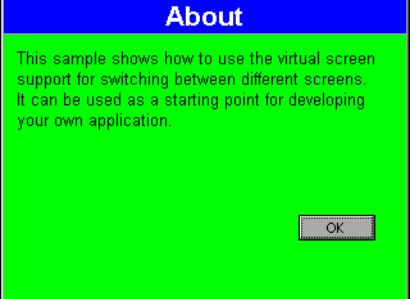
Viewer Screenshot of the above sample



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.

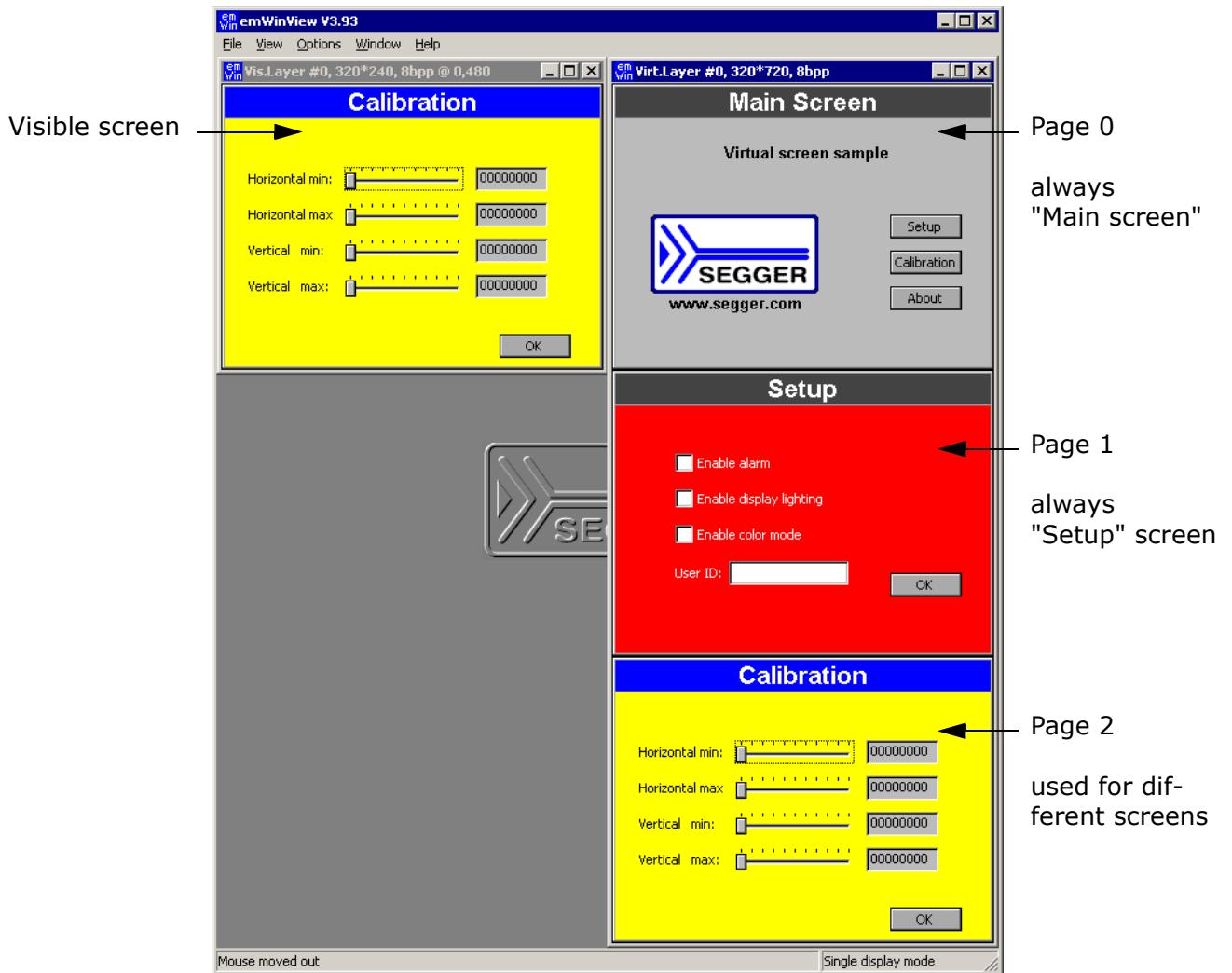
17.4.3 Dialog sample using the window manager

The second advanced sample 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:

Main screen / Page 0	Setup screen / Page 1
	
Calibration screen / Page 2	About screen / Page 2
	

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 sample



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.

17.5 Virtual screen API

The following table lists the available routines of the virtual screen support.

Routine	Explanation
GUI_GetOrg()	Returns the display start position.
GUI_SetOrg()	Sets the display start position.

GUI_GetOrg()

Description

Returns the display start position.

Prototype

```
void GUI_GetOrg(int * px, int * py);
```

Parameter	Meaning
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.

Add. 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	Meaning
x	New X position of the display start position.
y	New Y position of the display start position.

Chapter 18

Multi layer / multi display support

If you need to access more than 1 display or your LCD-controller supports more than 1 layer (and you want to use more than one layer) you need the multi layer support of emWin. This feature is a separate (optional) software item and is not included in the emWin basic package. The software for the multi layer support is located in the subdirectory GUI\MultiLayer.

Multi layer support and multi display support work the same way. In the configuration file you can define up to 5 displays/layers with their own settings. That means each display can be accessed with its own color settings, its own size and its own LCD-driver. In order to activate this feature, you simply need to use an appropriate LCD-configuration file LCDConf.h and you have to define the number of layers in GUIConf.h.

18.1 Introduction

Windows can be placed in any layer or display, drawing operations can be used on any layer or display. Since there are really only smaller differences from this point of view, multiple layers and multiple displays are handled the same way (Using the same API routines) and are simply referred to as multiple layers, even if the particular embedded system uses multiple displays. The emWin viewer allows you to look at every individual layer (display), but in the case of multiple layer systems also to look at the actual output (the composite view). Currently systems with multiple displays and multiple layers can be used, but not simulated.

18.1.1 Limitations

Currently systems with a total of max 5 displays / layers are supported.

18.1.2 Selecting a layer for drawing operations

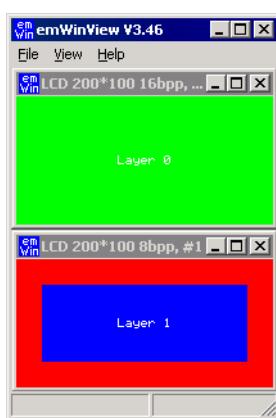
When drawing directly, per default layer 0 is used. An other layer can be selected by using `GUI_SelectLayer()`.

Sample

The following sample 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); /* Select layer 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 sample



18.1.3 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 (a child or grandchild or ...) of one of these desktop windows. Which layer a window is in depends solely on which desktop window it is a descendent of.

Sample

The following sample shows how to create 3 windows on 2 different desktop windows:

```
/* Create 1 child window on desktop 0 */
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                               WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0 );
/* Create 2 child windows on desktop 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 sample:

Screenshot	Window hierarchy
	<pre> graph TD Desktop0[Desktop 0] --- Window0[Window 0] Desktop1[Desktop 1] --- Window1[Window 1] Desktop1 --- Window2[Window 2] </pre>

18.1.3.1 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 an other layer, this window actually moves from one layer to an other layer.

Sample

The following sample shows how to attach a window to a new parent window:

```
/* Create 1 child window on desktop 0 */
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                               WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0 );
/* Create 2 child windows on desktop 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 sample before attaching the window to the new parent:

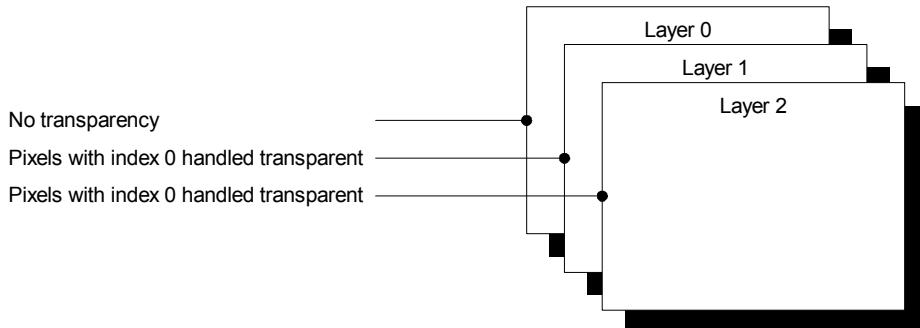
Screenshot	Window hierarchy
	<p>Layer 0</p> <pre> graph TD Desktop0[Desktop 0] --- Window0[Window 0] Desktop1[Desktop 1] --- Window1[Window 1] Desktop1 --- Window2[Window 2] </pre> <p>Layer 1</p>

The next table shows the screenshot and the window hierarchy of the above sample after attaching the window to the new parent:

Screenshot	Window hierarchy
	<p>Layer 0</p> <pre> graph TD Desktop0[Desktop 0] --- Window0[Window 0] Desktop0 --- Window2[Window 2] Desktop1[Desktop 1] --- Window1[Window 1] </pre> <p>Layer 1</p>

18.2 Using multi layer 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 the layers one above the other:



18.2.1 Using the viewer with multiple layers

The emWin viewer treats pixels with index 0 in a layer above layer 0 as transparent. This means in the composite view the pixels of the layers below are visible.

18.2.2 Using transparency

The following sample shows how to draw transparent and non transparent:

```
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_WHITE);           /* Select white for the background */
GUI_Clear();
GUI_SetColor(GUI_BLACK);            /* Select black for the text */
GUI_SetFont(&GUI_FontComic24B_ASCII);
GUI_DispStringHCenterAt("Layer#:1",
                        XSize / 2, 5);
GUI_SetColorIndex(0);               /* Select transparency */
GUI_FillCircle(LCD_GetXSize() / 2,
               Ysize / 2 + 15,
               Ysize / 2 - 20);          /* A transparent circle will be drawn */
```

18.2.3 Multi layer sample

Chapter 3 "Simulator and viewer" contains a multi layer sample. For details please take a look to this chapter.

18.2.4 Pitfalls

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 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

8666_1 is currently the only available mode. For details please take a look at chapter colors.

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 a sample definition for a custom palette with 15 gray scales:

```
#define LCD_FIXEDPALETTE_1 0
#define LCD_PHYSCOLORS_1 0x000000,           /* Transparency, not black */ \
                           0x000000, 0x222222, 0x333333, \
                           0x444444, 0x555555, 0x666666, 0x777777, \
                           0x888888, 0x999999, 0xAAAAAA, 0xBBBBBB, \
```

```
0xCCCCCC, 0xDDDDDD, 0xEEEEEE, 0xFFFFFFF
```

If you need to draw some black items with this palette you can not use `GUI_SetColor(0x000000)`. The result from the color conversion routines would be index 0 (transparent). You should use `GUI_SetColorIndex(1)`.

18.3 Using multi display support

Each display can be accessed with its own driver and with its own settings. You can define up to 5 different displays.

18.3.1 Enabling multi display support

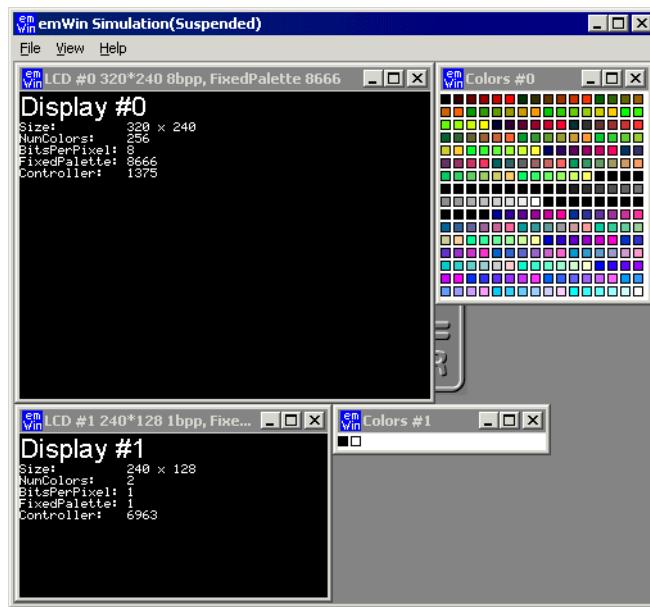
To enable the multi display support you have to define the number of layers in `GUIConf.h`:

```
#define GUI_NUM_LAYERS 2 /* Enables support for 2 displays/layers */
```

Further you have to modify your `LCDConf.h` as described later in this chapter.

18.3.2 Multi display sample

The sample below shows a screenshot of the simulation with 2 displays. The first 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 `LCDSSlin.c` configured for a Toshiba T6963 LCD-controller:



18.4 Configuring multi layer support

LCD Configuration of the above multi layer sample

```
#ifndef LCDCONF_H
#define LCDCONF_H

*****
*          Configuration of Layer 0
*
*****
```

```
#define LCD_CONTROLLER_0      3916
#define LCD_XSIZE_0           400           /* X-resolution of LCD, Logical coor. */
```

```

#define LCD_YSIZE_0           234      /* Y-resolution of LCD, Logical coor. */
#define LCD_BITSPERPIXEL_0    16
#define LCD_FIXEDPALETTE_0    655
#define LCD_SWAP_BYTE_ORDER_0 1
#define LCD_SWAP_RB_0          1
#define LCD_INIT_CONTROLLER_0()
#define LCD_READ_MEM_0(Off)    *((U16 *) (0xc00000 + (((U32)(Off)) << 1)))
#define LCD_WRITE_MEM_0(Off,data) *((U16 *) (0xc00000 + (((U32)(Off)) << 1))) = data

/*********************  

*  

*      Configuration of Layer 1  

*  

*****  

*/
#define LCD_CONTROLLER_1        3916     /* X-resolution of LCD, Logical coor. */
#define LCD_XSIZE_1             400       /* Y-resolution of LCD, Logical coor. */
#define LCD_YSIZE_1             234       /* Y-resolution of LCD, Logical coor. */
#define LCD_BITSPERPIXEL_1       8
#define LCD_FIXEDPALETTE_1      866661
#define LCD_SWAP_BYTE_ORDER_1   1
#define LCD_SWAP_RB_1            1
#define LCD_INIT_CONTROLLER_1()
#define LCD_READ_MEM_1(Off)     *((U16 *) (0xd00000 + (((U32)(Off)) << 1)))
#define LCD_WRITE_MEM_1(Off,data) *((U16 *) (0xd00000 + (((U32)(Off)) << 1))) = data

#endif /* LCDCONF_H */

```

18.5 Configuring multi display support

Configuration of the above multi display sample

```

#ifndef LCDCONF_H
#define LCDCONF_H

/*********************  

*  

*      Configuration of Display 0  

*  

*****  

*/
#define LCD_CONTROLLER_0         1375     /* X-resolution of LCD, Logical coor. */
#define LCD_XSIZE_0              320       /* Y-resolution of LCD, Logical coor. */
#define LCD_YSIZE_0              240       /* X-resolution of LCD, Logical coor. */
#define LCD_VXSIZE_0             320       /* Y-resolution of LCD, Logical coor. */
#define LCD_VYSIZE_0              240       /* If active: X <-> Y */
#define LCD_SWAP_XY_0              0
#define LCD_BITSPERPIXEL_0        8

/*********************  

*  

*      Full bus configuration  

*  

*/
#define LCD_READ_MEM_0(Off)      *((U16 *) (0xc00000 + (((U32)(Off)) << 1)))
#define LCD_WRITE_MEM_0(Off,data) *((U16 *) (0xc00000 + (((U32)(Off)) << 1))) = data
#define LCD_READ_REG_0(Off)       *((volatile U16 *) (0xc1ffe0 + (((U16)(Off)) << 1)))
#define LCD_WRITE_REG_0(Off,data) *((volatile U16 *) (0xc1ffe0 + (((U16)(Off)) << 1))) = data

/*********************  

*  

*      Define contents of registers  

*  

*/
#define LCD_REG0_0      (0)
#define LCD_REG1_0      (0x23) \
| (1<<2)
#define LCD_REG2_0      ((3<<6) \
| (1<<5) \
| (1<<4) \
| (0<<3) \
| (0<<2) \
| (0<<1) \
| (0<<0))
#define LCD_REG3_0      \
| ((0<<7) \
| (0<<3) \

```

```

        | (0<<2) \
        | (3<<0))
#define LCD_REG4_0 ((LCD_XSIZE_0 / 8 - 1)
#define LCD_REG5_0 ((LCD_YSIZE_0 - 1) & 255)
#define LCD_REG6_0 ((LCD_YSIZE_0 - 1) >> 8)
#define LCD_REG7_0 (0)
#define LCD_REG8_0 (31)
#define LCD_REG9_0 (0)
#define LCD_REGA_0 (0)
#define LCD_REGB_0 (0)
#define LCD_REGC_0 (0)
#define LCD_REGD_0 (0)
#define LCD_REG12_0 (LCD_BITSPERPIXEL_0 * (LCD_VXSIZE_0 - LCD_XSIZE_0) / 16)
#define LCD_REG13_0 LCD_REG5_0
#define LCD_REG14_0 LCD_REG6_0
#define LCD_REG1B_0 (0)
#define LCD_REG1C_0 (0x78)

//*****************************************************************************
*
*      Init sequence for 16 bit access
*/
#ifndef !LCD_SWAP_BYTE_ORDER_0
#define LCD_WRITE_REGLH_0(Addr, d0, d1) LCD_WRITE_REG(Addr, ((d0)<<8) | (d1))
#else
#define LCD_WRITE_REGLH_0(Addr, d0, d1) LCD_WRITE_REG(Addr, ((d1)<<8) | (d0))
#endif

#define LCD_INIT_CONTROLLER_0() \
    LCD_WRITE_REGLH_0(0x00 >> 1, LCD_REG0_0, LCD_REG1_0); \
    LCD_WRITE_REGLH_0(0x02 >> 1, LCD_REG2_0, LCD_REG3_0); \
    LCD_WRITE_REGLH_0(0x04 >> 1, LCD_REG4_0, LCD_REG5_0); \
    LCD_WRITE_REGLH_0(0x06 >> 1, LCD_REG6_0, LCD_REG7_0); \
    LCD_WRITE_REGLH_0(0x08 >> 1, LCD_REG8_0, LCD_REG9_0); \
    LCD_WRITE_REGLH_0(0x0a >> 1, LCD_REGA_0, LCD_REGB_0); \
    LCD_WRITE_REGLH_0(0x0c >> 1, LCD_REGC_0, LCD_REGD_0); \
    LCD_WRITE_REG((0xe0 >> 1, 0x00)); \
    LCD_WRITE_REG((0x10 >> 1, 0x00)); \
    LCD_WRITE_REGLH_0(0x12 >> 1, LCD_REG12_0, LCD_REG13_0); \
    LCD_WRITE_REGLH_0(0x14 >> 1, LCD_REG14_0, 0); \
    LCD_WRITE_REGLH_0(0x1a >> 1, 0, LCD_REG1B_0); \
    LCD_WRITE_REGLH_0(0x1c >> 1, LCD_REG1C_0, 0)

//*****************************************************************************
*
*      Configuration of Display 1
*
*****
```

```
#define LCD_CONTROLLER_1      6963
#define LCD_XSIZE_1           240      /* X-resolution of LCD, Logical coor. */
#define LCD_YSIZE_1           128      /* Y-resolution of LCD, Logical coor. */
#define LCD_BITSPERPIXEL_1    1
```

```
*****
```

```
*
```

```
*      Simple bus configuration
*/
```

```
void LCD_X_Write00(char c);
void LCD_X_Write01(char c);
char LCD_X_Read00(void);
char LCD_X_Read01(void);
#define LCD_WRITE_A1_1(Byte) LCD_X_Write01(Byte)
#define LCD_WRITE_A0_1(Byte) LCD_X_Write00(Byte)
#define LCD_READ_A1_1()      LCD_X_Read01()
#define LCD_READ_A0_1()      LCD_X_Read00()
```

```
#define LCD_INIT_CONTROLLER_1() LCD_X_Init()
```

```
#endif /* LCDCONF_H */
```

18.6 Multi layer API

The table below lists the available multi layer related routines in alphabetical order. Detailed descriptions follow:

Routine	Explanation
GUI_SelectLayer()	Selects a layer/display for output operations
GUI_SetLUTColorEx()	Set color of a color index in the given layer.
LCD_GetNumDisplays()	Returns the number of displays.

GUI_SelectLayer()

Description

Selects a layer for drawing operations.

Prototype

```
unsigned int GUI_SelectLayer(unsigned int Index);
```

Parameter	Meaning
Index	Layer index.

Return value

Index of previous selected layer.

GUI_SetLUTColorEx()

Description

Modifies a single entry in the color table and the LUT of the given layer.

Prototype

```
void GUI_SetLUTColorEx(U8 Pos, LCD_COLOR Color, unsigned int LayerIndex);
```

Parameter	Meaning
Pos	Position within the lookup table. Should be less than the number of colors (e.g. 0-3 for 2 bpp, 0-15 for 4 bpp, 0-255 for 8 bpp).
Color	24-bit RGB value.
Index	Layer index.

Add. Information

(see [GUI_SetLUTColor](#))

LCD_GetNumDisplays()

Description

Returns the number of displays/layers configured in your configuration.

Prototype

```
int LCD_GetNumDisplays(void);
```

Return value

Number of displays/layers configured in your configuration.

Chapter 19

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 a sample 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`.

19.1 Description

Pointer input devices are devices such as mouse, touch-screen, joystick. 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 call the routine `GUI_PID_StoreState()` whenever an event (such as mouse move, or press on the touch screen) has been detected.

The window manager takes care of the appropriate reaction to PID events; If the window manager is not used, your application is responsible for reacting to PID events.

19.2 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	Explanation
<code>GUI_PID_GetState()</code>	Return the current state of the PID.
<code>GUI_PID_StoreState()</code>	Store the current state of the PID.

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;
    unsigned char Pressed;
} GUI_PID_STATE;
```

Elements of `GUI_PID_STATE`

Data type	Element	Meaning
int	x	X position of pointer input device.
int	y	Y position of pointer input device.
unsigned char	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.

`GUI_PID_GetState()`

Description

Returns if the input device is currently pressed or not and fills the given `GUI_PID_STATE` structure with the current state information.

Prototype

```
int GUI_PID_GetState(GUI_PID_STATE *pState);
```

Parameter	Meaning
<code>pState</code>	Pointer to a structure of type <code>GUI_PID_STATE</code> to be filled with the current state.

Return value

1 if input device is currently pressed; 0 if not pressed.

Sample

```
GUI_PID_STATE State;
GUI_PID_GetState(&State);
```

`GUI_PID_StoreState()`

Description

Stores the current state of the pointer input device.

Prototype

```
void GUI_PID_StoreState(const GUI_PID_STATE *pState);
```

Parameter	Meaning
<code>pState</code>	Pointer to a structure of type <code>GUI_PID_STATE</code> .

Add. information

This function can be used from an interrupt service routine.

19.3 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.

19.3.1 Generic mouse 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	Explanation
GUI_MOUSE_GetState()	Return the current state of the mouse.
GUI_MOUSE_StoreState()	Store the current state of the mouse.

GUI_MOUSE_GetState()

Description

Returns the current state of the mouse.

Prototype

```
int GUI_MOUSE_GetState(GUI_PID_STATE *pState);
```

Parameter	Meaning
pState	Pointer to a structure of type GUI_PID_STATE .

Return value

1 if mouse is currently pressed; 0 if not pressed.

Add. 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	Meaning
pState	Pointer to a structure of type GUI_PID_STATE .

Add. information

This function will call [GUI_PID_StoreState\(\)](#).

This function can be used from an interrupt service routine.

Sample

```
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);
```

19.3.2 PS2 mouse driver

The driver supports any type of PS2 mouse.

19.3.2.1 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.

A sample 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
}
```

19.3.2.2 PS2 mouse driver API

The table below lists the available mouse driver routines in alphabetical order.

Routine	Explanation
<code>GUI_MOUSE_DRIVER_PS2_Init()</code>	Initialise the mouse driver.
<code>GUI_MOUSE_DRIVER_PS2_OnRx()</code>	Called from receive interrupt routines.

`GUI_MOUSE_DRIVER_PS2_Init()`

Description

Initialises 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	Meaning
<code>Data</code>	Byte of data received by ISR.

Add. 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.

19.4 Touch-screen drivers

A touch screen driver will typically simply call `GUI_PID_StoreState()` as described earlier. Any type of touch screen can be supported this way. You are responsible for writing 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.

19.4.1 Generic touch-screen 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	Explanation
<code>GUI_TOUCH_GetState()</code>	Return the current state of the touch-screen.
<code>GUI_TOUCH_StoreState()</code>	Store the current state of the touch-screen using X- and Y-coordinates.
<code>GUI_TOUCH_StoreStateEx()</code>	Store the current state of the touch-screen.

GUI_TOUCH_GetState()

Description

Returns the current state of the touch-screen.

Prototype

```
int GUI_TOUCH_GetState(GUI_PID_STATE *pState);
```

Parameter	Meaning
<code>pState</code>	Pointer to a structure of type <code>GUI_PID_STATE</code> .

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	Meaning
<code>x</code>	X-position.
<code>y</code>	Y-position.

Add. Information

If one of the given values is negative, the GUI assumes that the touch panel is not pressed.

This function can be used from an interrupt service routine.

Sample

```
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	Meaning
pState	Pointer to a structure of type GUI_PID_STATE.

Add. information

This function will call `GUI_PID_StoreState()`.

Sample

```

GUI_PID_STATE State;
State.x = _TouchPositionX;
State.y = _TouchPositionY;
if (_TouchIsPressed) {
    State.Pressed = 1;
} else {
    State.Pressed = 0;
}
GUI_TOUCH_StoreStateEx(&State);

```

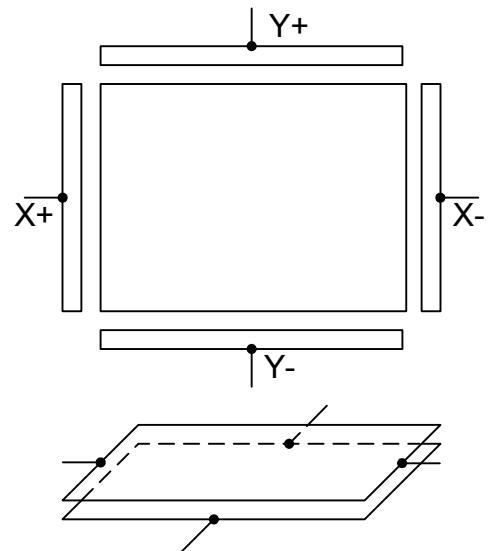
19.4.2 The analog touch screen driver

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 a 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.



19.4.2.1 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
- Use sample to determine values for `GUITouchConf.h`
- Modifying `GUITouchConf.h`

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 modul `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 off the voltage in Y and disable the measurement in X.

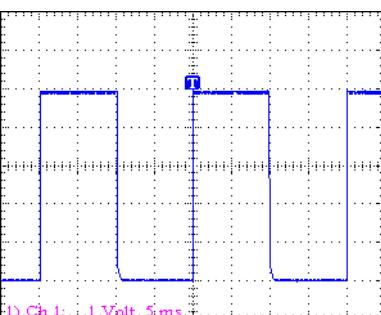
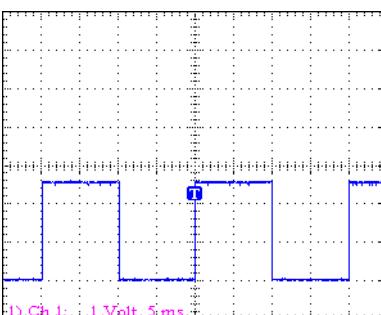
The measurement routines should return the measurement result of a A/D converter. Later in this chapter you will find a sample implementation of the hardware routines.

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. Your application should call it about 100 times/second. 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 multitasking system, you can use an interrupt service routine to do the job.

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
	

Use sample to determine values for `GUITouchConf.h`

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:

```
GUI_TOUCH_AD_TOP
GUI_TOUCH_AD_BOTTOM
GUI_TOUCH_AD_LEFT
GUI_TOUCH_AD_RIGHT
```

The sample 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\GUI and its name is TOUCH_Sample.c. Run this sample on your hardware. The output should be similar to the screenshot at the right side. The following table shows how to get the values:

Measurement of
A/D converter values
Analog input:
x:0423, y:0386
Position:
x:0093, y:0043



Value	How to get
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.

Modifying GUITouchConf.h

The last step is to modify the touch configuration file GUITouchConf.h. Insert the values into the file like the following sample:

```
*****  
*          SEGGER MICROCONTROLLER SYSTEME GmbH          *  
*          Solutions for real time microcontroller applications      *  
*****  
-----  
File       : GUITouch.Conf.h  
Purpose    : Configures emWin touch screen module  
-----  
*/  
  
#ifndef GUITOUCH_CONF_H  
#define GUITOUCH_CONF_H  
  
#define GUI_TOUCH_AD_TOP      877  
#define GUI_TOUCH_AD_BOTTOM   273  
#define GUI_TOUCH_AD_LEFT     232  
#define GUI_TOUCH_AD_RIGHT    918  
  
#endif /* GUITOUCH_CONF_H */
```

19.4.2.2 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 needfull to calibrate each device at runtime. This can be done by using the function GUI_TOUCH_Calibrate(). The sample folder contains the sample TOUCH_Calibrate.c which shows, how a touch screen can be calibrated at run time:



**Runtime calibration,
please touch the screen
at the center of the ring.**

19.4.2.3 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	Explanation
<code>GUI_TOUCH_X_ActivateX()</code>	Prepares measurement for Y-axis.
<code>GUI_TOUCH_X_ActivateY()</code>	Prepares measurement for X-axis.
<code>GUI_TOUCH_X_MeasureX()</code>	Returns the X-result of the A/D converter.
<code>GUI_TOUCH_X_MeasureY()</code>	Returns the Y-result of the A/D converter.

`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);
```

Sample implementation

The following shows a sample implementation of the touch hardware routines for a Mitsubishi 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 sample 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;
}

```

19.4.2.4 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	Explanation
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_SetDefaultCalibration()	Restores the default calibration.

GUI_TOUCH_Calibrate()

Description

Changes the calibration at run-time.

Prototype

```
int GUI_TOUCH_Calibrate(int Coord, int Log0, int Log1, int Phys0, int
                        Phys1);
```

Parameter	Meaning
Coord	0 for X-axis, 1 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.

Add. 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.

GUI_TOUCH_Exec()**Description**

Polls the touch-screen by calling the TOUCH_X routines to activate the measurement of the X- and Y-axes. You must ensure that this function is called for about 100 times per second.

Prototype

```
void GUI_TOUCH_Exec(void);
```

Add. 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.

GUI_TOUCH_SetDefaultCalibration()**Description**

Resets the calibration to the values set as default in the configuration file.

Prototype

```
void GUI_TOUCH_SetDefaultCalibration(void);
```

Add. information

If no values are set in the configuration file, the calibration will be restored to the original default values.

19.4.2.5 Configuring the analog touch-screen driver

There needs to exist a separate configuration file in your config folder named GUITouchConf.h. The following table shows all available config macros for the analog touch-screen driver included with emWin:

Type	Macro	Default	Explanation
B	GUI_TOUCH_SWAP_XY	0	Set to 1 to swap the X- and the Y-axes.
B	GUI_TOUCH_MIRROR_X	0	Mirrors the X-axis.
B	GUI_TOUCH_MIRROR_Y	0	Mirrors the Y-axis.
N	GUI_TOUCH_AD_LEFT	30	Minimum value returned by the A/D converter.
N	GUI_TOUCH_AD_RIGHT	220	Maximum value returned by the A/D converter.
N	GUI_TOUCH_AD_TOP	30	Minimum value returned by the A/D converter.

Type	Macro	Default	Explanation
N	GUI_TOUCH_AD_BOTTOM	220	Maximum value returned by the A/D converter.
N	GUI_TOUCH_XSIZE	LCD_XSIZE	Horizontal area covered by touch-screen.
N	GUI_TOUCH_YSIZE	LCD_YSIZE	Vertical area covered by touch-screen.

19.5 Joystick input sample

The following sample shows how to use the pointer input device API 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) {
    int Stat;
    int StatPrev = 0;
    int TimeAcc = 0;    // Dynamic acceleration value
    while (1) {
        Stat = HW_ReadJoystick();
        //
        // Handle dynamic pointer acceleration
        //
        if (Stat == StatPrev) {
            if (TimeAcc < 10) {
                TimeAcc++;
            }
        } else {
            TimeAcc = 1;
        }
        if (Stat || (Stat != StatPrev)) {
            GUI_PID_STATE State;
            int Max;
            //
            // 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;
            }
            Max = LCD_GetXSize() - 1;
            if (State.x >= Max) {
                State.x = Max;
            }
            Max = LCD_GetYSize() - 1;
            if (State.y > Max) {
                State.y = Max;
            }
            //
            // Inform emWin
            //
            State.Pressed = (Stat & JOYSTICK_ENTER) ? 1: 0;
            GUI_PID_StoreState(&State);
            StatPrev = Stat;
        }
        OS_Delay(40);
    }
}
```

Chapter 20

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.

20.1 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
<code>GUI_KEY_BACKSPACE</code>	Backspace key.
<code>GUI_KEY_TAB</code>	Tab key.
<code>GUI_KEY_ENTER</code>	Enter/return key.
<code>GUI_KEY_LEFT</code>	Left arrow key.
<code>GUI_KEY_UP</code>	Up arrow key.
<code>GUI_KEY_RIGHT</code>	Right arrow key.
<code>GUI_KEY_DOWN</code>	Down arrow key.
<code>GUI_KEY_HOME</code>	Home key (move to beginning of current line).
<code>GUI_KEY_END</code>	End key (move to end of current line).
<code>GUI_KEY_SHIFT</code>	Shift key.
<code>GUI_KEY_CONTROL</code>	Control key.
<code>GUI_KEY_ESCAPE</code>	Escape key.
<code>GUI_KEY_INSERT</code>	Insert key.
<code>GUI_KEY_DELETE</code>	Delete key.

20.1.1 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	Explanation
<code>GUI_StoreKeyMsg()</code>	Store a message in a specified key.
<code>GUI_SendKeyMsg()</code>	Send a message to a specified key.

`GUI_StoreKeyMsg()`

Description

Stores the message data (Key, PressedCnt) into the keyboard buffer.

Prototype

```
void GUI_StoreKeyMsg(int Key, int Pressed);
```

Parameter	Meaning
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined emWin character code.
Pressed	Key state (see table below).

Permitted values for parameter Pressed	
1	Pressed state.
0	Released (unpressed) state.

Add. information

This function can be used from an interrupt service routine. The input buffer contains only one character.

GUI_SendKeyMsg()

Description

Sends the keyboard data 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	Meaning
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined emWin character code.
Pressed	Key state (see <code>GUI_StoreKeyMsg()</code>).

Add. infoemation

This function should not be called from an interrupt service routine.

20.1.2 Application layer API

The table below lists the application-layer keyboard routines in alphabetical order. Detailed descriptions follow.

Routine	Explanation
<code>GUI_ClearKeyBuffer()</code>	Clear the key buffer.
<code>GUI_GetKey()</code>	Return the contents of the key buffer.
<code>GUI_StoreKey()</code>	Store a key in the buffer.
<code>GUI_WaitKey()</code>	Wait for a key to be pressed.

GUI_ClearKeyBuffer()

Description

Clears the key buffer.

Prototype

```
void GUI_ClearKeyBuffer(void);
```

GUI_GetKey()

Description

Returns the current contents of the key buffer.

Prototype

```
int GUI_GetKey(void);
```

Return value

Codes of characters in key buffer; 0 if no keys in buffer.

GUI_StoreKey()

Description

Stores a key in the buffer.

Prototype

```
void GUI_StoreKey(int Key);
```

Parameter	Meaning
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined emWin character code.

Add. 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);
```

Add. Information

The application is "blocked", meaning it will not return until a key is pressed.

Chapter 21

Cursors

Window manager support includes a system-wide cursor which may be changed to other, predefined styles. Although the cursor always exists, it is hidden by default like a window. It will not be visible until a call is made to show it, and may be hidden again at any point.

21.1 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	
Inverted arrow cursors		Inverted 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	
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	

21.2 Cursor API

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

Routine	Explanation
<code>GUI_CURSOR_Hide()</code>	Hide the cursor.
<code>GUI_CURSOR_Select()</code>	Set a specified cursor.
<code>GUI_CURSOR_SetPosition()</code>	Set the cursor position.
<code>GUI_CURSOR_Show()</code>	Show the cursor.

`GUI_CURSOR_Hide()`

Description

Hides the cursor.

Prototype

```
void GUI_CURSOR_Hide(void);
```

Add. information

This is the default cursor setting. If you want the cursor to be visible, you must call `GUI_CURSOR_Show()`.

GUI_CURSOR_Select()

Description

Sets a specified cursor style.

Prototype

```
void GUI_CURSOR_Select(const GUI_CURSOR *pCursor);
```

Parameter	Meaning
<code>pCursor</code>	Pointer to the cursor to be selected.

Permitted values for parameter <code>pCursor</code>	
<code>GUI_CursorArrows</code>	Small arrow.
<code>GUI_CursorArrowM</code>	Medium arrow.
<code>GUI_CursorArrowL</code>	Large arrow.
<code>GUI_CursorArrowSI</code>	Small inverted arrow.
<code>GUI_CursorArrowMI</code>	Medium inverted arrow.
<code>GUI_CursorArrowLI</code>	Large inverted arrow.
<code>GUI_CursorCrossS</code>	Small cross.
<code>GUI_CursorCrossM</code>	Medium cross.
<code>GUI_CursorCrossL</code>	Large cross.
<code>GUI_CursorCrossSI</code>	Small inverted cross.
<code>GUI_CursorCrossMI</code>	Medium inverted cross.
<code>GUI_CursorCrossLI</code>	Large inverted cross.

Add. information

If this function is not called, the default cursor is a medium arrow.

GUI_CURSOR_SetPosition()

Description

Sets the cursor position.

Prototype

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

Parameter	Meaning
<code>x</code>	X-position of the cursor.
<code>y</code>	Y-position of the cursor.

Add. 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);
```

Add. information

The default setting for the cursor is hidden; therefore this function must be called if you want the cursor to be visible.

Chapter 22

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\Anti-Alias`.

22.1 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).

22.1.1 Quality of antialiasing

The quality of antialiasing is set by the routine `GUI_AA_SetFactor()`, explained later in the 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 $2 \times 2 = 4$. The next line is drawn with an antialiasing factor of 3, so there are $3 \times 3 = 9$ shades, and so on. Factor 4 should be sufficient for most applications. Increasing the antialiasing factor further does not improve the result dramatically, but increases the calculation time.

1 2 3 4 5 6



Increasing the anti-aliasing factor further does not improve the result dramatically, but increases the calculation time.

22.1.2 Antialiased Fonts

Two types of antialiased fonts, low-quality (2bpp) and high-quality (4bpp), are supported. The routines needed 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		

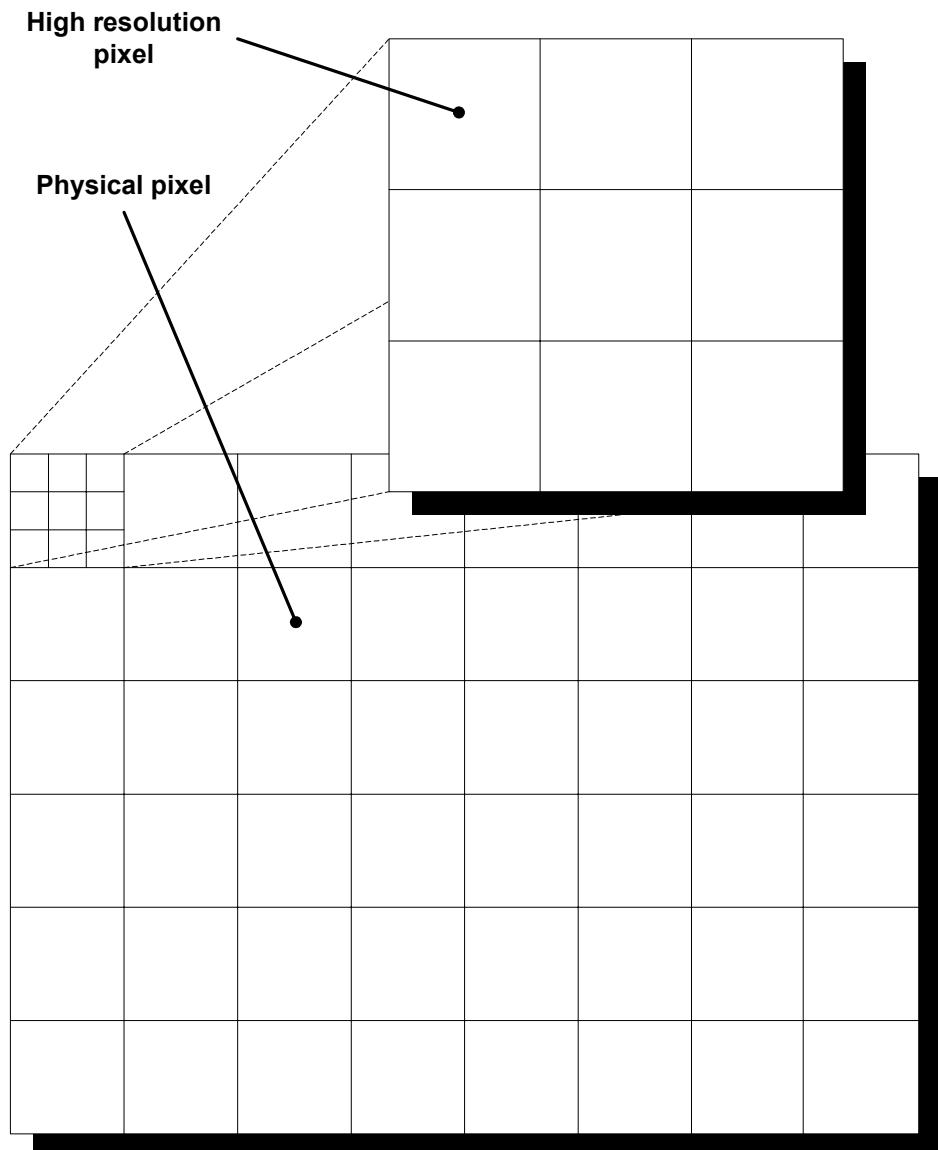
Antialiased fonts can be created with the emWin font converter. The general purpose of using antialiased fonts is to improve the appearance of text. While the effect of using high-quality antialiasing will be more visually pleasing than low-quality, computation time and memory consumption will increase proportionally. Low-quality (2bpp) fonts require twice the memory of non-antialiased (1bpp) fonts; high-quality (4bpp) fonts require four times the memory.

22.1.3 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. You do not need to consider the antialiasing factor in the function arguments. For example, to draw an antialiased line from (50, 100) to (100, 50) you would write:

```
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 anti-aliasing 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 sample programs using the high-resolution feature, see the examples at the end of the chapter.

22.2 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	Explanation
Control functions	
<code>GUI_AA_DisableHiRes()</code>	Disable high-resolution coordinates.
<code>GUI_AA_EnableHiRes()</code>	Enable high-resolution coordinates.
<code>GUI_AA_GetFactor()</code>	Return the current antialiasing factor.
<code>GUI_AA_SetFactor()</code>	Set the current antialiasing factor.
Drawing functions	
<code>GUI_AA_DrawArc()</code>	Draw an antialiased arc.
<code>GUI_AA_DrawLine()</code>	Draw an antialiased line.
<code>GUI_AA_DrawPolyOutline()</code>	Draw the outline of an antialiased polygon of max. 10 points.
<code>GUI_AA_DrawPolyOutlineEx()</code>	Draw the outline of an antialiased polygon.
<code>GUI_AA_FillCircle()</code>	Draw an antialiased circle.
<code>GUI_AA_FillPolygon()</code>	Draw a filled and antialiased polygon.

22.3 Control functions

`GUI_AA_DisableHiRes()`

Description

Disables high-resolution coordinates.

Prototype

```
void GUI_AA_DisableHiRes(void);
```

Add. 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	Meaning
Factor	The new antialiasing factor. Minimum: 1 (will result in no antialiasing); maximum: 6.

Add. information

Setting the parameter `Factor` to 1, though permitted, will effectively disable anti-aliasing and result in a standard font.

We recommend an antialiasing quality factor of 2-4. The default factor is 3.

22.4 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	Meaning
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).

Limitations

Currently the `ry` parameter is not available. The `rx` parameter is used instead.

Add. 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	Meaning
x0	X-starting position.
y0	Y-starting position.
x1	X-end position.
y1	Y-end position.

Add. 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	Meaning
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.

Add. 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);
}

```

Screen shot 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	Meaning
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.

Add. 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 \geq the number of points of the polygon.

For more details please refer to `GUI_AA_DrawPolyOutline()`.

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	Meaning
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). Minimum: 0 (will result in a point); maximum: 180.

Add. 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	Meaning
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.

Add. 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.

22.5 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 samples shipped with emWin.

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample code
*
*****
```

```
-----
File      : AA_Lines.c
Purpose   : Shows lines with different antialiasing qualities
-----
*/
```

```
#include "GUI.H"

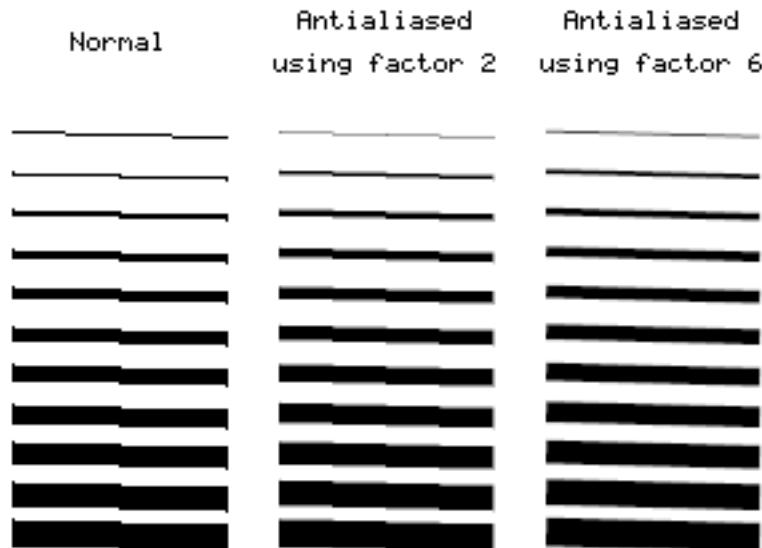
*****
*
*          Show lines with different antialiasing qualities
*
*****
```

```
static void DemoAntialiasing(void) {
    int i, x1, x2;
    int 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 faktor 2 */
    GUI_AA_SetFactor(2);
    GUI_DispStringHCenterAt("Antialiased\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 faktor 6 */
    GUI_AA_SetFactor(6);
    GUI_DispStringHCenterAt("Antialiased\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);
    }
}

*****
*
*          main
*
*****
```

```
void main(void) {
    GUI_Init();
    DemoAntialiasing();
    while(1)
        GUI_Delay(100);
}
```

Screen shot 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 SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample code
*
*****
```

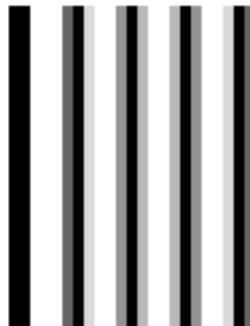
```
File      : AA_HiResPixels.c
Purpose   : Demonstrates high resolution pixels
-----*/
#include "GUI.H"

*****
*
*          Show lines placed on high resolution pixels
*
*****
```

```
static void ShowHiResPixels(void) {
    int i, Factor = 5;
    GUI_SetBkColor(GUI_WHITE);
    GUI_SetColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetLBorder(50);
    GUI_DispStringAt("This sample 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);
    }
}
```

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

Magnified screen shot 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 screen shot of this example because the effects of high-resolution antialiasing are only visible in the movement of the pointers. Without high-resolution mode there is no apparent jumping.

The example can be found as AA_HiResAntialiasing.c.

```
*****
*
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          Solutions for real time microcontroller applications
*
*          emWin sample 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) {
    int i;
    GUI_AUTODEV aAuto[2];
    PARAM Param;
    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);
        }
    }
}

/***********************
*
*          main
*
***********************/

```

```
void main(void) {
    GUI_Init();
    ShowHiresAntialiasing();
}
```


Chapter 23

Foreign Language Support

Text written in a foreign language like Arabic 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.

23.1 Unicode

The Unicode standard is a 16-bit character encoding scheme. All of the characters available worldwide are in a single 16-bit character set (which works globally). The Unicode standard is defined by the Unicode consortium.

emWin can display individual characters or strings in Unicode, although it is most common to simply use mixed strings, which can have any number of Unicode sequences within one ASCII string.

23.1.1 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

Encoding example

The text "Halöle" contains ASCII characters and European extensions. The following hexdump 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");
```

23.1.2 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.

23.1.3 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.

23.1.3.1 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 sample shows how to display the UTF-8 text with emWin:

```
#include "GUI.h"

static const char * _apStrings[] = {
    "Japanese:",
    "1 - \xe3\x82\x80\xe3\x83\x83\xb3\xe3\x82\xb3\xe3\x83\xbc",
    "\xe3\x83\x87\xe3\x82\x80\xe3\x83\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_DisppString(_apStrings[i]);
        GUI_DisppNextLine();
    }
    while(1) {
        GUI_Delay(500);
    }
}
```

23.1.4 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	Explanation
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_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.

23.1.4.1 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 GUI_UNI_PTR * s, int Len,
                           char * pBuffer, int BufferSize);
```

Parameter	Meaning
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.

Return value

The function returns the number of bytes written to the buffer.

Add. 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 GUI_UNI_PTR * s, int Len,
```

```
U16 * pBuffer, int BufferSize);
```

Parameter	Meaning
s	Pointer to UTF-8 string to be converted.
Len	Number of UTF-8 characters to be converted.
pBuffer	Pointer to a buffer to write in the result.
BufferSize	Buffer size in words.

Return value

The function returns the number of Unicode characters written to the buffer.

Add. Information

If the buffer is not big enough for the whole result, the function returns when the buffer is full.

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	Meaning
s	Pointer to a buffer to store the encoded character.
Char	Character to be encoded.

Return value

The number of bytes stored to the buffer.

Add. 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	Meaning
s	Pointer to the text to be encoded.

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	Meaning
s	Pointer to the text to be encoded.

Return value

Number of bytes used to encode the given character

Add. 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);
```

Add. information

After calling this function each byte of a text will be handled as one character. This is the default behaviour of emWin.

GUI_UC_SetEncodeUTF8()**Description**

Enables UTF-8 encoding.

Prototype

```
void GUI_UC_SetEncodeUTF8(void);
```

Add. information

After calling GUI_UC_SetEncodeUTF8 each string related routine of emWin encodes a given sting in accordance to the UTF-8 transformation.

23.1.4.2 Double byte functions**GUI_UC_DisPString()****Description**

This function displays the given double byte string.

Prototype

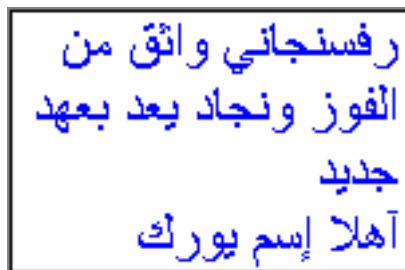
```
void GUI_UC_DispString(const U16 GUI_FAR *s);
```

Parameter	Meaning
s	Pointer to double byte string.

Add. 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.

23.2 Arabic language support



The basic difference between western languages and Arabic is, that Arabic is 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.

23.2.1 Basics

The Arabic character set is defined in the Unicode standard within the range from 0x0600 to 0x06FF. Text written in Arabic language normally contains only letters of this character set. 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 position in the text. One character can have up to 4 different notation forms:

- One, if it is at the beginning of a word
- One, if it is at the end of a word
- One, if it is in the middle of a word
- One, if the character stands alone

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'.

23.2.2 Notation forms

The above explanation shows, that the notation form is normally not identically with 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	0xFEAO	Jeem
0x062D	0xFEAO	0xFEA2	0xFEAO	0xFEAO	Hah
0x062E	0xFEAO	0xFEAO	0xFEAO	0xFEAO	Khah
0x062F	0xFEAA	-	-	-	Dal

Base	Isolated	Final	Initial	Medial	Character
0x0630	0xFEAB	0xFEAC	-	-	Thal
0x0631	0xFEAD	0xFEAE	-	-	Reh
0x0632	0xFEAF	0xFEBO	-	-	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	0xFEFF	0xFEFO	-	-	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	-	-	Farsi Yeh

23.2.3 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)	Ligatur (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

23.2.4 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 the Arabic support please add the following line to the file GUIConf.h:

```
#define GUI_SUPPORT_ARABIC 1
```

If enabled, emWin automatically writes Arabic text from the right to the left and transforms the characters as described above and carrys on writing non Arabic text from the left to the right.

23.2.5 Text alignment

If Arabic support is enabled, Arabic text (characters in the range 0x600-0x6FF) is rendered automatically from the right to the left, and non Arabic text is shown from the left to the right. When a string contains both, Arabic and non Arabic text, the rendering works as shown in the following table:

UTF-8 text	Rendering
"123 \xd8\xb9\xd9\x84\xd8\x a7 \xd8\xba\xd9\x86\xd9\x8a"	عل غني 123
"\xd8\xb9\xd9\x84\xd8\x a7 \xd8\xba\xd9\x86\xd9\x8a 123"	123 عل غني
"\xd8\xb9\xd9\x84\xd8\x a7 123 \xd8\xba\xd9\x86\xd9\x8a"	غني 123 عل

23.2.6 Font files used with Arabic text

Font files used to render Arabic languages 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.

23.3 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.

23.3.1 Creating Shift JIS fonts

The FontConverter 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.

For detailed information on how to create Shift-JIS fonts, please contact SEGGER Microcontroller Systeme GmbH (info@segger.com). A separate FontConverter documentation describes all you need for an efficient way of implementing Shift JIS in your emWin-projects.

23.3.2 Example

The following example defines a small font containing 6 characters: "A", "B", "C" and the Shift JIS characters 0x8350 (KATAKANA LETTER KE), 0x8351 (KATAKANA LETTER GE) and 0x8352 (KATAKANA LETTER KO). A mixed string is then drawn onto the display. The example is available as `FONT_ShiftJIS.c`.

```
/*
 *          SEGGER MICROCONTROLLER SYSTEME GmbH
 *          Solutions for real time microcontroller applications
 *
 *          emWin sample code
 *
 ****
-----  

File      : FONT_ShiftJIS.c  

Purpose   : Example demonstrating ShiftJIS capabilities of emWin  

-----  

*/  

#include "gui.h"  

/*
 *          Definition of ShiftJIS font
 *
 ****
/* LATIN CAPITAL LETTER A */
static const unsigned char acFontsJIS13_0041[ 13 ] = { /* code 0041 */  

    _____,  

    _____,  

    X_____,  

    X_____,  

    X_X_____,  

    X_X_____,  

    _X_X_____,  

    XXXXX_____,  

    _X__X_____,  

    _X_X_____,  

    XXX_XXX_____,  

    _____};  

/* LATIN CAPITAL LETTER B */
static const unsigned char acFontsJIS13_0042[ 13 ] = { /* code 0042 */  

    _____,  

    _____,  

    XXXXX_____,
```

```

_X__X__,
_X_X__,
_X_X__,
XXXX__,
_X_X__,
_X_X__,
_X_X__,
XXXX__,
______;
};

/* LATIN CAPITAL LETTER C */
static const unsigned char acFontSJIS13_0043[ 13] = { /* code 0043 */
______,
______,
XX_X,
_X_XX,
_X_X,
______,
______,
______,
______,
_X_X,
XXX,
______;
};

/* KATAKANA LETTER KE */
static const unsigned char acFontSJIS13_8350[ 26] = { /* code 8350 */
XX_____,______,
_X_____,______,
_X_____,______,
XXXXXX,XXXX__,
_X_X,______,
_X_X,______,
_X_X,______,
_X,______,
_X,______,
_X,______,
_X,______,
_X,______,
_X,______,
_X,______,
_X,______,
_X,______,
XX_____,______;
};

/* KATAKANA LETTER GE */
static const unsigned char acFontSJIS13_8351[ 26] = { /* code 8351 */
XX_____,X_X__,
_X_____,X_X__,
_X_____,______,
XXXXXX,XXX__,
_X_X,______,
_X_X,______,
_X_X,______,
_X,______,
XX_____,______;
};

/* KATAKANA LETTER KO */
static const unsigned char acFontSJIS13_8352[ 26] = { /* code 8352 */
_____,______,
XXXXXX,XX______,
_____,X______,
_____,X______,
_____,X______,
_____,X______,
_____,X______,
_____,X______,
_____,X______,
XXXXXX,XXXX__,
_____,______,
_____,______;
};

static const GUI_CHARINFO GUI_FontSJIS13_CharInfo[6] = {
{ 7, 7, 1, (void *)&acFontSJIS13_0041 } /* code 0041 */,
{ 7, 7, 1, (void *)&acFontSJIS13_0042 } /* code 0042 */,
{ 7, 7, 1, (void *)&acFontSJIS13_0043 } /* code 0043 */,
{ 14, 14, 2, (void *)&acFontSJIS13_8350 } /* code 8350 */,
{ 14, 14, 2, (void *)&acFontSJIS13_8351 } /* code 8351 */
}

```

```

    ,{ 14, 14, 2, (void *)&acFontsSJIS13_8352 } /* code 8352 */
};

static const GUI_FONT_PROP GUI_FontSJIS13_Prop2 = {
    0x8350 /* first character */
    ,0x8352 /* last character */
    ,&GUI_FontSJIS13_CharInfo[ 3 ] /* address of first character */
    ,(void*)0 /* pointer to next GUI_FONT_PROP */
};

static const GUI_FONT_PROP GUI_FontSJIS13_Prop1 = {
    0x0041 /* first character */
    ,0x0043 /* last character */
    ,&GUI_FontSJIS13_CharInfo[ 0 ] /* address of first character */
    ,(void *)&GUI_FontSJIS13_Prop2 /* pointer to next GUI_FONT_PROP */
};

static const GUI_FONT GUI_FontSJIS13 = {
    GUI_FONTTYPE_PROP_SJIS /* type of font */
    ,13 /* height of font */
    ,13 /* space of font y */
    ,1 /* magnification x */
    ,1 /* magnification y */
    ,(void *)&GUI_FontSJIS13_Prop1
};

/******************
*
* Definition of string containing ASCII and ShiftJIS characters
*
***** */
static const char aSJIS[] = {
    "ABC\x83\x50\x83\x51\x83\x52\x0"
};

/******************
*
* Demonstrates output of ShiftJIS characters
*
***** */
void DemoShiftJIS(void) {
    GUI_SetFont(&GUI_Font13HB_1);
    GUI_DispStringHCenterAt("emWin-sample: ShiftJIS characters", 160, 0);
    /* Set ShiftJIS font */
    GUI_SetFont(&GUI_FontSJIS13);
    /* Display string */
    GUI_DispStringHCenterAt(aSJIS, 160, 40);
}

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

```

Screen shot of above example

emWinGSC-sample: ShiftJIS characters

ABCケゲユ

Chapter 24

Display drivers

A display driver supports a particular family of display controllers (typically LCD controllers) and all displays which are connected to one or more of these controllers. The driver is essentially generic, meaning it can be configured by modifying the configuration file `LCDConf.h`. The driver itself does not need to be modified. This file contains all configurable options 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 LCD controllers can be accessed, as well as supported color depths and types of interfaces.
- Additional RAM requirements.
- Additional functions.
- How to access the hardware.
- Special configuration switches.
- Special requirements for particular LCD controllers.

24.1 Available drivers and supported display controllers

The following table lists the available drivers and which display controllers are supported by each:

Driver	Value for macro LCD_CONTROLLER	Display Controller	Supported bits/pixel
LCDColorOnMono	444	Passive color display connected to an display controller with linear memory operating in monochrome mode. Example: Cirrus Logic EP7312	12
LCDFujitsu	8720 8721	Fujitsu MB87J2020 (Jasmine) Fujitsu MB87J2120 (Lavender)	1, 2, 4, 8, 16
LCDLin	1300	Any display controller with linear video memory in 1, 2, 4, 8 and 16 bits/pixel mode, built- in LCD controllers such as Sharp 79531. Epson S1D13700 (direct interface) Solomon SSD1905 Fujitsu MB86290A (Crimson) Fujitsu MB86291 (Scarlet) Fujitsu MB86292 (Orchid) Fujitsu MB86293 (Coral Q) Fujitsu MB86294 (Coral B) Fujitsu MB86295 (Coral P) Toshiba Capricorn 2	1, 2, 4, 8, 16
	1301	Epson S1D13A03, S1D13A04, S1D13A05	
	1304	Epson SED1352, S1D13502	
	1352	Epson SED1353, S1D13503	
	1353	Epson SED1354, S1D13504	
	1354	Epson SED1356, S1D13506	
	1356	Epson SED1374, S1D13704	
	1374	Epson SED1375, S1D13705	
	1375	Epson SED1376, S1D13706	
	1376	Epson SED1386, S1D13806	
LCDMem	3200	Any display controller with linear video memory in 4, 8 or 16 bits/pixel mode (ARM or MIPS CPUs such as Sharp LH754xx, LH79520, Motorola Dragonball or NEC VR4181A) which should be accessed only in 32 bit mode.	1, 2
	0	No controller, writes into main memory Requires ISR or special hardware to refresh LCD (monochrome displays)	
LCDMemC	0	No controller, writes into main memory Requires ISR or special hardware to refresh LCD (color displays)	3, 6
LCDNoritake	9000	Noritake display GU256X128C-3900	1

Driver	Value for macro LCD_CONTROLLER	Display Controller	Supported bits/pixel
LCDPage1bpp	1501 1501 1501 1502 1503 1504 1505 1506 1507 1508 1520 1560 1565 1565 1565 1565 1565 1566 1567 1568 1569 1575	Samsung KS0713, Samsung S6B0713 Samsung KS0724, Samsung S6B0724 UltraChip UC1606 Samsung KS0108B, Samsung S6B0108B Hitachi HD61202 Philips PCF8810, PCF8811 Philips PCF8535 New Japan Radio Company NJU6679 Philips PCD8544 Epson S1D15710 Epson SED1520 Epson SED1560 Epson SED1565, S1D10605 Novatek NT7502 Samsung S6B1713 Solomon SSD1815 Sitronix ST7565 Sunplus SPLC501C Epson SED1566 Epson SED1567 Epson SED1568 Epson SED1569 Epson SED1575, S1D15705	1
LCDPage4bpp	7528	Sitronix ST7528	4
LCDSLin	6901 6901 6901 6963	Epson SED1330 Epson SED1335 Epson S1D13700 (indirect interface) Toshiba T6963	1
LCDVesa	8600	Any VESA compatible hardware	8, 16
LCDXylon	9100	FPGA based display controller from Xylon	8
LCD0323	323	Solomon SSD0323 OLED controller	4
LCD07X1	701 702 711 741 741	Solomon SSD1854 STMicroelectronics STE2010 Samsung KS0711, Samsung S6B0711 Samsung KS0741, Samsung S6B0741 Sitronix ST7541	2
LCD1200	1200 1201	Toppoly C0C0 Toppoly C0E0	16
LCD13701	13701	Epson S1D13701 OLED controller	9, 12
LCD159A	1601	Epson SED159A	8
LCD15E05	1701	Epson S1D15E05	2
LCD1611	1800 1801	UltraChip UC1611 UltraChip UC1610	2, 4
LCD161620	180	NEC µPD161620	12
LCD1781	1781	Solomon SSD1781	16
LCD501	501	Leadis LDS501	8
LCD6331	6331	Samsung S6B33B0X, Samsung S6B33B1X, Samsung S6B33B2X	16
LCD6642X	66420 66421	Hitachi HD66420 Hitachi HD66421	2
LCD667XX	66766 66766 66772 66789	Hitachi HD66766 Samsung S6D0110A Hitachi HD66772 Hitachi HD66789	16
LCD66750	66750 66753	Hitachi HD66750 Hitachi HD66753	2

The basic package contains 2 drivers which don't support a specific LCD controller. They can be used as template for a new driver or for measurement purpose:

Driver	Value for macro LCD_CONTROLLER	LCD Controller	Supported bits/pixel
LCDTemplate	-1	Driver template. Can be used as a starting point for writing a new driver. Part of the basic package	-
LCDNull	-2	Empty driver. (Performs no output) Can be used for measurement purpose. Part of the basic package.	-

Selecting a driver

As described in Chapter 27: "Low-Level Configuration", the macro `LCD_CONTROLLER` defines the LCD controller used. A controller is specified by its appropriate value, listed in the table above.

The following sections discuss each of the available drivers individually.

24.2 CPU / Display controller interface

Different display controllers have different CPU interfaces. The most common ones are the following:

- Full bus interface
- Simple bus interface
- 4 pin SPI interface
- 3 pin SPI interface
- I2C bus interface

Below we explain these interfaces and how to configure them. Note that not all config macros are always required. For details about which macros are required please take a look to the driver documentation later in this chapter. The Chapter 27: "Low-Level Configuration", explains the macros itself.

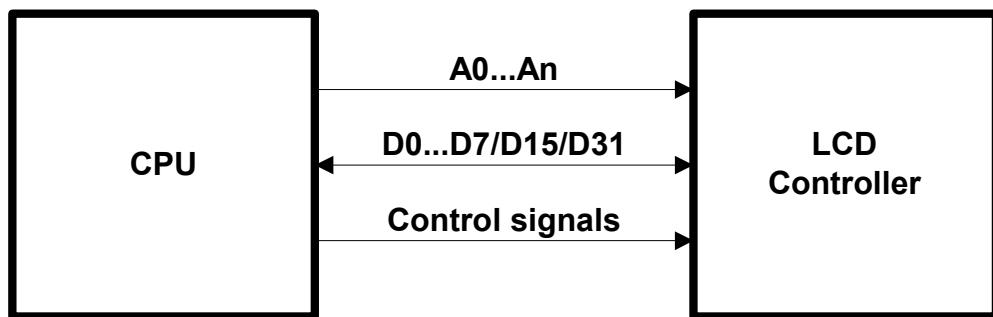
24.2.1 Full bus interface

Some LCD 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 full bus interface configuration, video memory is directly accessible by the CPU; the full-address bus is connected to the LCD controller.

The only knowledge required when configuring a full bus interface is information about the address range (which will generate a CHIP-SELECT signal for the LCD controller) and whether 8- or 16-bit accesses should be used (bus-width to the LCD 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 for LCD controllers with full bus interface



Macros used by a full bus interface

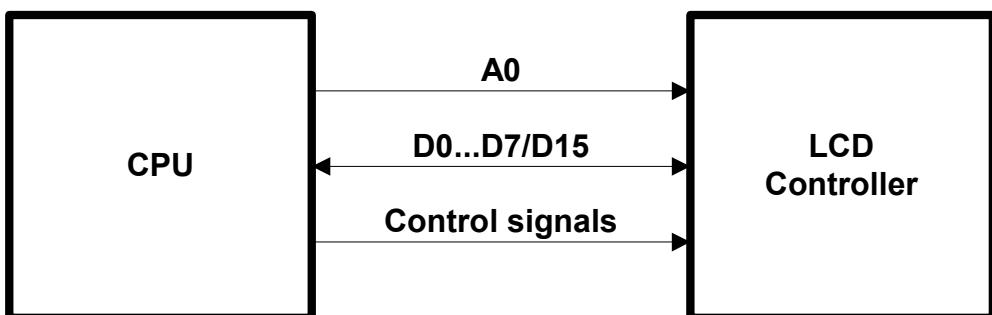
The following table shows the used hardware access macros:

Type	Macro	Explanation
F	LCD_READ_MEM	Reads the video memory of the LCD controller.
F	LCD_WRITE_MEM	Writes data to the video memory of the LCD controller.
F	LCD_READ_REG	Reads the register of the LCD controller.
F	LCD_WRITE_REG	Writes data to a specified register of the LCD controller.

24.2.2 Simple bus interface

Most LCD controllers for smaller displays (usually up to 240*128 or 320*240) use a simple bus interface to connect to the CPU. With a simple bus, 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 for LCD controllers with simple bus interface



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.

Macros used by a simple bus interface

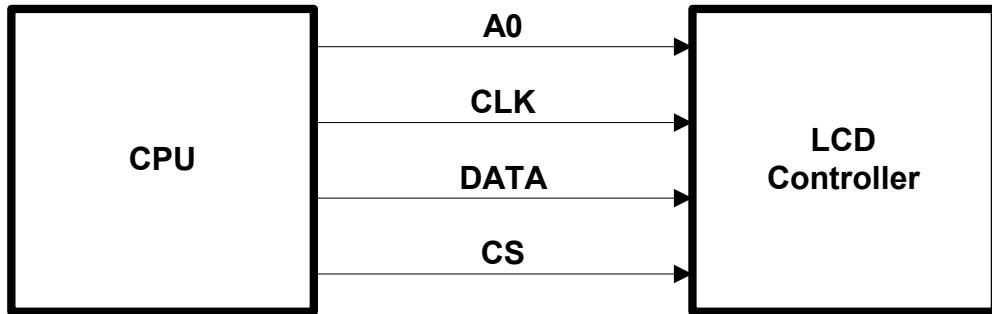
The following table shows the used hardware access macros:

Type	Macro	Explanation
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 LCD controller with A0 - line low.
F	LCD_WRITE_A1	Writes a byte to LCD controller with A0 - line high.
F	LCD_WRITEITEM_A1	Writes several bytes to the LCD controller with A0 - line high.

24.2.3 4 pin SPI interface

Using a 4 pin SPI interface is very similar to a simple bus 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 for LCD controllers with 4 pin SPI interface



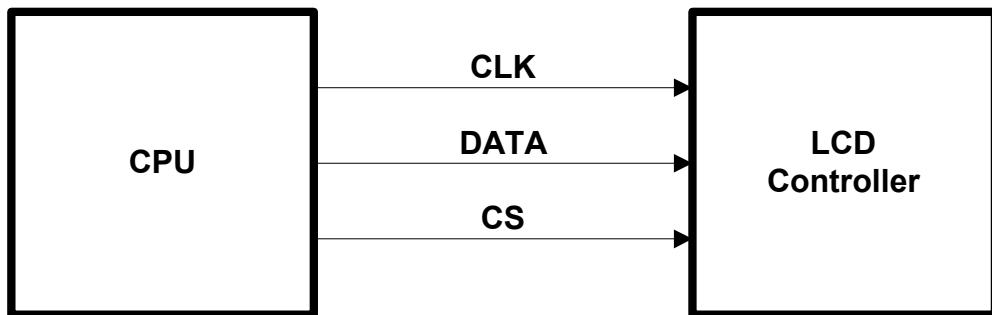
Macros used by a 4 pin SPI interface

The following table shows the used hardware access macros:

Type	Macro	Explanation
F	LCD_WRITE_A0	Writes a byte to LCD controller with A0 (C/D) - line low.
F	LCD_WRITE_A1	Writes a byte to LCD controller with A0 (C/D) - line high.
F	LCD_WRITEITEM_A1	Writes several bytes to the LCD controller with A0 (C/D) - line high.

24.2.4 3 pin SPI interface

Typical block diagram for LCD controllers with 3 pin SPI interface



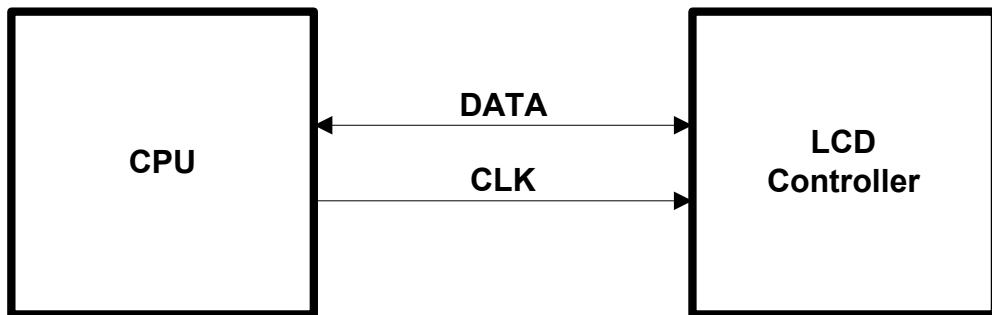
Macros used by a 3 pin SPI interface

The following table shows the used hardware access macros:

Type	Macro	Explanation
F	LCD_WRITE	Writes a byte to LCD controller.
F	LCD_WRITEITEM	Writes several bytes to the LCD controller.

24.2.5 I2C bus interface

Typical block diagram for LCD controllers with I2C bus interface



Macros used by a 3 pin SPI interface

The following table shows the used hardware access macros:

Type	Macro	Explanation
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 LCD controller.
F	LCD_WRITE_A1	Writes a data byte to LCD controller.
F	LCD_WRITEITEM_A1	Writes several data bytes to the LCD controller.

24.3 Detailed display driver descriptions

24.3.1 LCDColorOnMono driver

This driver supports systems with passive color displays connected to an LCD controller in monochrome mode. The LCD controller is assumed to have a linear video memory organization and to work in a 16 gray-scale mode, yielding 16 levels of intensity for every sub-pixel and therefore $16 \times 16 \times 16 = 4096$ colors on the display.

If the LCD controller supports colors, it is usually better to use it in color mode (with the LCDLin driver), as this more efficient.

An example of where this driver is used is the LCD controller built-into the Cirrus Logic EP7312.

Supported hardware

Controllers

#	LCD controller	Add. info
444	Any Passive color display connected to an LCD controller with linear memory operating in monochrome mode.	-

Bits per pixel

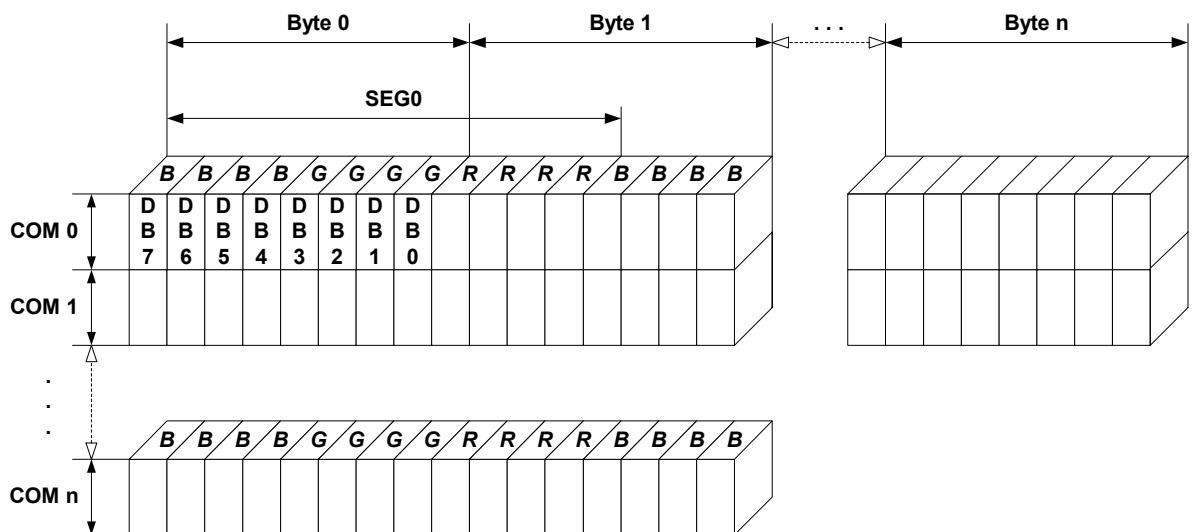
Supported color depth is 12 bpp.

Interfaces

The driver supports a 16 bit full bus interface from the CPU to the LCD controller.

Display data RAM organization

12 bits per pixel, fixed palette = 444



This driver supports a 12 bpp memory area for color displays. The pictures above show the dependence between the memory area handled by the driver and the SEG and COM lines of the LCD.

RAM requirements of the driver

None.

Additional driver functions

None.

Hardware configuration

This driver requires a full bus interface for hardware access as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the LCD controller.
LCD_READ_MEM	Read the contents of video memory of controller.
LCD_WRITE_MEM	Write to video memory (display data RAM) of controller.

Example:

A QVGA color display needs the following defines in LCDConf.h:

```
#define LCD_XSIZE 320
#define LCD_YSIZE 240
```

The LCD controller needs to be initialized to control a monochrome display of 960x240 resolution.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
LCD_ON	Function replacement macro which switches the LCD on.
LCD_OFF	Function replacement macro which switches the LCD off.

Special requirements

The driver needs to work in one of the following fixed palette modes:

- 44412 (default if working in 12bpp mode)
- 444121
- 44416

The driver does not work with other palettes.

24.3.2 LCDFujitsu driver

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

#	LCD controller	Add. info
8720	Jasmin	Fully optimized, LUT is initialized
8721	Lavender	Fully optimized, LUT is initialized

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.

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 of the driver

About 16 bytes for some static variables.

Additional driver functions

None.

Hardware configuration

This driver requires a full bus interface for hardware access as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_REG</code>	Read a register of the display controller. (as 32 bit value) (optional)
<code>LCD_WRITE_REG</code>	Write a register of the display controller. (as 32 bit value) (optional)

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, Jasmin

```
#define LCD_XSIZE      640 /* X-resolution of LCD, Logical coor. */
#define LCD_YSIZE      480 /* Y-resolution of LCD, Logical coor. */
#define LCD_BITSPERPIXEL 8
#define LCD_CONTROLLER 8720 /* Jasmine */
```

Additional configuration switches

The following table shows optional configuration macros available for this driver:

Macro	Explanation
LCD_ON	Function replacement macro which switches the display on.
LCD_OFF	Function replacement macro which switches the display off.

Special requirements

None

24.3.3 LCDLin driver (8 and 16 bit access)

Generally display controller with linear video memory can be accessed with the LCDLin driver for 8 and 16 bit access and with the LCDLin driver for 32 bit access. If 32 bit access is possible, it is recommended to use the 32 bit driver with the better performance.

This driver can be used with any LCD Controller with linear memory organization (as described below) and full bus interface (8 or 16 bit data bus). Most controllers for bigger displays and higher color depth (typically starting at quarter VGA) comply with this requirement and can therefore be controlled by this driver.

Supported hardware

Controllers

The following table list the supported controllers and their assigned numbers for LCD_CONTROLLER, as well as the level of support:

#	LCD controller	Add. info
1300	Any LCD controller with linear memory and full bus interface, such as: Epson SED1352, S1D13502 Epson SED1353, S1D13503 Epson S1D13700 (direct interface) Solomon SSD1905 Fujitsu MB86290A (Crescent) Fujitsu MB86291 (Scarlet) Fujitsu MB86292 (Orchid) Fujitsu MB86293 (Coral Q) Fujitsu MB86294 (Coral B) Fujitsu MB86295 (Coral P) Microcontrollers with built-in LCD controllers such as Sharp LH79531	The LUT (color look up table) is not handled by the driver. If a LUT mode is used (typically 16 or 256 colors), the application program is responsible for initialization of the LUT.
1301	Toshiba Capricorn 2	LUT is handled by driver if required All layers can be supported
1304	Epson S1D13A03, S1D13A04, S1D13A05	LUT is handled by driver if required 2 D Engine supported (BitBLT)
1354	Epson SED1354, S1D13504	LUT is handled by driver if required
1356	Epson SED1356, S1D13506	LUT is handled by driver if required. 2 D Engine supported (BitBLT)
1374	Epson SED1374, S1D13704	LUT is handled by driver if required
1375	Epson SED1375, S1D13705	LUT is handled by driver if required
1376	Epson SED1376, S1D13706	LUT is handled by driver if required
1386	Epson SED1386, S1D13806	LUT is handled by driver if required 2 D Engine supported (BitBLT)

Bits per pixel

Supported color depths are 1, 2, 4, 8 and 16 bpp.

Interfaces

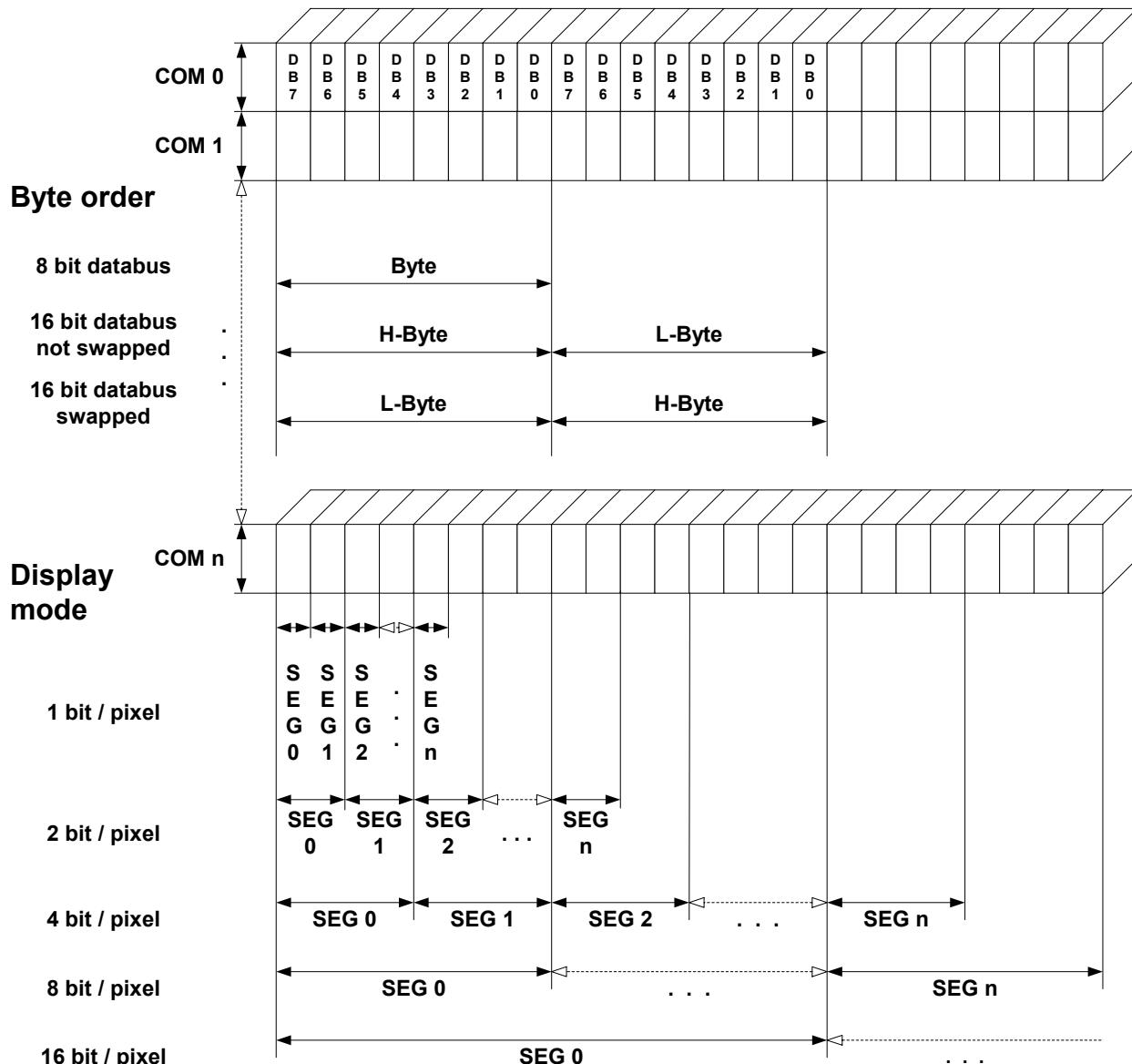
The chips supported by this driver can be interfaced in 8/16-bit parallel (full bus) modes.

The driver supports both interfaces. Please refer to the respective LCD controller manual in order to determine if your chip can be interfaced in 8-bit mode.

Built-in LCD controllers

This driver can also be used with built-in LCD controllers. In this case, either 8 or 16 bit access can be selected. It is typically best to use 8 bit access if the built-in LCD controller operates in an 8-bpp mode and likewise to use 16 bit access if the built-in LCD-controller operates in a 16-bpp mode.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD in terms of the color depth.

Additional RAM requirements of the driver

None.

Additional driver functions

None.

Hardware configuration

This driver requires a full bus interface for hardware access as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_MEM</code>	Read the contents of video memory of controller.

Macro	Explanation
LCD_READ_REG	Read the contents of a configuration register of controller.
LCD_WRITE_MEM	Write to video memory (display data RAM) of controller.
LCD_WRITE_REG	Write to a configuration register of controller.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
LCD_BUSWIDTH	Select bus-width (8/16) of LCD controller/CPU interface. Default is 16.
LCD_CNF4	Endian mode selection for S1D13A03-A05 controllers. Default is 0.
LCD_ENABLE_MEM_ACCESS	Switch the M/R signal to memory access. Only used for S1D13506 and S1D13806 LCD controllers.
LCD_ENABLE_REG_ACCESS	Switch the M/R signal to register access. Only used for S1D13506 and S1D13806 LCD controllers.
LCD_ON	Function replacement macro which switches the LCD on
LCD_OFF	Function replacement macro which switches the LCD off
LCD_SET_LUT_ENTRY	Function replacement macro used to set a single lookup table or palette RAM entry.
LCD_SWAP_BYTE_ORDER	Inverts the endian mode (swaps the high and low bytes) between CPU and LCD controller when using a 16-bit bus interface.
LCD_USE_BITBLT	If set to 0, it disables the BitBLT engine. If set to 1 (the default value), the driver will use all available hardware acceleration.

Additional info for S1D13A03, S1D13A04 and S1D13A05

LCD_CNF4

The configuration switch LCD_CNF4 configures the endian mode selection. If the CNF4 pin of the controller is configured as high the macro should be 1, if the pin is low it should be 0 (default). To set the endian mode to big endian the following line should be added to LCDConf.h:

```
#define LCD_CNF4 (1) /* Selects the big endian mode */
```

Additional info for S1D13806, S1D13A03, S1D13A04 and S1D13A05

LCD_SWAP_RB

The configuration switch LCD_SWAP_RB (swaps the red and blue components) must be activated (set to 1) by inserting the following line into LCDConf.h:

```
#define LCD_SWAP_RB (1) /* Has to be set */
```

LCD_INIT_CONTROLLER

When writing or modifying the initialization macro, consider the following:

- To initialize the embedded SDRAM, bit 7 of register 20 (SDRAM initialization bit) must be set to 1 (a minimum of 200 µs after reset).
- When the SDRAM initialization bit is set, the actual initialization sequence occurs at the first SDRAM refresh cycle. The initialization sequence requires approximately 16 MCLKs to complete, and memory accesses cannot be made while the initialization is in progress.

For more information, please see the LCD controller documentation.

LCD_READ_REG, LCD_WRITE_REG

In order for the BitBLT engine to work, the data type of the offset must be unsigned long. This is set with the configuration macros LCD_READ_REG and LCD_WRITE_REG as follows:

```
#define LCD_READ_REG(Off)      *((volatile U16*)(0x800000+(((U32)(Off))<<1)))
#define LCD_WRITE_REG(Off,Data)  *((volatile U16*)(0x800000+(((U32)(Off))<<1)))=Data
```

24.3.4 LCDLin driver (32 bit access)

Generally display controller with linear video memory can be accessed with the LCD-Lin driver for 8 and 16 bit access and with the LCDLin driver for 32 bit access. If 32 bit access is possible, it is recommended to use the 32 bit driver with the better performance.

This driver can be used with any display controller with linear memory organization (as described below) and full bus interface (32 bit data bus). Most controllers for bigger displays and higher color depth (typically starting at quarter VGA) comply with this requirement and can therefore be controlled by this driver.

Supported hardware

Controllers

The following table list the supported controllers and their assigned numbers for LCD_CONTROLLER, as well as the level of support:

#	LCD controller	Add. info
3200	Any display controller with linear video memory in 4, 8 or 16 bits/pixel mode such as the build in display controller of Sharp LH754XX, ARM or MIPS CPU's.	The LUT (color look up table or palette RAM) is not handled by the driver. If a LUT mode is used, the application program is responsible for initialization of the LUT. LUT support can be added by using the macro LCD_SET_LUT_ENTRY.

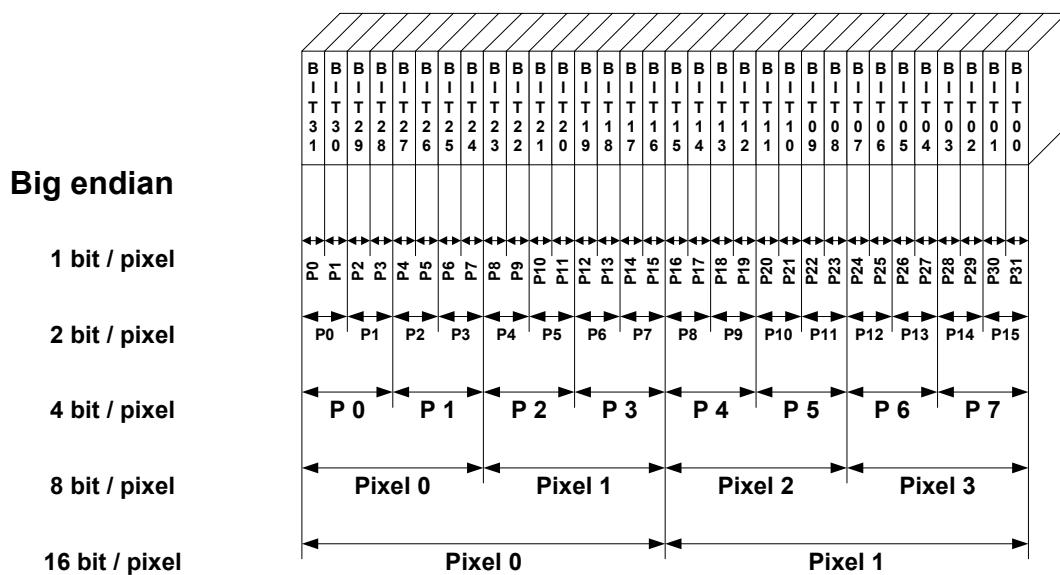
Bits per pixel

Supported color depths are 1, 2, 4, 8 and 16 bpp.

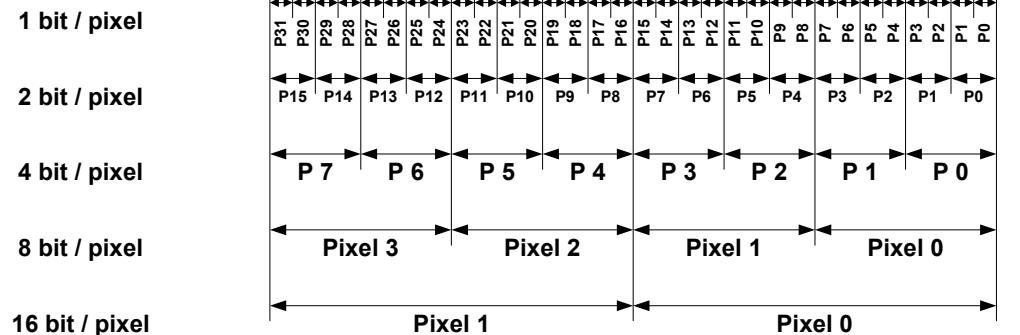
Interfaces

The driver supports any 32 bit full bus interface.

Display data RAM organization



Little endian



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.

Additional RAM requirements of the driver

None.

Additional driver functions

None.

Hardware configuration

This driver requires a full bus interface for hardware access as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_ENDIAN_BIG	Should be set to 1 for big endian mode, 0 for little endian mode.
LCD_INIT_CONTROLLER	Initialization sequence for the LCD controller.
LCD_VRAM_ADDR	Defines the start address of the video memory.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
LCD_FILL_RECT	Function replacement macro which can be used to draw a rectangle.
LCD_LIN_SWAP	Swaps pixels within a byte.
LCD_OFF	Function replacement macro which switches the LCD off.
LCD_ON	Function replacement macro which switches the LCD on.
LCD_SET_LUT_ENTRY	Used to set a single lookup table or palette RAM entry.

LCD_LIN_SWAP()

Description

This macro enables swapping of pixels within one byte.

Type

Numeric.

Prototype

```
#define LCD_LIN_SWAP
```

Add. information

Sometimes a display driver like the embedded display driver of the Motorola MX1 has a different pixel assignment as the default assignment shown under 'Display data RAM organization'. In this case the macro LCD_LIN_SWAP can be used to swap the pixels within one byte after reading from and before writing to the video RAM.

If the value of the macro is > 0, pixel swapping is activated. The value of LCD_LIN_SWAP defines the swapping mode. The following table shows the supported swapping modes in dependence of the defined value:

Value	Default	After swapping																
1	<table border="1"><tr><td>P7</td><td>P6</td><td>P5</td><td>P4</td><td>P3</td><td>P2</td><td>P1</td><td>P0</td></tr></table>	P7	P6	P5	P4	P3	P2	P1	P0	<table border="1"><tr><td>P0</td><td>P1</td><td>P2</td><td>P3</td><td>P4</td><td>P5</td><td>P6</td><td>P7</td></tr></table>	P0	P1	P2	P3	P4	P5	P6	P7
P7	P6	P5	P4	P3	P2	P1	P0											
P0	P1	P2	P3	P4	P5	P6	P7											
2	<table border="1"><tr><td>P3</td><td>P2</td><td>P1</td><td>P0</td></tr></table>	P3	P2	P1	P0	<table border="1"><tr><td>P0</td><td>P1</td><td>P2</td><td>P3</td></tr></table>	P0	P1	P2	P3								
P3	P2	P1	P0															
P0	P1	P2	P3															
4	<table border="1"><tr><td>P1</td><td>P0</td></tr></table>	P1	P0	<table border="1"><tr><td>P0</td><td>P1</td></tr></table>	P0	P1												
P1	P0																	
P0	P1																	

Example

```
#define LCD_LIN_SWAP 1
```

LCD_FILL_RECT()

Description

This macro can be used for calling a user defined routine by the driver for filling.

Type

Function replacement.

Prototype

```
#define LCD_FILL_RECT(x0, y0, x1, y1, Index)
```

Parameter	Meaning
x0	Leftmost X-position of the rectangle to be filled.
y0	Topmost Y-position of the rectangle to be filled.
x1	Rightmost X-position of the rectangle to be filled.
y1	Bottommost Y-position of the rectangle to be filled.
Index	Color index to be used for filling.

Add. information

If this macro is defined, the driver calls the function defined by this macro instead of using its own filling routine. Using this macro can make sense if for example a BitBLT engine should be used for filling instead of the driver internal filling function. Index values are in the range of 0 - ((1 << LCD_BITS_PER_PIXEL) - 1).

Example

```
void CustomFillRect(int x0, int y0, int x1, int y1, int Index);
#define LCD_FILL_RECT(x0, y0, x1, y1, Index) CustomFillRect(x0, y0, x1, y1, Index)
```

How to migrate from LCDLin to LCDLin32

The driver for 8 and 16 bit access needs the definition of 2 memory access macros, LCD_READ_MEM and LCD_WRITE_MEM. The driver for 32 bit access needs the definition of the display RAM memory address. The following sample shows how to define the memory access.

Example

Configuration for 16 bit access:

```
#define LCD_CONTROLLER 1300
#define LCD_READ_MEM(Off)      *((U16*)(0xc00000 +(((U32)(Off)) << 1)))
#define LCD_WRITE_MEM(Off,Data) *((U16*)(0xc00000 +(((U32)(Off)) << 1))) = Data
```

Configuration for 32 bit access:

```
#define LCD_CONTROLLER 3200
#define LCD_VRAM_ADR    0xc00000
```

24.3.5 LCDMem driver

Using the CPU as LCD controller

In systems with relatively fast CPUs and small (quarter VGA or less) LCDs, there is no need for an LCD controller. The microcontroller (CPU) can do the job of the LCD controller on the side, refreshing the display in an interrupt service routine. The CPU's memory is used as video memory.

Advantages of this approach include the following:

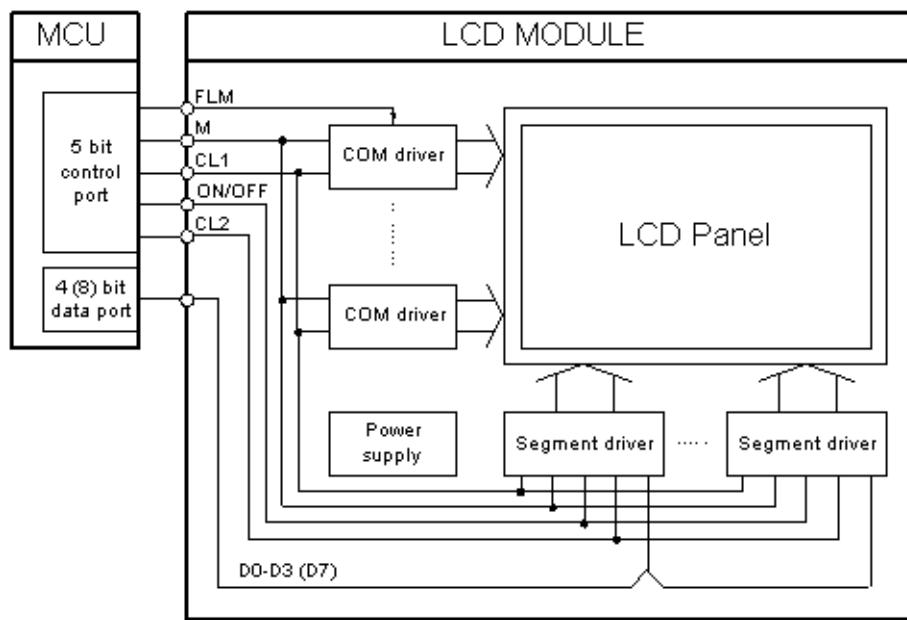
- Very fast update of display possible.
- Eliminating the LCD controller (and its external RAM) reduces hardware costs.
- Simplified hardware design.
- 4 levels of gray can be displayed.

The disadvantage is that much of the available computation time is used up. 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 LCD driver because emWin simply places all the display data into the LCD cache. You yourself must write the hardware-dependent portion that periodically transfers the data in the cache memory to your LCD. Sample code for transferring the video image into the display is available in both "C" and optimized assembler for M16C and M16C/80. The assembler files can be found in the folder `LCDDriver` of emWin. The output routine in 'C' can be found under `Sample\LCDConf\LCDMem\LCD_ISR.c`.

How to connect the CPU to the row/column drivers

It is quite easy to connect the microcontroller to the row/column drivers. Five control lines are needed, as well as either 4 or 8 data lines (depending on whether the column drivers are able to operate in 8-bit mode). 8-bit mode is recommended as it is more efficient, saving calculation time of the CPU. All data lines should be on a single port, using port bits 0..3 or 0..7 in order to guarantee efficient access. This setup is illustrated below:



CPU load

The CPU load depends on the hardware and controller used, as well as on the size of the display. For example:

Mitsubishi M16C62 Controller, 16MHz, 160*100 display, 8-bit interface, 80 Hz update
= app. 12% CPU load.

Mitsubishi M16C62 Controller, 16MHz, 240*128 display, 8-bit interface, 80 Hz update
= app. 22% CPU load.

Supported hardware

Controllers

None.

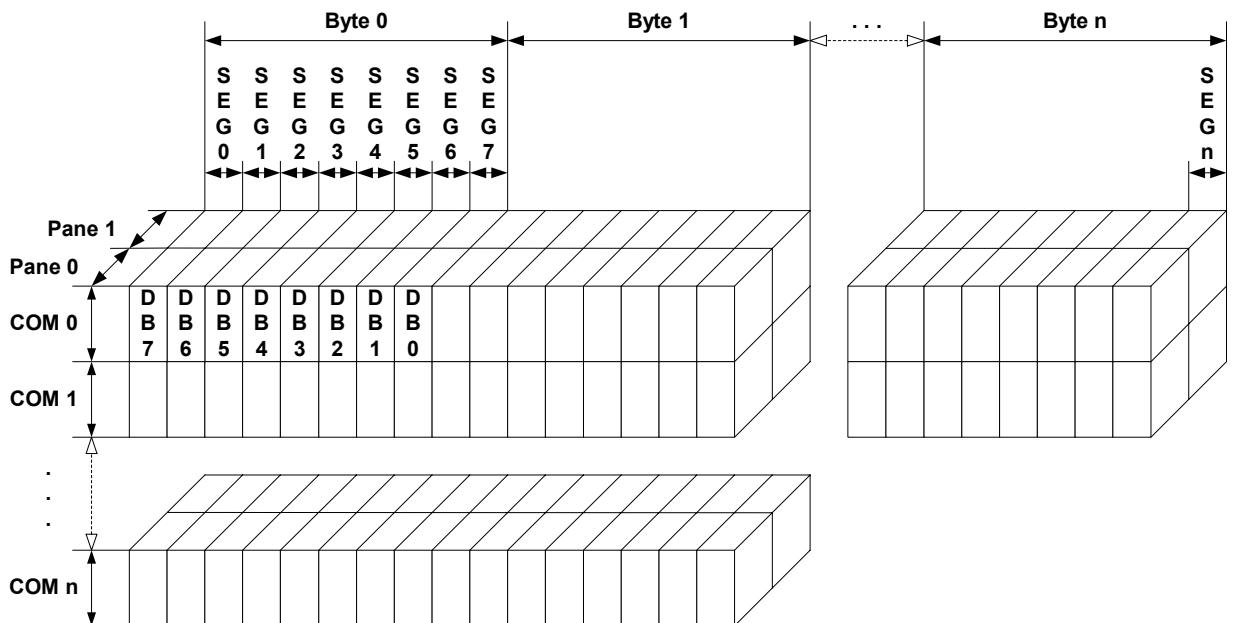
Bits per pixel

Supported color depth is 1 and 2 bpp.

Interfaces

The driver supports 1/4/8-bit interfaces from the CPU to the LCD.

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 two panes for each pixel. The lower bit of one pixel is stored in pane 0 and the higher bit is stored in pane 1. The advantage of this method is that the output of the display data can be executed very quickly. If working in 1 bpp mode only one pane will be used.

RAM requirements of the driver

The driver only handles a memory area containing the display data. The required size of the display memory area may be calculated as follows:

1 bit per pixel

Size of RAM (in bytes) = $(LCD_XSIZE + 7) / 8 * LCD_YSIZE$

2 bit per pixel

Size of RAM (in bytes) = $(LCD_XSIZE + 7) / 8 * LCD_YSIZE * 2$

Additional driver functions

None.

Hardware configuration

Normally, the hardware interface is an interrupt service routine (ISR) which updates the LCD. An output routine written in "C" code is shipped with emWin. This routine should serve only as an example. To optimize the execution speed, it must be adapted in assembler code.

For detailed information on how to write the output routine, please take a look at the sample supplied with the driver or contact us.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<code>LCD_TIMERINIT0</code>	Timing value used by ISR for displaying pane 0.
<code>LCD_TIMERINIT1</code>	Timing value used by ISR for displaying pane 1.
<code>LCD_ON</code>	Function replacement macro which switches the LCD on.
<code>LCD_OFF</code>	Function replacement macro which switches the LCD off.

24.3.6 LCDMemC driver

This driver, like LCDMem, is designed for a system without an LCD controller. The difference is that LCDMemC supports color displays. For more information on using the CPU instead of an LCD controller, please see the previous section on the LCDMem driver.

Supported hardware

Controllers

None.

Bits per pixel

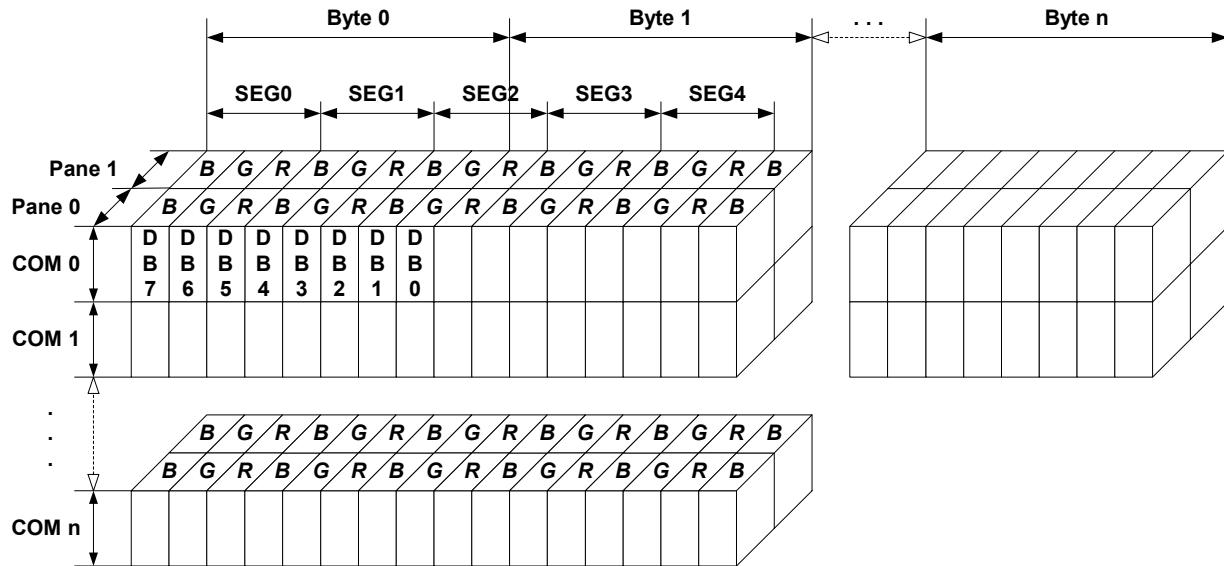
Supported color depths are 3 and 6 bpp.

Interfaces

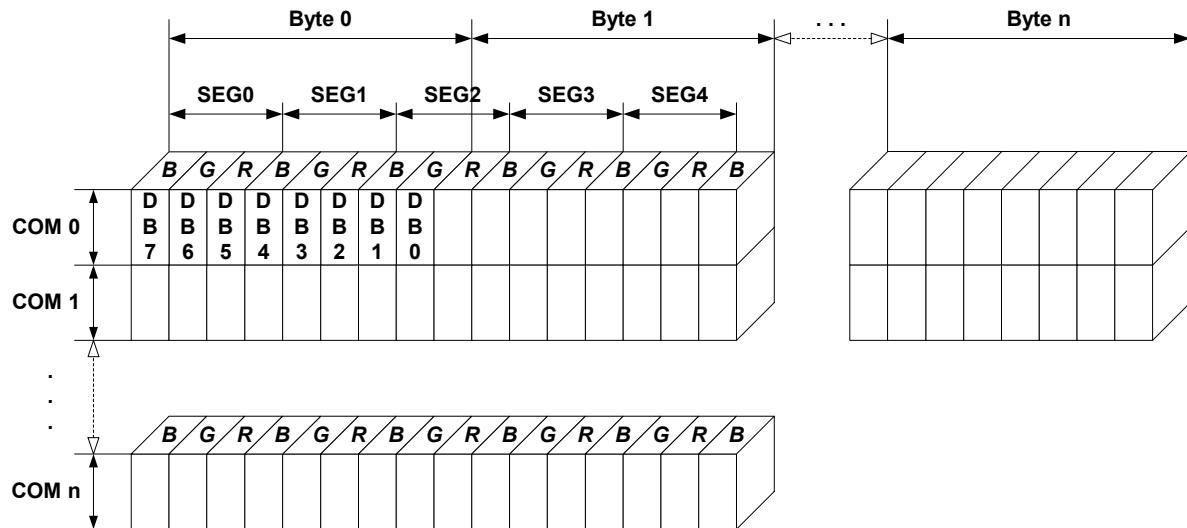
The driver supports 1/4/8-bit interfaces from the CPU to the LCD.

Display data RAM organization

6 bits per pixel, fixed palette = 222



3 bits per pixel, fixed palette = 111



This driver supports a 3 or 6 bpp memory area for color displays. The pictures above show the dependence between the memory area handled by the driver and the SEG and COM lines of the LCD.

6 bits per pixel, fixed palette mode 222

When using the 6 bpp mode, the display memory is divided into 2 panes for each pixel. The lower bit of each pixel is stored in pane 0 and the higher bit is stored in pane 1. The advantage of this method is that the output of the display data can be executed very quickly.

3 bits per pixel, fixed palette mode 111

When using this mode, only one pane exists for each pixel.

RAM requirements of the driver

The driver only handles a memory area containing the display data. The required size of the display memory area may be calculated as follows:

6 bits per pixel, fixed palette mode 222

Size of RAM (in bytes) = $(LCD_XSIZE + 7) / 8 * LCD_YSIZE * 3 * 2$

3 bits per pixel, fixed palette mode 111

Size of RAM (in bytes) = $(LCD_XSIZE + 7) / 8 * LCD_YSIZE * 3$

Additional driver functions

None.

Hardware configuration

Normally, the hardware interface is an interrupt service routine (ISR) which updates the LCD. An output routine written in "C" code is shipped with emWin. This routine should serve only as an example. To optimize the execution speed, it must be adapted in assembler code.

For detailed information on how to write the output routine, please take a look at the sample supplied with the driver or contact us.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
LCD_TIMERINIT0	Timing value used by ISR for displaying pane 0.
LCD_TIMERINIT1	Timing value used by ISR for displaying pane 1 (only used by 6 bpp mode).
LCD_ON	Function replacement macro which switches the LCD on.
LCD_OFF	Function replacement macro which switches the LCD off.

24.3.7 LCDNoritake display driver

Supported hardware

Displays

The driver has been tested with the following display:

- Noritake GU256X128C_3900 configured for the 'Graphic DMA Mode'

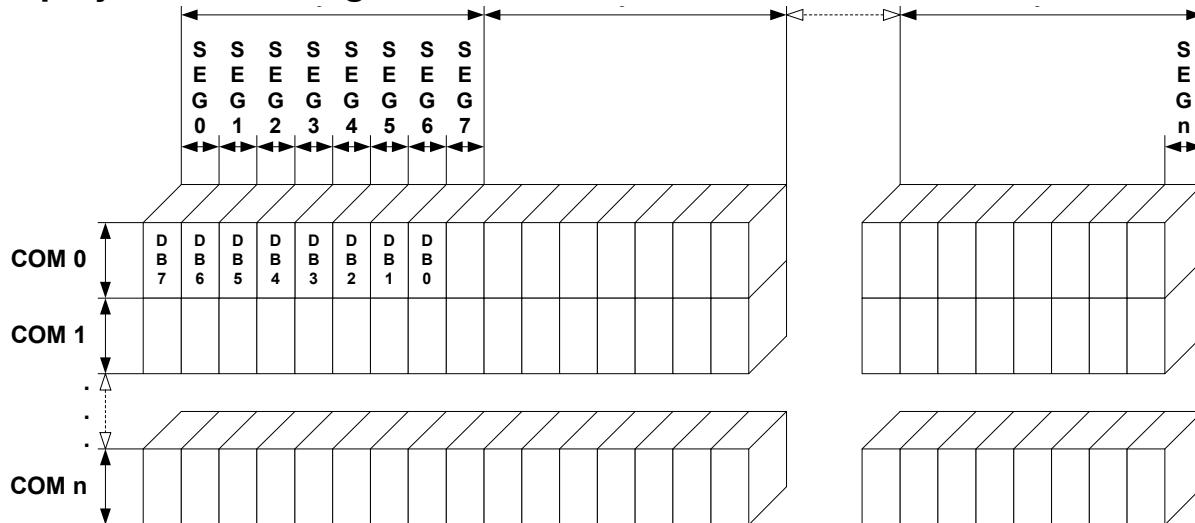
Bits per pixel

Supported color depth is 1 bpp.

Interfaces

The driver supports the 8-bit parallel (simple bus) interface.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM-requirement

This driver requires a display data cache, containing a complete copy of the contents of the display data RAM. The amount of memory used by the cache may be calculated as follows:

$$\text{Size of RAM (in bytes)} = (\text{LCD_XSIZE} + 7) / 8 * \text{LCD_YSIZE}$$

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which should be defined for hardware access:

Macro	Explanation
LCD_DAD	Display address used for the communication protocoll. Default value is display address 0.
LCD_INIT_CONTROLLER	Initialization sequence for the display.
LCD_WRITE_A1	Write a byte to the display.
LCD_WRITEITEM_A1	Write multiple bytes to the display.

24.3.8 LCDPage1bpp driver

Supported hardware

Controllers

This driver has been tested with the following LCD controllers:

- Epson SED1520, SED1560, SED1565, SED1566, SED1567, SED1568, SED1569, SED1575, S1D15710, S1D10605, S1D15705
- Hitachi HD61202
- New Japan Radio Company NJU6679
- Novatec NT7502
- Philips PCF8810, PCF8811, PCF8535, PCD8544
- Samsung KS0713, KS0724, KS0108B, S6B1713, S6B0724, S6B0108B, S6B1713
- Sitronix ST7565
- Solomon SSD1815
- Sunplus SPLC501C
- UltraChip UC1606

It should be assumed that it will also work with any controller of similar organization.

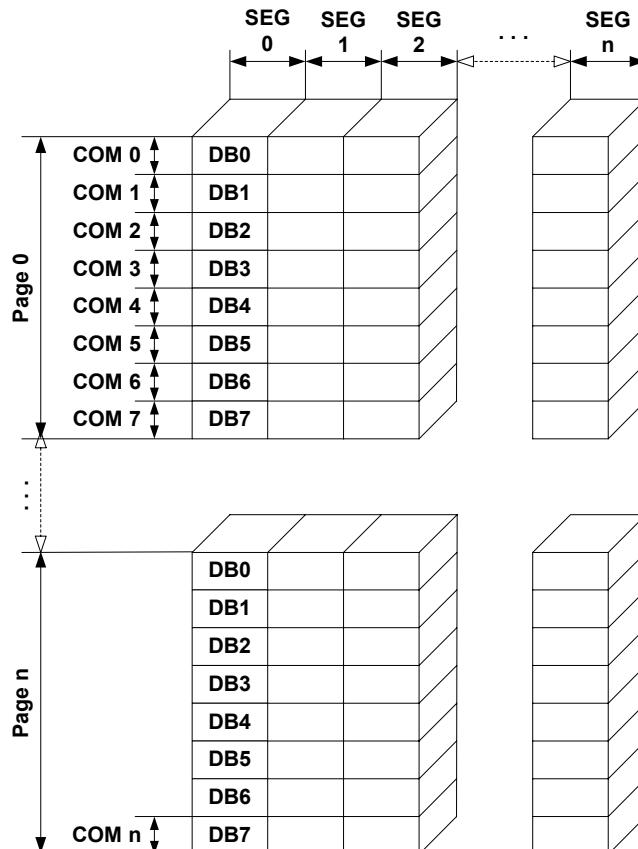
Bits per pixel

Supported color depth is 1bpp.

Interfaces

8-bit parallel (simple bus), serial (SPI4) and I2C bus interfaces are supported.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver can be used with or without a display data cache in the most cases. If one display contains more than 1 LCD controller you can not disable the cache. The data cache contains a complete copy of the contents of the LCD 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 LCD-access. The amount of memory used by the cache may be calculated as follows:

$$\text{Size of RAM (in bytes)} = (\text{LCD_YSIZE} + 7) / 8 * \text{LCD_XSIZE}$$

Additional driver functions

LCD_L0_ControlCache

For information about this function, please refer to Chapter 28: "LCD Driver API".

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_A0</code>	Read a byte from LCD controller with A-line low.
<code>LCD_READ_A1</code>	Read a byte from LCD controller with A-line high.
<code>LCD_WRITE_A0</code>	Write a byte to LCD controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to LCD controller with A-line high.
<code>LCD_WRITEITEM_A1</code>	Write multiple bytes to LCD controller with A-line high.

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 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	Explanation
<code>LCD_CACHE</code>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).
<code>LCD_FIRSTCOM0</code>	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.
<code>LCD_FIRSTSEG0</code>	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.
<code>LCD_SUPPORT_CACHECONTROL</code>	When set to 0, the cache control functions of <code>LCD_L0_ControlCache()</code> driver API are disabled.

Special requirements for certain LCD controllers

None.

24.3.9 LCDPage4bpp driver

Supported hardware

Controllers

This driver has been tested with the following LCD controllers:

- Sitronix ST7528

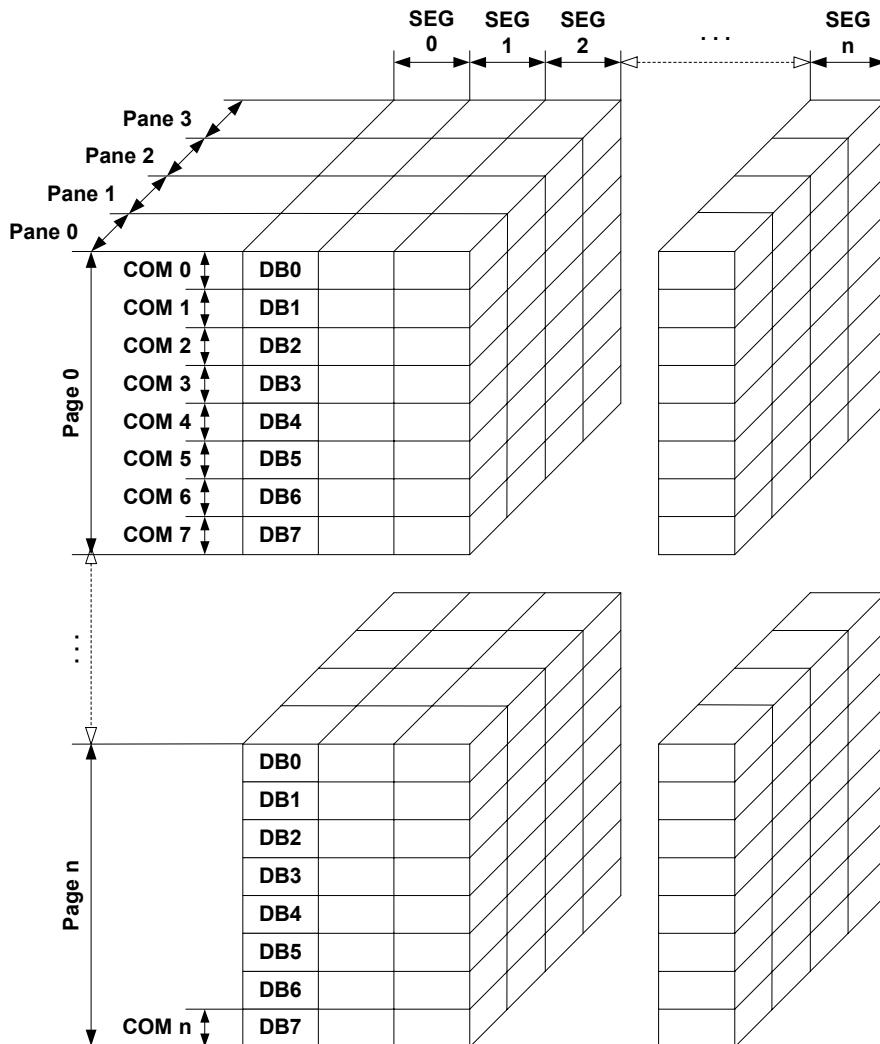
Bits per pixel

Supported color depth is 4 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) and 4 pin SPI interface.

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.

Additional RAM requirements of the driver

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.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface, a 4 pin SPI interface or a I2C bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the LCD controller.
LCD_READ_A1	Read a byte from LCD controller with A-line high. (Used only if working without cache)
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.

Display orientation

A Sitronix ST7528 display controller 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	Explanation
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_SEGO	Defines the number of available segment outputs of the display controller. Possible values for Sitronix ST7528 are: 132 (default, mode 0) 160 (mode 1)

24.3.10 LCDSDLin driver

Supported hardware

Controllers

The following table lists the supported controllers and their assigned numbers for LCD_CONTROLLER, as well as the level of support:

#	LCD controller	Add. info
6963	Toshiba T6963	1 BPP LCD controller with external memory. Graphic mode, fully supported
6901	Epson SED1330, SED1335, S1D13700	1 BPP LCD controller with external memory. Graphic mode, fully supported Access to the controller is slow, please refer to section "Add info about certain display controllers" for details.

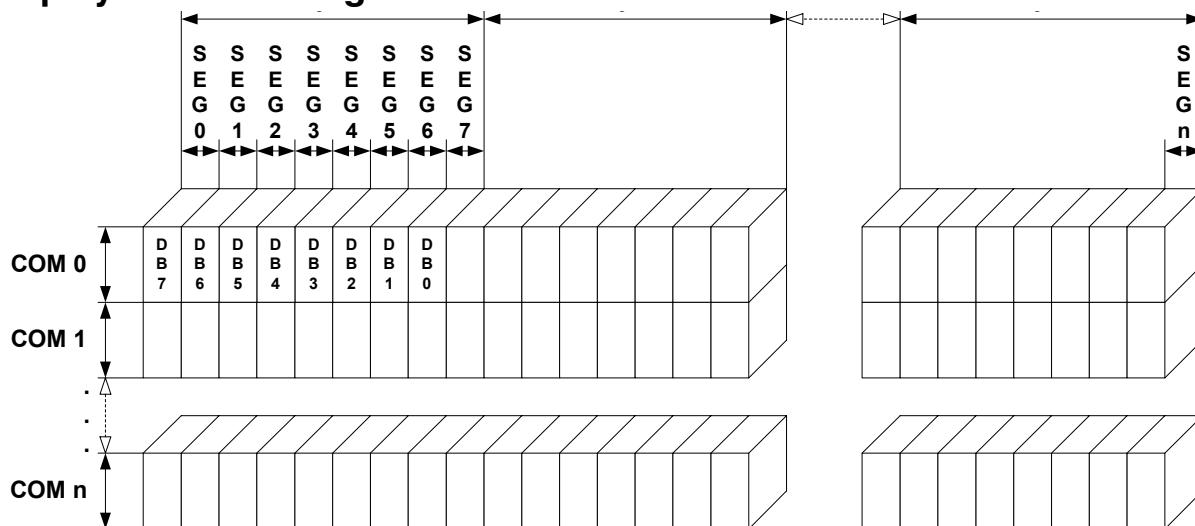
Bits per pixel

Supported color depth is 1 bpp.

Interfaces

The driver supports the 8-bit parallel (simple bus) interface.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM-requirement

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD 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 LCD-access. The amount of memory used by the cache may be calculated as follows:

$$\text{Size of RAM (in bytes)} = (\text{LCD_XSIZE} + 7) / 8 * \text{LCD_YSIZE}$$

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the LCD controller.
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.

Additional configuration switches for SED 1330, SED 1335

The following table shows optional configuration switches available for this driver:

Macro	Explanation
LCD_EXTENDED_WAIT	If set to 1 (default value) the driver ensures that no distortion will be shown when the driver writes to the LCD controller.

LCD_EXTENDED_WAIT

There is only a short time window to write to the LCD controller if you want to make sure, no distortion is visible on the LCD display when the driver writes to the LCD controller. If the configuration macro is set to 1 (default value) it makes sure that no distortion will be visible.

If you disable this behavior the driver performance will be accelerated. But when writing to the display small distortions will be visible. To disable the LCD_EXTENDED_WAIT macro add the following line to your LCDConf.h:

```
#define LCD_EXTENDED_WAIT 0
```

Additional info for SED 1330, SED 1335

These controllers are based on an old design. This design allows writing into display memory only in the horizontal & vertical non-display periods; otherwise the display will be momentarily distorted. Unfortunately, in order to make sure the driver only writes during the non-display periods, it has to wait for the beginning of the non display period and can then only write one byte. This slows the process of writing into display memory down considerably, even on very fast CPUs. This is not a limitation of emWin or the driver, but a limitation of the controller. If you need higher speed, you can use the LCD_EXTENDED_WAIT macro described above (and live with the distortions) or select a different LCD controller (Which we recommend if you are in an early stage of your design).

24.3.11 LCDVesa driver

This driver supports any PC-compatible hardware compatible with the VESA BIOS Extension standard (VBE) V1.2. It uses the INT 10h ROM BIOS functions to provide the VBE services. The driver has been written and tested with the Open Watcom compiler V1.0 but it should also work with other compilers.

Supported hardware

Any PC-compatible hardware as described above.

Supported resolutions

The following resolutions are supported by the driver:

- 320 x 200
- 640 x 480
- 800 x 600

Bits per pixel

Supported color depths are 8 and 16 bpp.

Display data RAM organization

Display data is organized in banks of up to 64Kb as described by the VESA standard.

RAM requirements of the driver

About 50 bytes for some static variables.

Additional driver functions

None.

Hardware configuration

This driver uses the INT 10h ROM BIOS functions to provide the VBE services. Therefore this driver does not need any additional hardware configuration.

Color format (R/B swap)

Red and blue needs to be swapped in the low level configuration. This can be done by setting the config switch `LCD_SWAP_RB` to 1 in the configuration file.

Example:

`LCDConf.h` for a VGA display 640x480, 16bpp:

```
#define LCD_XSIZE          640 /* X-resolution of LCD, Logical coor. */
#define LCD_YSIZE          480 /* Y-resolution of LCD, Logical coor. */
#define LCD_BITSPERPIXEL    16
#define LCD_CONTROLLER      8600
#define LCD_SWAP_RB         1
```

Additional configuration switches

None.

Special requirements

None

24.3.12 LCDXylon driver

Supported hardware

Controllers

The display driver has been written to support a FPGA-based display controller from Xylon.

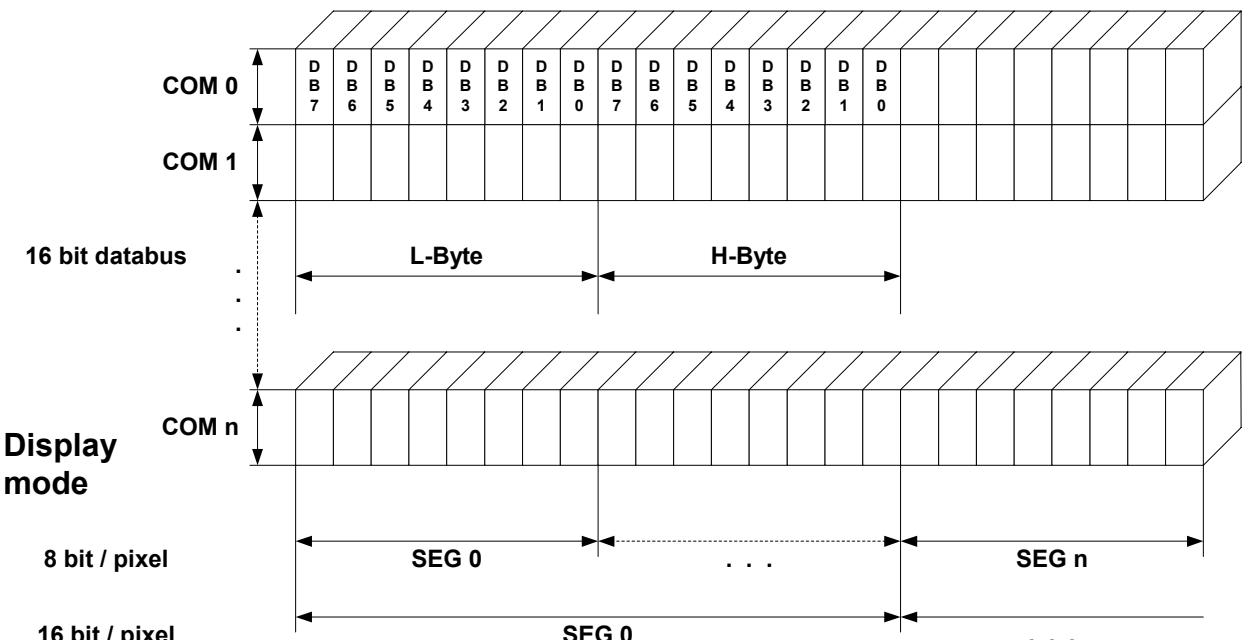
Bits per pixel

Supported color depth is 8 and 16 bpp.

Interfaces

16-bit parallel (full bus) mode.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD in terms of the color depth.

Additional RAM requirements of the driver

None.

Additional driver functions

None.

Hardware configuration

This driver requires a full bus interface for hardware access as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_MEM</code>	Read the contents of video memory of controller.
<code>LCD_READ_REG</code>	Read the contents of a configuration register of controller.
<code>LCD_WRITE_MEM</code>	Write to video memory (display data RAM) of controller.
<code>LCD_WRITE_REG</code>	Write to a configuration register of controller.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
LCD_USE_BITBLT	If set to 0, it disables the BitBLT engine. If set to 1 (the default value), the driver will use all available hardware acceleration.
LCD_USE_BITBLT_1BPP	If set to 0, it disables the BitBLT engine for rendering 1bpp bitmaps. If set to 1 (the default value), the driver will use the hardware acceleration.
LCD_USE_BITBLT_FILL	If set to 0, it disables the BitBLT engine for filling rectangles. If set to 1 (the default value), the driver will use the hardware acceleration.

24.3.13 LCD0323 driver

Supported hardware

Controllers

This driver has been tested with the following display controller:

- Solomon SSD0323 OLED controller

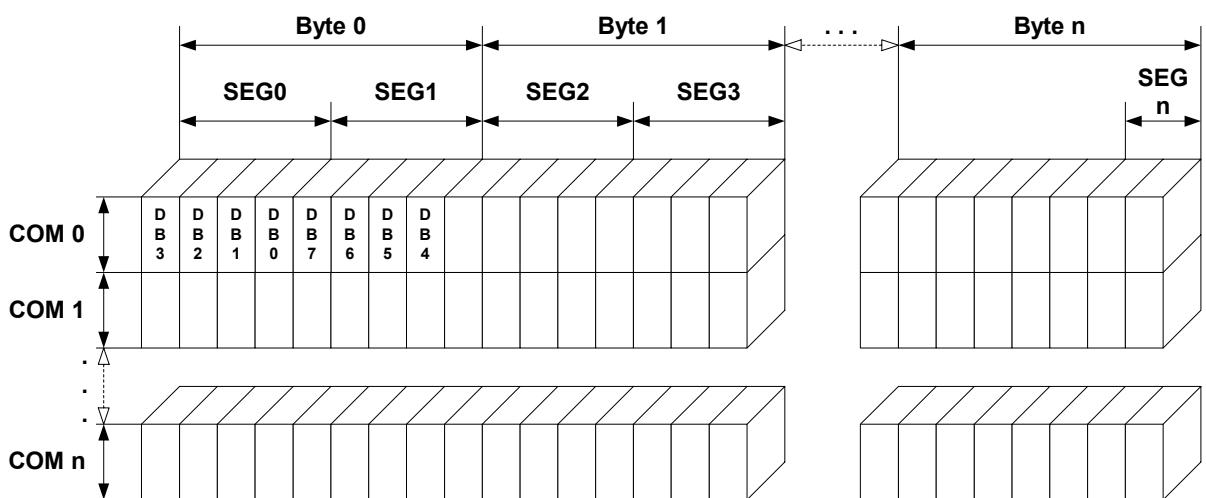
Bits per pixel

Supported color depth is 4bpp.

Interfaces

8-bit parallel (simple bus) and serial (SPI4) bus interfaces are supported.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the display.

Additional RAM requirements of the driver

This driver can be used with or without a display data 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 RAM access. Not using a cache degrades the performance of this driver. The amount of memory used by the cache is 4096 bytes.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_A1</code>	Read a byte from LCD controller with A-line high.
<code>LCD_WRITE_A0</code>	Write a byte to LCD controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to LCD controller with A-line high.
<code>LCD_WRITEITEM_A1</code>	Write multiple bytes to LCD controller with A-line high.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
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).

Special requirements for certain LCD controllers

None.

24.3.14 LCD07X1 driver

Supported hardware

Controllers

This driver has been tested with the following LCD controllers:

- Samsung KS0711
- Samsung KS0741
- Solomon SSD1854

Bits per pixel

Supported color depth is 2 bpp.

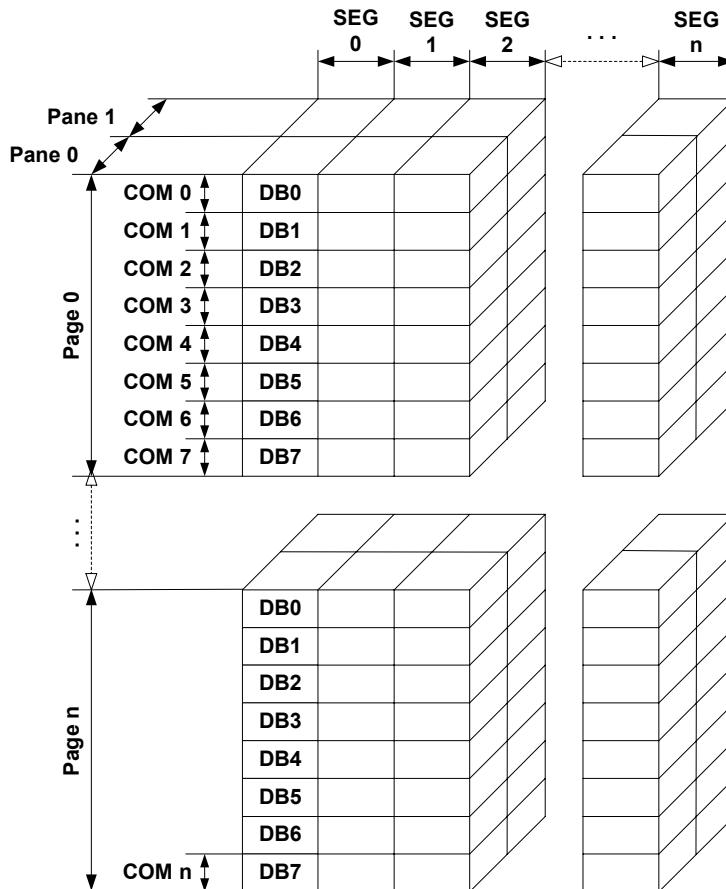
Interfaces

The chip supports three types of interfaces:

- 8-bit parallel (simple bus) interface
- 4-pin serial peripheral interface, or SPI.
- 3-pin SPI.

The current version of the driver supports the 8-bit parallel (simple bus) or 4-pin SPI interface.

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 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.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD 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 LCD-access. The amount of memory used by the cache may be calculated as follows:
 $\text{Size of RAM (in bytes)} = (\text{LCD_YSIZE} + 7) / 8 * \text{LCD_XSIZE} * 2$

Additional driver functions

LCD_L0_ControlCache

For information about this function, please refer to Chapter 28: "LCD Driver API".

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_A0</code>	Read a byte from LCD controller with A-line low. (Used only if working without cache)
<code>LCD_READ_A1</code>	Read a byte from LCD controller with A-line high. (Used only if working without cache)
<code>LCD_WRITE_A0</code>	Write a byte to LCD controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to LCD controller with A-line high.
<code>LCD_Writem_A1</code>	Write multiple bytes to LCD controller with A-line high.

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	Explanation
<code>LCD_FIRSTCOM0</code>	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.
<code>LCD_FIRSTSEG0</code>	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.

Special requirements for certain LCD controllers

None.

24.3.15 LCD1200 driver

Supported hardware

Controllers

This driver has been tested with the following display controllers:

- Toppoly C0C0
- Toppoly C0E0

Bits per pixel

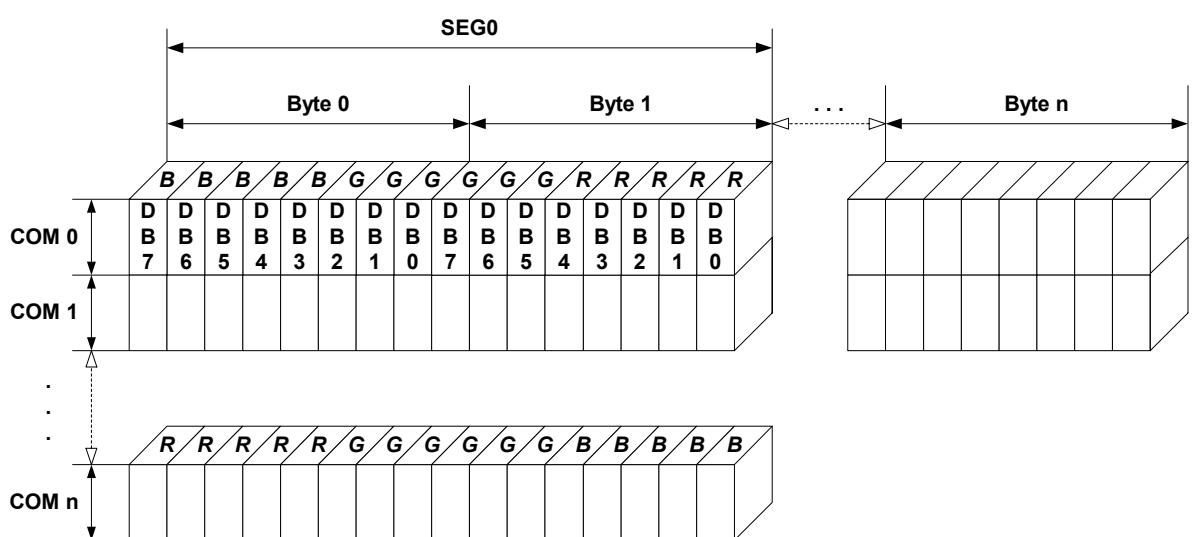
Supported color depth is 16 bpp.

Interfaces

The driver supports 16-bit parallel (simple bus) interface.

Display data RAM organization

16 bits per pixel, fixed palette = 565



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

The driver can be used with a write buffer used 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 executes only one time the macro LCD_WRITEITEM_A1 to transfer the data to the display controller. The default buffer size is 500 words.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface or with a 3 pin SPI interface. The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the display controller.
<code>LCD_WRITE_BUFFER_SIZE</code>	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 contents of the buffer with one execution of the macro <code>LCD_WRITEITEM_A1</code> instead of multiple executions. The default buffer size is 500 words.
<code>LCD_READ_A1</code>	Reads a word from display controller with RS-line high.
<code>LCD_WRITEITEM_A1</code>	Writes multiple words to display controller with RS-line high.
<code>LCD_WRITE_A0</code>	Writes multiple words to display controller with RS-line low.
<code>LCD_WRITE_A1</code>	Writes multiple words to display controller with RS-line high.

Additional configuration switches

None.

Special requirements

The driver needs to work in the fixed palette mode 565, which is the default value for 16 bit per pixel configurations. The driver does not work with other palettes. 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 file LCDConf.h:

```
#define LCD_SWAP_RB      1
```

24.3.16 LCD13701 driver

Supported hardware

Controllers

This driver has been tested with the following display controller:

- Epson S1D13701 OLED controller

Bits per pixel

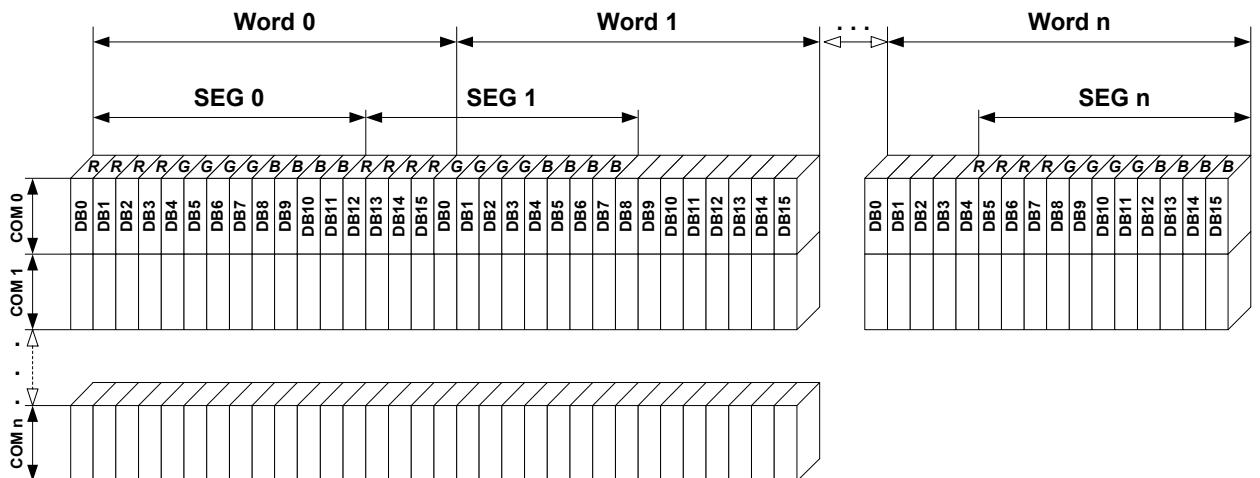
Supported color depth is 9 and 12 bpp.

Interfaces

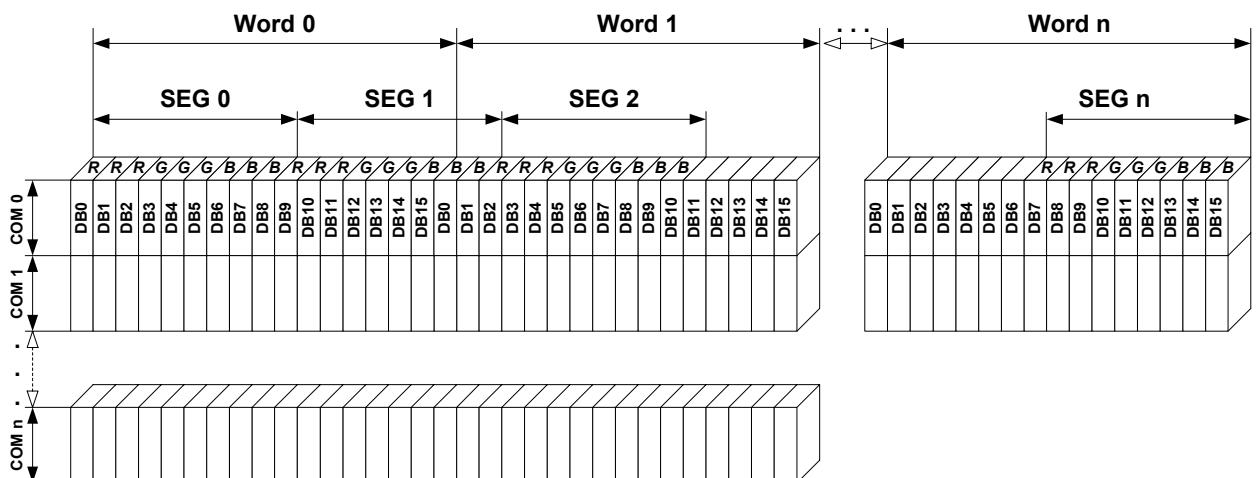
16-bit parallel (simple bus) bus interfaces is supported.

Display data RAM organization

12 bits per pixel, fixed palette = 444



9 bits per pixel, fixed palette = -1



The picture above shows the relation between the display memory and the SEG and COM lines of the display.

Additional RAM requirements of the driver

This driver can be used with or without a display data 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 RAM access. Not using a cache degrades the performance of this driver. The amount of memory used by the cache can be calculated as follows:

$$\text{Size of RAM (in bytes)} = (\text{LCD_XSIZE} * \text{LCD_BITSPERPIXEL} / 16) * \text{LCD_YSIZE} * 2$$

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the display controller.
LCD_READ_A1	Read a word from display controller with A-line high.
LCD_WRITE_A0	Write a word to display controller with A-line low.
LCD_WRITE_A1	Write a word to display controller with A-line high.
LCD_WRAITEM_A1	Write multiple words to display controller with A-line high.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
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).

Special requirements for certain LCD controllers

None.

24.3.17 LCD159A driver

Supported hardware

Controllers

This driver has been tested with the following display controllers:

- Epson SED159A

It should be assumed that it will also work with any controller of similar organization.

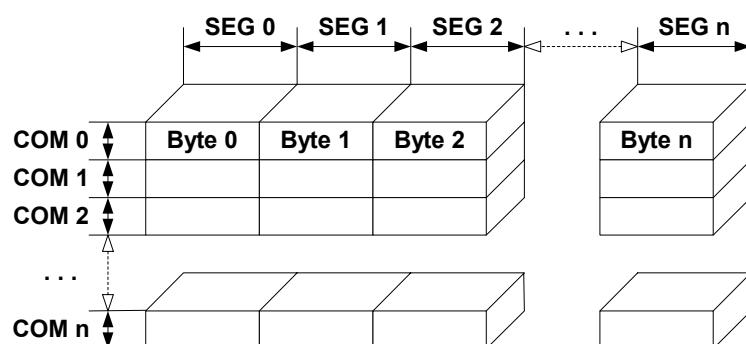
Bits per pixel

Supported color depth is 8 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) interface.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

None.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the LCD controller.
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.

Additional configuration switches

None.

Special requirements for certain LCD controllers

None.

24.3.18 LCD15E05 driver

Supported hardware

Controllers

This driver has been tested with the following display controllers:

- Epson S1D15E05

It should be assumed that it will also work with any controller of similar organization.

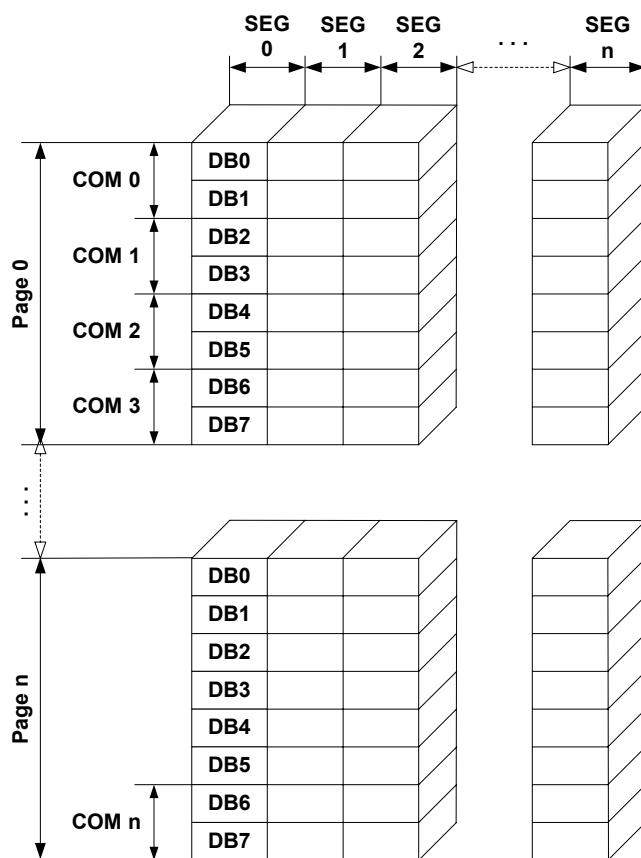
Bits per pixel

Supported color depth is 2 bpp.

Interfaces

The driver supports the 8-bit parallel (simple bus) and 4 pin SPI interfaces.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD 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 LCD-access. The amount of memory used by the cache may be calculated as follows:

$$\text{Size of RAM (in bytes)} = (\text{LCD_YSIZE} + 7) / 8 * \text{LCD_XSIZE}$$

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_A0</code>	Read a byte from LCD controller with A-line low.
<code>LCD_READ_A1</code>	Read a byte from LCD controller with A-line high.
<code>LCD_WRITE_A0</code>	Write a byte to LCD controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to LCD controller with A-line high.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<code>LCD_CACHE</code>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

Special requirements for certain LCD controllers

None.

24.3.19 LCD1611 driver

Supported hardware

Controllers

This driver has been tested with the following display controllers:

- UltraChip UC1610, UltraChip UC1611

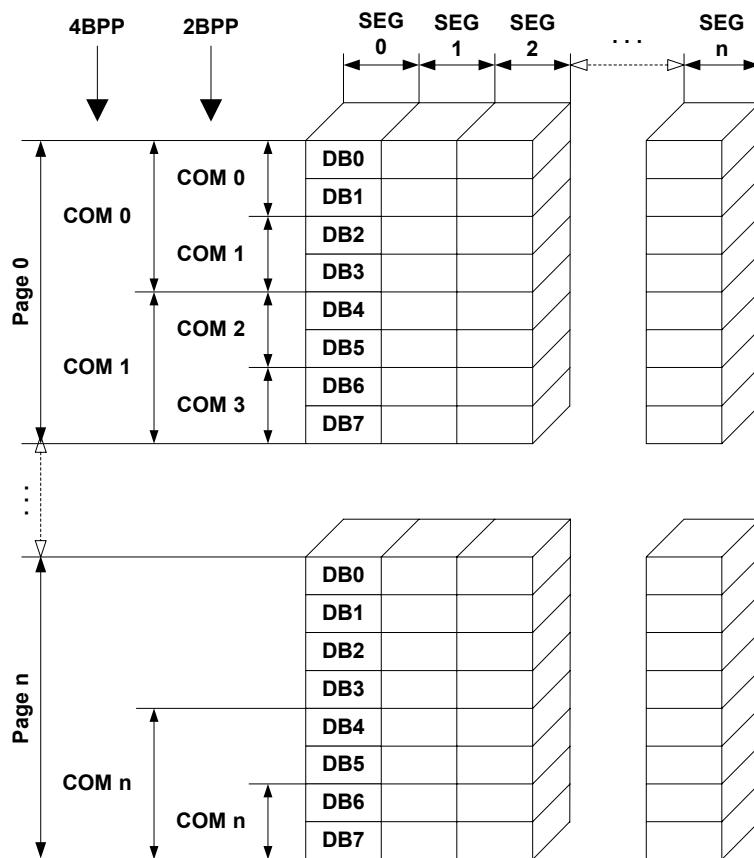
Bits per pixel

Supported color depth is 2bpp (UC1610) and 4bpp (UC1611).

Interfaces

8-bit parallel (simple bus), serial (SPI4) and I2C bus interfaces are supported.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver can be used with or without a display data cache. The data cache contains a complete copy of the contents of the LCD data RAM. If a cache is not 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) =

$$(LCD_YSIZE + (8 / LCD_BITSPERPIXEL - 1)) / 8 * LCD_BITSPERPIXEL * LCD_XSIZE$$

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_A0</code>	Read a byte from LCD controller with A-line low.
<code>LCD_READ_A1</code>	Read a byte from LCD controller with A-line high.
<code>LCD_WRITE_A0</code>	Write a byte to LCD controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to LCD controller with A-line high.
<code>LCD_WRITEITEM_A1</code>	Write multiple bytes to LCD controller with A-line high.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<code>LCD_CACHE</code>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

Special requirements for certain LCD controllers

None.

24.3.20 LCD161620 driver

Supported hardware

Controllers

This driver has been tested with the following display controller:

- NEC µPD161620

Bits per pixel

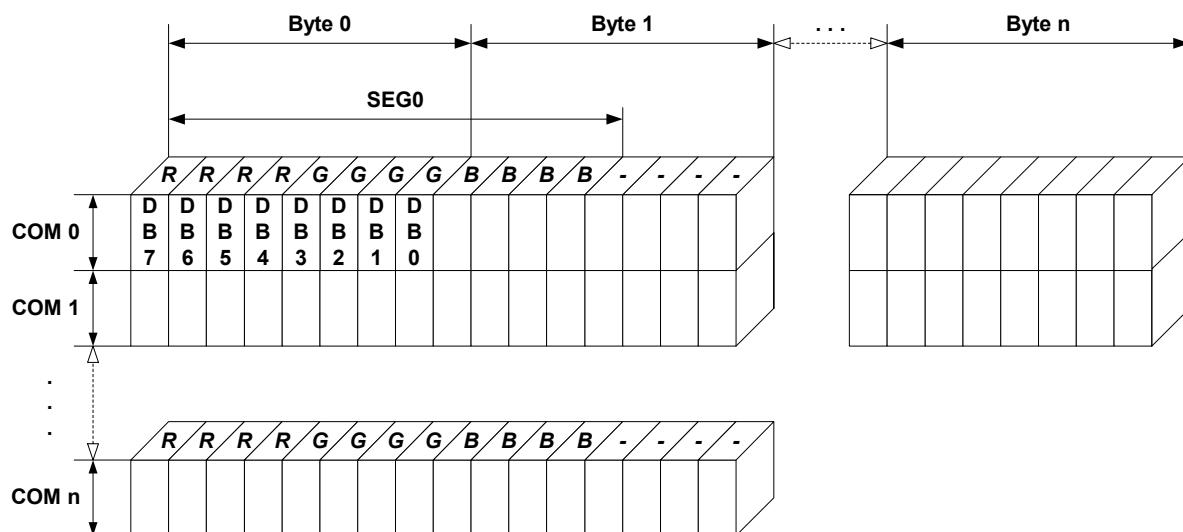
Supported color depth is 12 bpp.

Interfaces

The chip supports 8-bit parallel (simple bus) interface.

Display data RAM organization

12 bits per pixel, fixed palette = 444121



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD. The driver works in '1-pixel/2-byte'-mode.

Additional RAM requirements of the driver

None.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following tables lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the display controller.
LCD_READ_A0	Read a byte from display controller with A-line low.
LCD_READ_A1	Read a byte from display controller with A-line high.
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.

Additional configuration switches

None.

Special requirements

The driver needs to work in the fixed palette mode 444121. The driver does not work with other palettes.

You should use the following macro definition in the file LCDConf.h:

```
#define LCD_FIXEDPALETTE 444121
```

24.3.21 LCD1781 driver

Supported hardware

Controllers

This driver has been tested with the following display controller:

- Solomon SSD1781

Bits per pixel

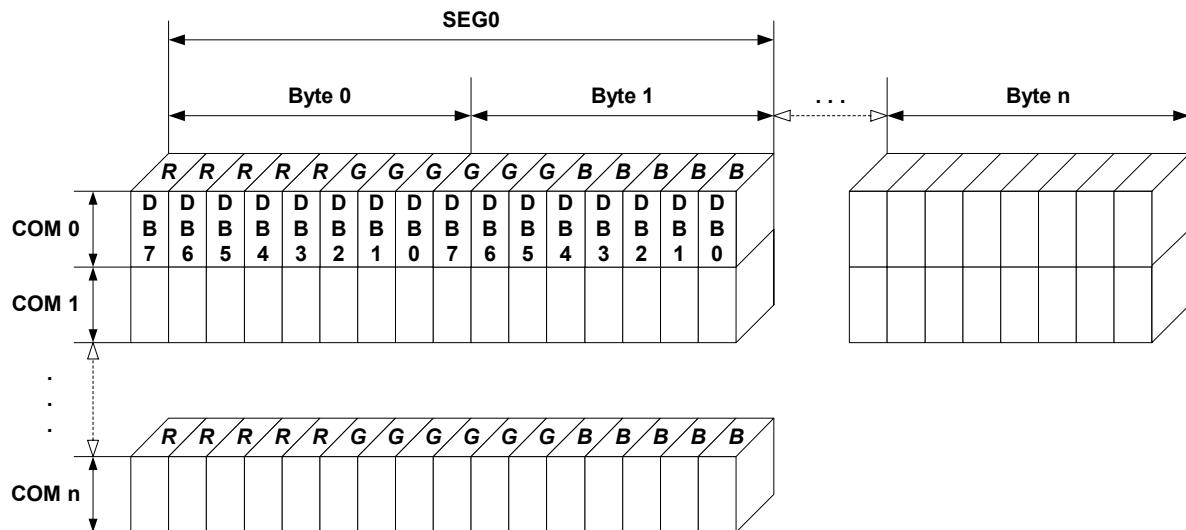
Supported color depth is 16 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) or 4 pin SPI interface.

Display data RAM organization

16 bits per pixel, fixed palette = 565



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements.

The amount of memory used by the cache is: `LCD_XSIZE x LCD_YSIZE x 2 bytes`.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface or a 4 pin SPI interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the display controller.
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_WRITEITEM_A1	Write multiple bytes to display controller with A-line high.

Display orientation

The controller SSD1781 supports rotating and mirroring. So it is recommended to use the hardware functions of the controller if rotating and/or mirroring is needed instead of the display orientation macros LCD_MIRROR_... and LCD_SWAP_XY of emWin. The configuration sample folder contains a sample which uses the hardware to rotate the output 180 degrees. The command 0xBC of the controller can be used to mirror and/or rotate the display output. For more details about how to change the display orientation via the hardware functions please refer to the manual of the SSD1781.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
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_GET_BUSY	The SSD1781 controller has a separate pin which should be used to query the busy state. This macro should return 1 if the busy line of the SSD1781 is high and 0 if the line is low. If the macro is defined it is used at the beginning of each low level driver function to wait until the driver is not busy. Example: <code>#define LCD_GET_BUSY() (P6 & 0x80)</code>
LCD_WAIT	If the busy line is not available the macro LCD_WAIT can be used to call a function which waits a while in dependence of the number of drawn pixels. If defined the LCD_WAIT macro is executed after each high level drawing function call of the SSD1781. Example: <code>#define LCD_WAIT(NumPixels) OS_Delay((NumPixels + 900) / 1000)</code>

Special requirements

The driver needs to work in the fixed palette mode 565, which is the default value for 16 bit per pixel configurations. The driver does not work with other palettes. 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 file LCDConf.h:

```
#define LCD_SWAP_RB           1
```

24.3.22 LCD501 driver

Supported hardware

Controllers

This driver has been tested with the following OLED controller:

- Leadis LDS501

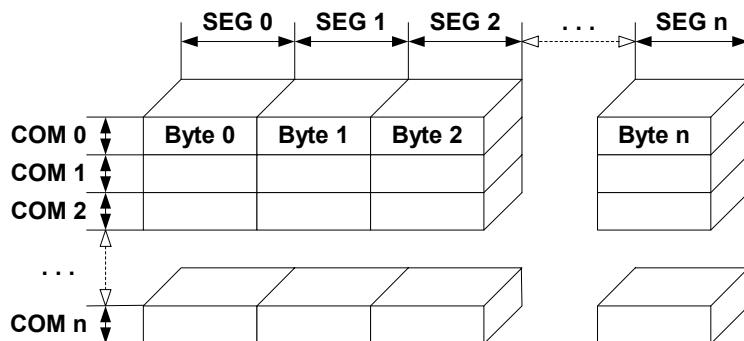
Bits per pixel

Supported color depth is 8 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) interfaces.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

This LCD driver can only be used with 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: $96 \times 64 = 6144$ bytes. The driver can not be used without data cache.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the display controller.
<code>LCD_WRITE_A0</code>	Write a byte to display controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to display controller with A-line high.

Additional configuration switches

None.

Special requirements

The driver needs to work in the fixed palette mode 233. The driver does not work with other palettes. 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 file LCDConf.h:

```
#define LCD_FIXEDPALETTE 233
#define LCD_SWAP_RB      1
```

24.3.23 LCD6331 driver

Supported hardware

Controllers

This driver has been tested with the following display controller:

- Samsung S6B33B1X

Bits per pixel

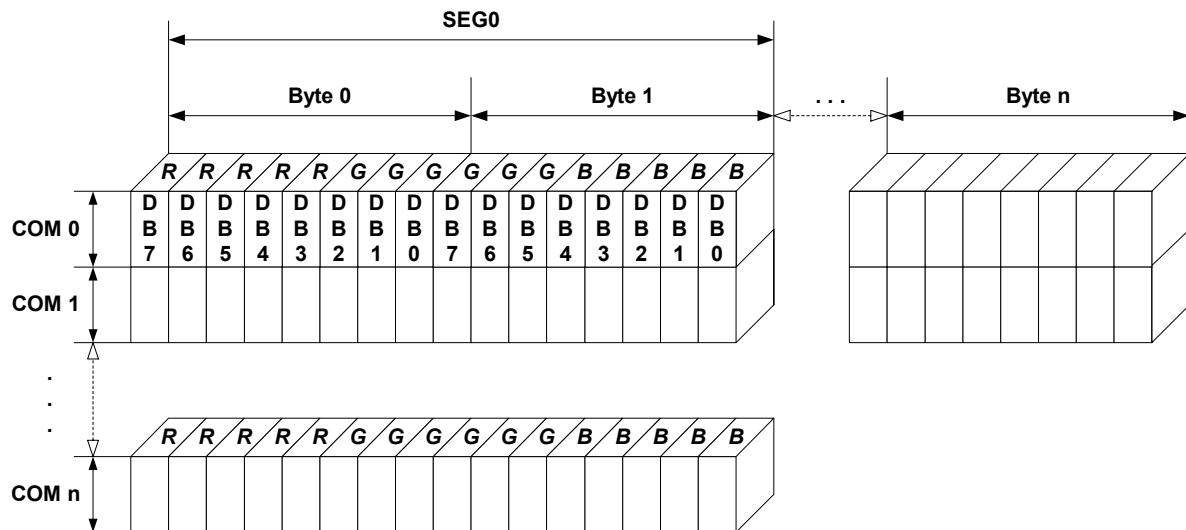
Supported color depth is 16 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) and 4 pin SPI interface.

Display data RAM organization

16 bits per pixel, fixed palette = 565



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

This LCD driver can only be used with 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: $\text{LCD_XSIZE} \times \text{LCD_YSIZE} \times 2$ bytes. The driver can not be used without data cache.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface or with a 4 pin SPI interface. The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the display controller.
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_WRITEITEM_A1	Write multiple bytes to display controller with A-line high.

The driver initializes the 'Driver Output Mode' and 'Entry Mode' itself. The user does not need to initialize this registers in the LCD_INIT_CONTROLLER macro.

Additional configuration switches

None.

Special requirements

The driver needs to work in the fixed palette mode 565. The driver does not work with other palettes. 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 file LCDConf.h:

```
#define LCD_FIXEDPALETTE 565
#define LCD_SWAP_RB 1
```

24.3.24 LCD6642X driver

Supported hardware

Controllers

This driver has been tested with the following display controllers:

- Hitachi HD66420
- Hitachi HD66421

It should be assumed that it will also work with any controller of similar organization.

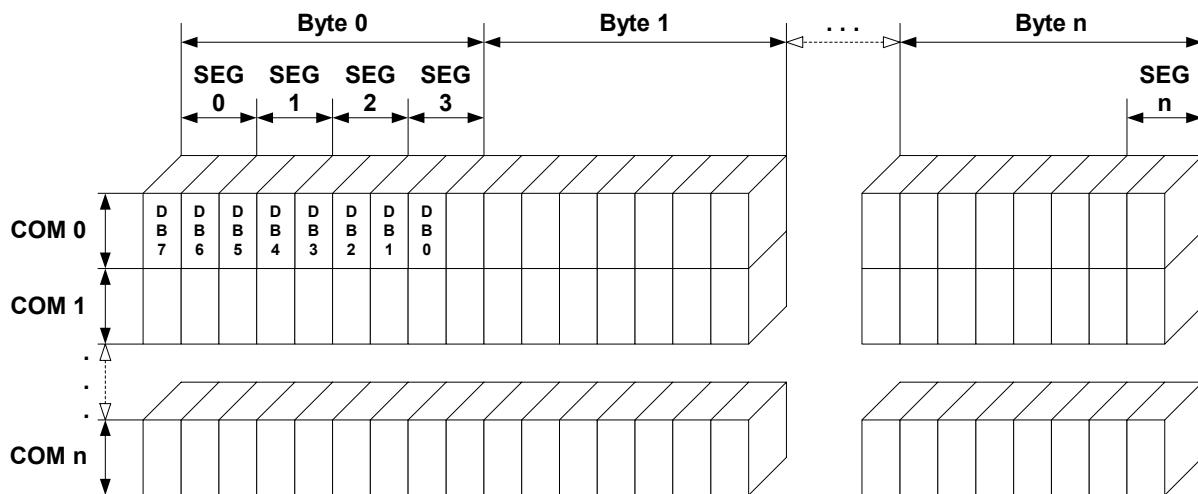
Bits per pixel

Supported color depth is 2 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) interfaces.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not 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:

$$\text{Size of RAM (in bytes)} = (\text{LCD_XSIZE} + 7) / 8 * \text{LCD_YSIZE} * 2$$

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the display controller.
LCD_READ_A0	Read a byte from display controller with A-line low.

Macro	Explanation
LCD_READ_A1	Read a byte from display controller with A-line high.
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.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
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).

Special requirements

None.

24.3.25 LCD667XX driver

Supported hardware

Controllers

This driver has been tested with the following LCD controllers:

- Hitachi HD66766
- Hitachi HD66772

Bits per pixel

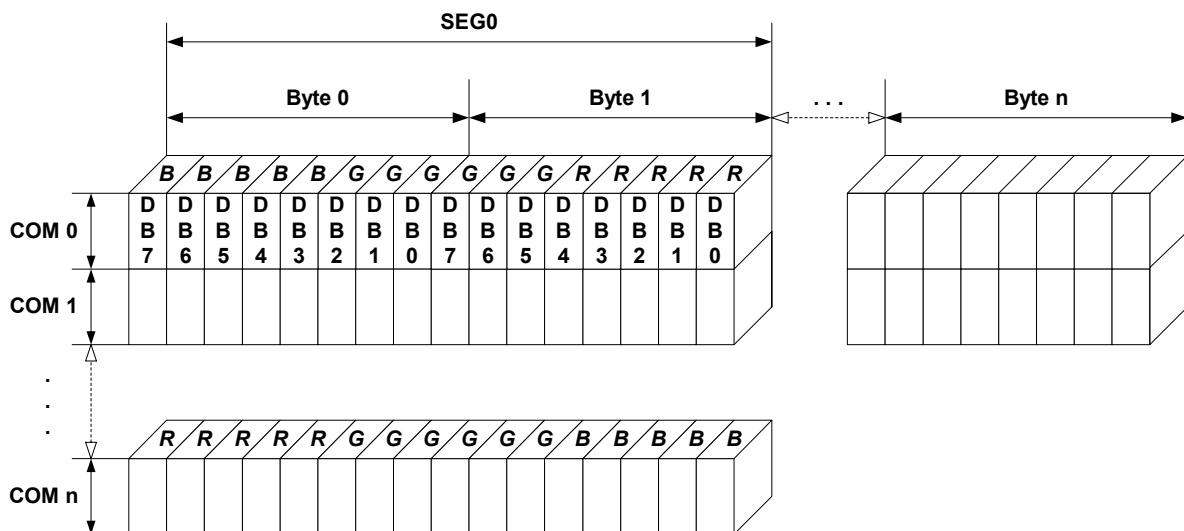
Supported color depth is 16 bpp.

Interfaces

The driver supports 8-bit parallel, 16 bit parallel and 3 pin SPI interface. Default mode is 8-bit parallel.

Display data RAM organization

16 bits per pixel, fixed palette = 565



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

This LCD 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 only recommended 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.

The driver can be used with a write buffer used 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 executes only one time the macro LCD_WRITEITEM_A1 to transfer the data to the display controller. The default buffer size is 500 bytes.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface or with a 3 pin SPI interface. The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the display controller.
<code>LCD_NUM_DUMMY_READS</code>	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.
<code>LCD_SERIAL_ID</code>	With a serial interface this macro defines the ID signal of the device ID code. It should be 0 (default) or 1.
<code>LCD_USE_SERIAL_3PIN</code>	Should be set to 1 if the serial interface is used. Default is 0.
<code>LCD_USE_PARALLEL_16</code>	Should be set to 1 if the 16 bit parallel interface is used. Default is 0.
<code>LCD_WRITE_BUFFER_SIZE</code>	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 contents of the buffer with one execution of the macro <code>LCD_WRITEITEM_A1</code> , instead of multiple macro executions. The default buffer size is 500 bytes.
<code>LCD_READM_A1</code>	Read multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) from display controller with RS-line high.
<code>LCD_WRITEITEM_A1</code>	Write multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) to display controller with RS-line high.
<code>LCD_WRITEITEM_A0</code>	Write multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) to display controller with RS-line low.

The driver initializes the 'Driver Output Mode' and 'Entry Mode' register itself. The user does not need to initialize this registers in the `LCD_INIT_CONTROLLER` macro.

Additional configuration switches

None.

Special requirements

None.

24.3.26 LCD66750 driver

Supported hardware

Controllers

This driver has been tested with the following LCD controllers:

- Hitachi HD66750
- Hitachi HD66753

It should be assumed that it will also work with any controller of similar organization.

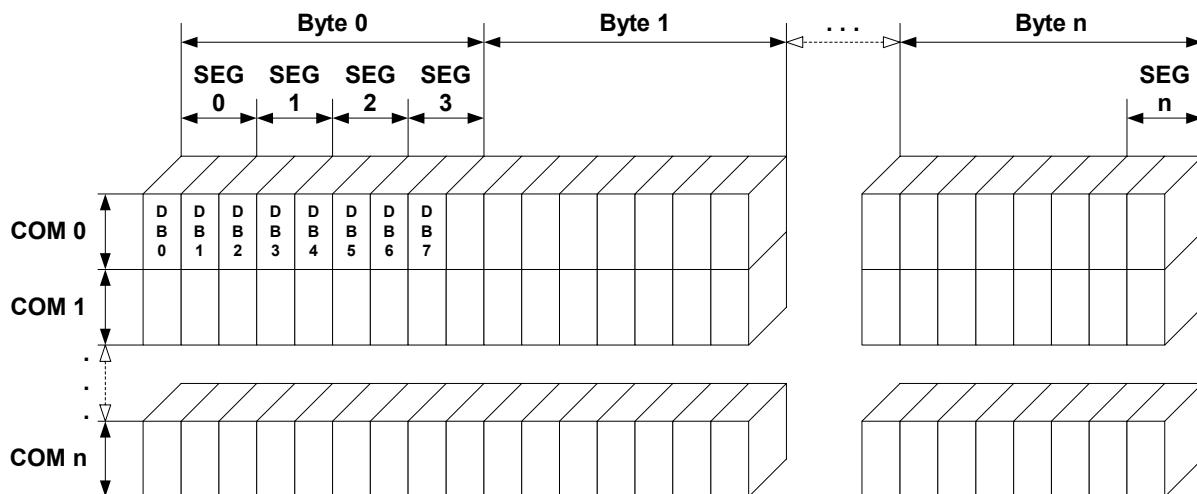
Bits per pixel

Supported color depth is 2 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) interfaces.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not 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 is 4096 bytes if HD66750 is used and 5544 bytes if HD66753 is used.

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 27: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
LCD_INIT_CONTROLLER	Initialization sequence for the LCD controller.
LCD_READ_A0	Read a byte from LCD controller with A-line low.

Macro	Explanation
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.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
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).

Special requirements for certain LCD controllers

None.

24.3.27 LCDTemplate driver

This driver is part of the basic package and can be easily adapted to each LCD controller. It contains the complete functionality needed for a LCD driver.

Adapting the template driver

To adapt the driver to a currently not supported LCD controller you only have to adapt the routines LCD_L0_SetPixelIndex and LCD_L0_GetPixelIndex. The upper layers calling this routines makes sure that the given coordinates are in range, so that no check on the parameters needs to be performed.

In a second step, a new driver should be modified to use its own number for activation (LCD_CONTROLLER), and optionally be optimized to improve drawing speed.

24.3.28 LCDNull driver

This driver is part of the basic package and can be used for measurement purpose. It contains all API functions of a LCD driver without any function.

Using this driver

Since the driver contains only 'empty' API functions it makes it possible to measure the time difference used for some GUI-operations between using the real hardware driver and this empty driver. The time difference is the time used for the LCD display operation.

24.4 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 program 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.

24.4.1 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_L0: Driver routines

Routine	Explanation
Init & display control group	
<code>LCD_L0_Init()</code>	Initialize the display.
<code>LCD_L0_ReInit()</code>	Reinitialize LCD without erasing the contents.
<code>LCD_L0_Off()</code>	Switch LCD off.
<code>LCD_L0_On()</code>	Switch LCD on.
Drawing group	
<code>LCD_L0_DrawBitmap()</code>	Universal draw bitmap routine.
<code>LCD_L0_DrawHLine()</code>	Draw a horizontal line.
<code>LCD_L0_DrawPixel()</code>	Draw a pixel in the current foreground color.
<code>LCD_L0_DrawVLine()</code>	Draw a vertical line.
<code>LCD_L0_FillRect()</code>	Fill a rectangular area.
<code>LCD_L0_SetPixelIndex()</code>	Draw a pixel in a specified color.
<code>LCD_L0_XorPixel()</code>	Invert a pixel.
"Get" group	
<code>LCD_L0_GetPixelIndex()</code>	Returs the index of the color of a specific pixel.
"Set" group	
<code>LCD_L0_SetOrg()</code>	Not yet used, reserved for future use (must exist in driver).
Lookup table group	
<code>LCD_L0_SetLUTEntry()</code>	Modifiy a single entry of LUT.
Misc. group (optional)	
<code>LCD_L0_ControlCache()</code>	Lock/unlock/flush LCD cache.

LCD: LCD layer routines

Routine	Explanation
<code>LCD_GetBitsPerPixel()</code>	Return the number of bits per pixel.
<code>LCD_GetBitsPerPixelEx()</code>	Returns the number of bits per pixel of given layer/display.
<code>LCD_GetFixedPalette()</code>	Return the fixed palette mode.
<code>LCD_GetFixedPaletteEx()</code>	Returns the fixed palette mode of given layer/display.
<code>LCD_GetNumColors()</code>	Return the number of available colors.
<code>LCD_GetNumColorsEx()</code>	Returns the number of available colors of given layer/display.
<code>LCD_GetVXSize()</code>	Return virtual X-size of LCD in pixels.
<code>LCD_GetVXSizeEx()</code>	Returns virtual X-size of given layer/display in pixels.
<code>LCD_GetVYSize()</code>	Return virtual Y-size of LCD in pixels.
<code>LCD_GetVYSizeEx()</code>	Returns virtual Y-size of given layer/display in pixels.
<code>LCD_GetXMag()</code>	Returns the magnification factor in x.
<code>LCD_GetXMagEx()</code>	Returns the magnification factor of given layer/display in x.
<code>LCD_GetXSize()</code>	Return physical X-size of LCD in pixels.
<code>LCD_GetXSizeEx()</code>	Returns physical X-size of given layer/display in pixels.
<code>LCD_GetYMag()</code>	Returns the magnification factor in y.
<code>LCD_GetYMagEx()</code>	Returns the magnification factor of given layer/display in y.
<code>LCD_GetYSize()</code>	Return physical Y-size of LCD in pixels.
<code>LCD_GetYSizeEx()</code>	Returns physical Y-size of given layer/display in pixels.

24.4.2 Init & display control group

`LCD_L0_Init()`

Description

Initializes the LCD using the configuration settings in `LCDConf.h`. This routine is called automatically by `GUI_Init()` if the upper GUI layer is used and therefore should not need to be called manually.

Prototype

```
void LCD_L0_Init (void);
```

LCD_L0_ReInit()

Description

Reinitializes the LCD using the configuration settings, without erasing the display contents.

Prototype

```
void LCD_L0_ReInit (LCD_INITINFO* pInitInfo);
```

LCD_L0_Off(), LCD_L0_On()

Description

Switch the display off or on, respectively.

Prototypes

```
void LCD_L0_Off(void);
void LCD_L0_On(void);
```

Add. information

Use of these routines does not affect the contents of the video memory or other settings. You may therefore safely switch off the display and switch it back on without having to refresh the contents.

24.4.3 Drawing group

LCD_L0_DrawBitmap()

Description

Draws a pre-converted bitmap.

Prototype

```
LCD_L0_DrawBitMap(int x0, int y0,
                  int Xsize, int Ysize,
                  int BitsPerPixel,
                  int BytesPerLine,
                  const U8* pData, int Diff,
                  const LCD_PIXELINDEX* pTrans);
```

Parameter	Meaning
x0	Upper left X-position of bitmap to draw.
y0	Upper left Y-position of bitmap to draw.
Xsize	Number of pixels in horizontal direction.
Ysize	Number of pixels in vertical direction.
BitsPerPixel	Number of bits per pixel.
BytesPerLine	Number of bytes per line of the image.
pData	Pointer to the actual image, the data that defines what the bitmap looks like.
Diff	Number of pixels to skip from the left side.

LCD_L0_DrawHLine()

Description

Draws a horizontal line one pixel thick, at a specified position using the current foreground color.

Prototype

```
void LCD_L0_DrawHLine(int x0, int y, int x1);
```

Parameter	Meaning
x0	Start position of line.
y	Y-position of line to draw.
x1	End position of line.

Add. information

With most LCD controllers, this routine executes 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 `DrawLine` routine.

LCD_L0_DrawPixel()

Description

Draws one pixel at a specified position using the current foreground color.

Prototype

```
void LCD_L0_DrawPixel(int x, int y);
```

Parameter	Meaning
x	X-position of pixel to draw.
y	Y-position of pixel to draw.

LCD_L0_DrawVLine()

Description

Draws a vertical line one pixel thick, at a specified position using the current foreground color.

Prototype

```
void LCD_L0_DrawVLine(int x , int y0, int y1);
```

Parameter	Meaning
x	X-position of line to draw.
y0	Start position of line.
y1	End position of line.

Add. information

With most LCD-controllers, this routine executes 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 `DrawLine` routine.

LCD_L0_FillRect()

Description

Draws a filled rectangle at a specified position using the current foreground color.

Prototype

```
void LCD_L0_FillRect(int x0, int y0, int x1, int y1);
```

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

LCD_L0_SetPixelIndex()**Description**

Draws one pixel using a specified color

Prototype

```
void LCD_L0_SetPixelIndex(int x, int y, int ColorIndex);
```

Parameter	Meaning
x	X-position of pixel to draw.
y	Y-position of pixel to draw.
ColorIndex	Color to be used.

LCD_L0_XorPixel()**Description**

Inverts one pixel.

Prototype

```
void LCD_L0_XorPixel(int x, int y);
```

Parameter	Meaning
x	X-position of pixel to invert.
y	Y-position of pixel to invert.

24.4.4 "Get" group**LCD_L0_GetPixelIndex()****Description**

Returns the RGB color index of a specified pixel.

Prototype

```
int LCD_L0_GetPixelIndex(int x, int y);
```

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

Return value

The index of the pixel.

Add. information

For further information see Chapter 11: "Colors".

24.4.5 Lookup table (LUT) group

LCD_L0_SetLUTEntry()

Description

Modifies a single entry to the LUT of the LCD controller(s).

Prototype

```
void LCD_L0_SetLUTEntry(U8 Pos, LCD_COLOR Color);
```

Parameter	Meaning
Pos	Position within the lookup table. Should be less than the number of colors, e.g. 0-3 for 2bpp, 0-15 for 4bpp, 0-255 for 8bpp.
Color	24-bit RGB value. The closest value possible will be used for the LUT. If a color LUT is to be initialized, all 3 components are used. In monochrome modes the green component is used, but it is still recommended (for better understanding of the program code) to set all 3 colors to the same value (such as 0x555555 or 0xa0a0a0).

24.4.6 Miscellaneous group

LCD_L0_ControlCache()

Description

Locks, unlocks or flushes the cache. This routine may be used to set the cache to a locked state, in which all drawing operations on the driver cause changes in the video memory's cache (in CPU RAM), but do not cause any visible output. Unlocking or flushing then causes those changes to be written to the display. This can help to avoid flickering of the display and also accelerate drawing. It does not matter how many different drawing operations are executed; the changes will all be written to the display at once. In order to be able to do this, `LCD_SUPPORT_CACHECONTROL` must be enabled in the configuration file.

Prototype

```
U8 LCD_ControlCache(U8 command);
```

Parameter	Meaning
command	Specify the command to be given to the cache. Use the symbolic values in the table below.

Permitted values for parameter <code>command</code>	
<code>LCD_CC_UNLOCK</code>	Set the default mode: cache is transparent.
<code>LCD_CC_LOCK</code>	Lock the cache, no write operations will be performed until cache is unlocked or flushed.
<code>LCD_CC_FLUSH</code>	Flush the cache, writing all modified data to the video RAM.

Return value

Information on the state of the cache. Ignore.

Add. information

When the cache is locked, the driver maintains a "hitlist" -- a list of bytes which have been modified and need to be written to the display. This hitlist uses 1 bit per byte of video memory.

This is an optional feature which is not supported by all LCD drivers

Example

The code in the following example performs drawing operations on the display which overlap. In order to accelerate the update of the display and to avoid flickering, the cache is locked before and unlocked after these operations.

```
LCD_ControlCache(LCD_CC_LOCK);

GUI_FillCircle(30,30,20);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(50,30,10);
GUI_SetTextMode(GUI_TEXTMODE_XOR);
GUI_DispStringAt("Hello world\n",0,0);
GUI_DrawHLine(16, 5,25);
GUI_DrawHLine(18, 5,25);
GUI_DispStringAt("XOR Text",0,20);
GUI_DispStringAt("XOR Text",0,60);

LCD_ControlCache(LCD_CC_UNLOCK);
```

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	Meaning
Index	Layer index.

Return value

Number of bits per pixel.

LCD_GetFixedPalette()

Description

Returns the fixed palette mode.

Prototype

```
int LCD_GetFixedPalette(void);
```

Return value

The fixed palette mode. See Chapter 11: "Colors" for more information on fixed palette modes.

LCD_GetFixedPaletteEx()

Description

Returns the fixed palette mode.

Prototype

```
int LCD_GetFixedPaletteEx(int Index);
```

Parameter	Meaning
Index	Layer index.

Return value

The fixed palette mode. See Chapter 9: "Colors" for more information on fixed palette modes.

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	Meaning
Index	Layer index.

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	Meaning
Index	Layer index.

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);
```

Parameter	Meaning
Index	Layer index.

Return value

Magnification factor in X- or Y-axis.

LCDGetXSize(), LCDGetYSize()

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	Meaning
Index	Layer index.

Return value

Physical X/Y-size of the display.

Chapter 25

VNC server

The emWin VNC server can be used for administration of the embedded target and a variety of other purposes. It supports compressed (hexitle) 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 (e.g. 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, it is not included in the basic package.

25.1 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 (e.g. for a manual) from a "live" system.

25.1.1 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.

25.1.2 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.

Authentication

The Server currently does not require authentication. However, this feature can be added to the server easily at request.

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, an other one can connect.

25.2 The VNC viewer

Availability

The subfolder 'Tool' contains a VNC viewer from the AT&T Laboratories Cambridge. It is free software and is distributed under the terms of the GNU Public License. The current version of the VNC software can be downloaded from <http://www.uk.research.att.com/vnc>, where more information about the VNC protocol, as well as source code for servers and clients for different platforms are available.

Version

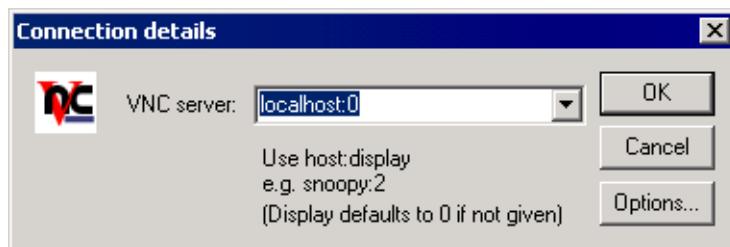
The VNC implementation of emWin has been tested with version 3.3.3R2 and should work with this version or later versions.

Platforms

The viewer is available for different platforms. Please take a look to the website of AT&T Laboratories Cambridge for detailed information about the availability.

25.2.1 Starting the VNC viewer

Start the viewer by double-clicking the file Tool\VNCViewer.exe. It will prompt for the VNC server to be connected:



Connecting to a VNC server using the simulation on the same PC

When running VNCViewer and simulation on the same PC, type 'localhost:0' to connect. ':0' means server index 0. If you omit the server index the viewer assumes server 0. So in the most cases you can type 'localhost' to connect to the simulation.

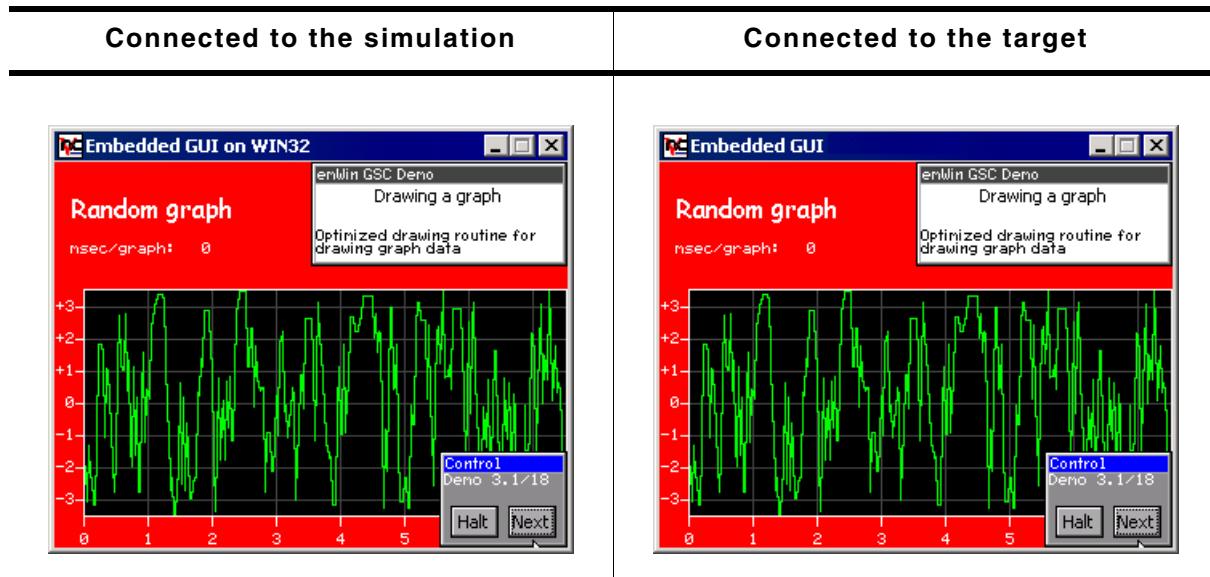
Connecting to a VNC server running on a different PC or the target

To connect to VNC server running on a different PC or on the target system, enter the name or IP address of the machine (optionally followed by a ':' and the server number). To connect to a VNC server on the computer 'Joerg', with IP address 192.168.1.2, you may enter '192.168.1.2:0', or 'Joerg:0' or 'Joerg'.

To connect to a target with IP address 192.168.1.254, enter '192.168.1.254'.

Screenshot

The following screenshots shows the viewer:



25.3 emWin VNC server

25.3.1 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.

Sample

A ready to use sample (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.

25.3.2 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.

25.3.3 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 sample is available under `Sample\GUI_X\GUI_VNC_X_VNCServer.c`, which should require only

smaller modifications. The sample file does not use dynamic memory allocation to allocate memory for the GUI_VNC_CONTEXT structure described later. So this implementation only allows to start one server.

25.4 Required resources

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.

25.5 Configuration options

Type	Macro	Default	Explanation
N	GUI_VNC_BUFFER_SIZE	1000	Frame buffer size. The buffer is located on the stack. Typically bigger sizes result in only minor accelerations. A reasonable buffer size is app. 200 bytes.
B	GUI_VNC_LOCK_FRAME	0	If set to 1 the GUI will be locked during a frame is send to the viewer. This option could make sense if screenshots for a documentation should be made.
S	GUI_VNC_PROGNAME	(see explanation)	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 a faster but needs bigger code (app. 1.4 k more).

25.6 VNC API

The following table lists the available VNC-related functions in alphabetical order. Detailed description of the routines can be found in the sections that follow.

Routine	Explanation
GUI_VNC_AttachToLayer()	Attaches a VNC server to a layer. Without a multi display configuration the given index must be 0.
GUI_VNC_Process()	The actual VNC server; initialises the communication with the viewer.
GUI_VNC_X_StartServer()	Routine to be called to start a VNC viewer.

[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	Meaning
pContext	Pointer to a GUI_VNC_CONTEXT structure.
LayerIndex	Zero based index of layer to be handled by the server.

Return value

0 if the function succeed, != 0 if the function fails.

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	Meaning
pContext	Pointer to a GUI_VNC_CONTEXT structure.
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.

Add. information

The GUI_VNC_CONTEXT structure is used by the server to store connection state information's.

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_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 a sample 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	Meaning
LayerIndex	Layer to be shown by the viewer.
ServerIndex	Server index.

Add. 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 26

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.

26.1 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	Explanation
<code>GUI_Delay()</code>	Delay for a specified period of time.
<code>GUI_Exec()</code>	Execute callback functions (all jobs).
<code>GUI_Exec1()</code>	Execute one callback function (one job only).
<code>GUI_GetTime()</code>	Return the current system time.

GUI_Delay()

Description

Delays for a specified period of time.

Prototype

```
void GUI_Delay(int Period);
```

Parameter	Explanation
<code>Period</code>	Period in ticks until function should return.

Add. 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_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.

Add. 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.

Add. 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

```
int GUI_GetTime(void);
```

Return value

The current system time in ticks.

Add. information

This function will call `GUI_X_GetTime()`.

Chapter 27

Low-Level Configuration (LCD-Conf.h)

Before you can use emWin on your target system, you need to configure the software for your application. Configuring means modifying the configuration (header) files which usually reside in the (sub)directory `Config`. We try to keep the configuration as simple as possible, but there are some configuration macros (in the file `LCD-Conf.h`) which you must modify in order for the system to work properly. These include:

- LCD macros, defining the size of the display as well as optional features (such as mirroring, etc.)
- LCD controller macros, defining how to access the controller you are using.

27.1 Available configuration macros

The following table shows the available macros used for low-level configuration:

Type	Macro	Default	Explanation
General (required) configuration			
S	LCD_CONTROLLER	---	Select LCD controller.
N	LCD_BITSPERPIXEL	---	Specify bits per pixel.
S	LCD_FIXEDPALETTE	---	Specify fixed palette mode. Set to 0 for a user-defined color lookup table (then LCD_PHYSCOLORS must be defined).
N	LCD_XSIZE	---	Define horizontal resolution of LCD.
N	LCD_YSIZE	---	Define vertical resolution of LCD.
Initialisation of the controller			
F	LCD_INIT_CONTROLLER()	---	Initialization sequence for the LCD controller(s). Not applicable with all controllers.
Display orientation			
B	LCD_MIRROR_X	0	Activate to mirror X-axis.
B	LCD_MIRROR_Y	0	Activate to mirror Y-axis.
B	LCD_SWAP_XY	0	Activate to swap X- and Y-axes. If set to 0, SEG lines refer to columns and COM lines refer to rows.
Color configuration			
N	LCD_MAX_LOG_COLORS	256	Maximum number of logical colors that the driver can support in a bitmap.
A	LCD_PHYSCOLORS	---	Defines the contents of the color lookup table. Only required if LCD_FIXEDPALETTE is set to 0.
B	LCD_PHYSCOLORS_IN_RAM	0	Only relevant if physical colors are defined. Puts physical colors in RAM, making them modifiable at run time
B	LCD_REVERSE	0	Activate to invert the display at compile time.
B	LCD_REVERS_LUT	0	Activate to initialize the lookup table with inverted colors at run time.
F	LCD_SET_LUT_ENTRY	---	Used to set a single lookup table or palette RAM entry.
B	LCD_SWAP_RB	0	Activate to swap the red and blue components.
Magnifying the LCD			
N	LCD_XMAG<n>	1	Horizontal magnification factor of LCD.
N	LCD_YMAG<n>	1	Vertical magnification factor of LCD.
Simple bus interface configuration			
F	LCD_READ_A0(Result)	---	Read a byte from LCD controller with A-line low.
F	LCD_READ_A1(Result)	---	Read a byte from LCD controller with A-line high.
F	LCD_WRITE_A0(Byte)	---	Write a byte to LCD controller with A-line low.
F	LCD_WRITE_A1(Byte)	---	Write a byte to LCD controller with A-line high.
F	LCD_WITEM_A1	---	Write multiple bytes to LCD controller with A-line high.
Full bus interface configuration			
F	LCD_READ_MEM(Index)	---	Read the contents of video memory of controller.
F	LCD_READ_REG(Index)	---	Read the contents of a configuration register of controller.
F	LCD_WRITE_MEM(Index, Data)	---	Write to video memory (display data RAM) of controller.

Type	Macro	Default	Explanation
F	LCD_WRITE_REG(Index, Data)	---	Write to a configuration register of controller.
S	LCD_BUSWIDTH	16	Select bus-width (8/16) of LCD controller/ CPU interface.
F	LCD_ENABLE_REG_ACCESS	---	Switch the M/R signal to register access. Not applicable with all controllers.
F	LCD_ENABLE_MEM_ACCESS	---	Switch the M/R signal to memory access. Not applicable with all controllers.
B	LCD_SWAP_BYTE_ORDER	0	Activate to invert the endian mode (swap the high and low bytes) between CPU and LCD controller when using a 16-bit bus interface.
Virtual display			
N	LCD_VXSIZE	LCD_XSIZE	Horizontal resolution of virtual display. Not applicable with all drivers.
N	LCD_VYSIZE	LCD_YSIZE	Vertical resolution of virtual display. Not applicable with all drivers.
N	LCD_XORG<n>	0	LCD controller <n>: leftmost (lowest) X-position.
N	LCD_YORG<n>	0	LCD controller <n>: topmost (lowest) Y-position.
LCD controller configuration: common/segment lines			
N	LCD_XORG<n>	0	LCD controller <n>: leftmost (lowest) X-position.
N	LCD_YORG<n>	0	LCD controller <n>: topmost (lowest) Y-position.
N	LCD_FIRSTSEG<n>	0	LCD controller <n>: first segment line used.
N	LCD_LASTSEG<n>	LCD_XSIZE-1	LCD controller <n>: last segment line used.
N	LCD_FIRSTCOM<n>	0	LCD controller <n>: first common line used.
N	LCD_LASTCOM<n>	LCD_YSIZE-1	LCD controller <n>: last common line used.
COM/SEG lookup tables			
A	LCD_LUT_COM	---	COM lookup table for controller.
A	LCD_LUT_SEG	---	SEG lookup table for controller.
Miscellaneous			
N	LCD_NUM_CONTROLLERS	1	Number of LCD controllers used.
B	LCD_CACHE	1	Deactivate to disable use of display data cache, which slows down the speed of the driver. Not applicable with all drivers.
B	LCD_USE_BITBLT	1	Deactivate to disable BitBLT engine. If set to 1, the driver will use all available hardware acceleration.
B	LCD_SUPPORT_CACHECONTROL	0	Activate to enable cache control functions of LCD_L0_ControlCache() driver API. Not applicable with all controllers.
N	LCD_TIMERINIT0	---	Timing value used by ISR for displaying pane 0 when using CPU as controller.
N	LCD_TIMERINIT1	---	Timing value used by ISR for displaying pane 1 when using CPU as controller.
F	LCD_ON	---	Function replacement macro which switches the LCD on.
F	LCD_OFF	---	Function replacement macro which switches the LCD off.

How to configure the LCD

We recommend the following procedure:

1. Make a copy of a configuration file of similar configuration. Several configuration samples for your particular LCD controller can be found in the folder Sample\LCDConf\xxx, where xxx is your LCD driver.
2. Configure the bus interface by defining the appropriate simple bus or full bus macros.

3. Define the size of your LCD (`LCD_XSIZE`, `LCD_YSIZE`).
4. Select the controller used in your system, as well as the appropriate bpp and the palette mode (`LCD_CONTROLLER`, `LCD_BITSPERPIXEL`, `LCD_FIXEDPALETTE`).
5. Configure which common/segment lines are used, if necessary.
6. Test the system.
7. Reverse X/Y if necessary (`LCD_REVERSE`); go back to step 6 in this case.
8. Mirror X/Y if necessary (`LCD_MIRROR_X`, `LCD_MIRROR_Y`); go back to step 6 in this case.
9. Check all the other configuration switches.
10. Erase unused sections of the configuration.

27.2 General (required) configuration

LCD_CONTROLLER

Description

Defines the LCD controller used.

Type

Selection switch

Add. information

The LCD controller used is designated by the appropriate number. Please refer to Chapter 24: "LCD Drivers" for more information about available options.

Example

Specifies an Epson SED1565 controller:

```
#define LCD_CONTROLLER 1565 /* Selects SED 1565 LCD-controller */
```

LCD_BITSPERPIXEL

Description

Specifies the number of bits per pixel.

Type

Numerical value

LCD_FIXEDPALETTE

Description

Specifies the fixed palette mode.

Type

Selection switch

Add. information

Set the value to 0 to use a color lookup table instead of a fixed palette mode. The macro `LCD_PHYSCOLORS` must then be defined.

LCD_XSIZE, LCD_YSIZE

Description

Define the horizontal and vertical resolution (respectively) of the display used.

Type

Numerical values

Add. information

The values are logical sizes; X-direction specifies the direction which is used as the X-direction by all routines of the LCD driver.
Usually the X-size equals the number of segments.

27.3 Initialisation of the controller

LCD_INIT_CONTROLLER

Description

Initializes the LCD controller(s).

Type

Function replacement

Add. information

This macro must be user-defined to initialize some controllers. It is executed during the `LCD_L0_Init()` and `LCD_L0_Reinit()` routines of the driver. Please consult the data sheet of your controller for information on how to initialize your hardware.

Example

The sample below has been written for and tested with an Epson SED1565 controller using an internal power regulator.

```
#define LCD_INIT_CONTROLLER() \
LCD_WRITE_A0(0xe2); /* Internal reset */ \
LCD_WRITE_A0(0xae); /* Display on/off: off */ \
LCD_WRITE_A0(0xac); /* Power save start: static indicator off */ \
LCD_WRITE_A0(0xa2); /* LCD bias select: 1/9 */ \
LCD_WRITE_A0(0xa0); /* ADC select: normal */ \
LCD_WRITE_A0(0xc0); /* Common output mode: normal */ \
LCD_WRITE_A0(0x27); /* V5 voltage regulator: medium */ \
LCD_WRITE_A0(0x81); /* Enter electronic volume mode */ \
LCD_WRITE_A0(0x13); /* Electronic volume: medium */ \
LCD_WRITE_A0(0xad); /* Power save end: static indicator on */ \
LCD_WRITE_A0(0x03); /* static indicator register set: on (constantly on) */ \
LCD_WRITE_A0(0x2F); /* Power control set: booster, regulator and follower off */ \
LCD_WRITE_A0(0x40); /* Display Start Line */ \
LCD_WRITE_A0(0xB0); /* Display Page Address 0 */ \
LCD_WRITE_A0(0x10); /* Display Column Address MSB */ \
LCD_WRITE_A0(0x00); /* Display Column Address LSB */ \
LCD_WRITE_A0(0xaf); /* Display on/off: on */ \
LCD_WRITE_A0(0xe3); /* NOP */
```

27.4 Display orientation

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-mirror, 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 orientation is o.k. (Text on the display is readable; i.e. runs from left to right, is not upside-down and not mirrored), none of the configuration macros for display orientation are required. Otherwise, start by swapping X/Y if necessary and the mirror the X / Y axis as required or take a look at the table below which indicates which config switches have to be activated in which case. The orientation is handled as follows: Mirroring in X and Y first, then swapping (if selected).

Please note, that if more than one display orientation should be used at runtime, the multi display / multi layer feature is required.

Display	Orientation macros in LCDConf.h
	No orientation macro required
	Use #define LCD_MIRROR_X 1
	Use #define LCD_MIRROR_Y 1
	Use #define LCD_MIRROR_X 1 #define LCD_MIRROR_Y 1
	Use #define LCD_SWAP_XY
	Use #define LCD_SWAP_XY #define LCD_MIRROR_X 1

Display	Orientation macros in LCDConf.h
	Use <code>#define LCD_SWAP_XY</code> <code>#define LCD_MIRROR_X 1</code> <code>#define LCD_MIRROR_Y 1</code>
	Use <code>#define LCD_SWAP_XY</code> <code>define LCD_MIRROR_Y 1</code>

Driver optimizations

We can not optimize all drivers for all possible combinations of orientations and other config switches. In general, the default orientation is optimized. If you need to use a driver in an orientation which has not been optimized, please contact us.

LCD_MIRROR_X

Description

Inverts the X-direction (horizontal) of the display.

Type

Binary switch

0: inactive, X not mirrored (default)

1: active, X mirrored

Add. information

If activated: X -> LCD_XSIZE-1-X.

This macro, in combination with LCD_MIRROR_Y and LCD_SWAP_XY, can be used to support any orientation of the display. Before changing this configuration switch, make sure that LCD_SWAP_XY is set as required by your application.

LCD_MIRROR_Y

Description

Inverts the Y-direction (vertical) of the display.

Type

Binary switch

0: inactive, Y not mirrored (default)

1: active, Y mirrored

Add. information

If activated: Y -> LCD_YSIZE-1-Y.

This macro, in combination with LCD_MIRROR_X and LCD_SWAP_XY, can be used to support any orientation of the display. Before changing this configuration switch, make sure that LCD_SWAP_XY is set as required by your application.

LCD_SWAP_XY

Description

Swaps the horizontal and vertical directions (orientation) of the display.

Type

Binary switch

0: inactive, X-Y not swapped (default)

1: active, X-Y swapped

Add. information

If set to 0 (not swapped), SEG lines refer to columns and COM lines refer to rows.

If activated: X -> Y.

When changing this switch, you will also have to swap the X-Y settings for the resolution of the display (using `LCD_XSIZE` and `LCD_YSIZE`).

27.5 Color configuration

LCD_MAX_LOG_COLORS

Description

Defines the maximum number of colors supported by the driver in a bitmap.

Type

Numerical value (default is 256)

Add. information

If you are using a 4-grayscale LCD, it is usually sufficient to set this value to 4. However, in this case remember not to try to display bitmaps with more than 4 colors.

LCD_PHYSCOLORS

Description

Defines the contents of the color lookup table, if one is used.

Type

Alias

Add. information

This macro is only required if `LCD_FIXEDPALETTE` is set to 0. Refer to the color section for more information.

LCD_PHYSCOLORS_IN_RAM

Description

Puts the contents of the physical color table in RAM if enabled.

Type

Binary switch

0: inactive (default)

1: active

LCD_REVERSE

Description

Inverts the display at compile time.

Type

Binary switch

0: inactive, not reversed (default)

1: active, reversed

LCD_SET_LUT_ENTRY

Description

This macro can be used to set a single LUT entry. If defined the macro will be executed each time the GUI needs to set a LUT entry (typically during the initialisation).

Type

Function replacement

Prototype

```
#define LCD_SET_LUT_ENTRY(Pos, Color)
```

Parameter	Meaning
Pos	Zero based index of LUT entry to be set.
Color	RGB value of color to be set.

LCD_SWAP_RB

Description

Swaps the red and blue color components.

Type

Binary switch

0: inactive, not swapped (default)

1: active, swapped

27.6 Magnifying the LCD

Some hardware requires the LCD to be magnified in order to display images correctly. The software must compensate for hardware which internally needs magnification. It does so by activating a layer (above the driver layer) to automatically handle the job of magnifying the display.

LCD_XMAG

Description

Specifies the horizontal magnification factor of the LCD.

Type

Numerical value (default is 1)

Add. information

A factor of 1 results in no magnification.

LCD_YMAG

Description

Specifies the vertical magnification factor of the LCD.

Type

Numerical value (default is 1)

Add. information

A factor of 1 results in no magnification.

27.7 Simple bus interface configuration

27.7.1 Macros used by a simple bus interface

The following macros are used for LCD controllers with simple 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	Meaning
Result	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

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	Meaning
Result	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

LCD_WRITE_A0

Description

Writes a byte to LCD controller with A0 (C/D) - line low.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A0(Byte)
```

Parameter	Meaning
Byte	Byte to write.

LCD_WRITE_A1

Description

Writes a byte to LCD controller with A0 (C/D) - line high.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A1(Byte)
```

Parameter	Meaning
Byte	Byte to write.

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	Meaning
paBytes	Placeholder for the pointer to the first data byte.
NumberOfBytes	Number of data bytes to be written.

27.7.2 Example of memory mapped interface

The following example demonstrates how to access the LCD by a memory mapped interface:

```
void WriteM_A1(char *paBytes, int NummerOfBytes) {
    int i;
    for (i = 0; i < NummerOfBytes; i++) {
        (*(volatile char *)0xc0001) = *(paBytes + i);
    }
}

#define LCD_READ_A1(Result) Result = (*(volatile char *)0xc0000)
#define LCD_READ_A0(Result) Result = (*(volatile char *)0xc0001)
#define LCD_WRITE_A1(Byte)   (*(volatile char *)0xc0000) = Byte
#define LCD_WRITE_A0(Byte)   (*(volatile char *)0xc0001) = Byte

#define LCD_Writem_A1(paBytes, NummerOfBytes) WriteM_A1(paBytes, NummerOfBytes)
```

27.7.3 Sample routines for connection to I/O pins

Several examples can be found in the folder Sample\LCD_X:

- Port routines for 6800 interface
- Port routines for 8080 interface
- Simple port routines for a serial interface
- Port routines for a simple I2C bus interface

These samples can be used directly. All you need to do is to define the port access macros listed at the top of each example and to map them in your LCDConf.h in a similar manner to that shown below:

```
void LCD_X_Write00(char c);
void LCD_X_Write01(char c);
char LCD_X_Read00(void);
char LCD_X_Read01(void);
#define LCD_WRITE_A1(Byte) LCD_X_Write01(Byte)
#define LCD_WRITE_A0(Byte) LCD_X_Write00(Byte)
#define LCD_READ_A1(Result) Result = LCD_X_Read01()
#define LCD_READ_A0(Result) Result = LCD_X_Read00()
```

Note that not all LCD controllers handle the A0 or C/D bit in the same way. For example, a Toshiba controller requires that this bit be low when accessing data and an Epson SED1565 requires it to be high.

27.8 3 pin SPI configuration

27.8.1 Macros used by a 3 pin SPI interface

The following macros are used for LCD controllers with 4 pin SPI interface.

LCD_WRITE

Description

Writes a byte to the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITE(Byte)
```

Parameter	Meaning
Byte	Byte to write.

LCD_WRITEM

Description

Writes several bytes to the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITEM(paBytes, NumberOfBytes)
```

Parameter	Meaning
paBytes	Placeholder for the pointer to the first data byte.
NumberOfBytes	Number of data bytes to be written.

27.8.2 Sample routines for connection to I/O pins

An example can be found in the folder Sample\LCD_X:

- LCD_X_SERIAL.c, port routines for a serial interface

This sample can be used directly. All you need to do is to define the port access macros listed at the top of the example and to map them in your LCDConf.h in a similar manner to that shown below:

```
void LCD_X_Write(char c);
void LCD_X_WriteM(char * pData, int NumBytes);
#define LCD_WRITE(Byte) LCD_X_Write(Byte)
#define LCD_WRITEM(data, NumBytes) LCD_X_WriteM(data, NumBytes)
```

27.9 4 pin SPI configuration

27.9.1 Macros used by a 4 pin SPI interface

The following macros are used for LCD controllers with 4 pin SPI interface.

LCD_WRITE_A0

Description

Writes a byte to LCD controller with A0 - line low.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A0(Byte)
```

Parameter	Meaning
Byte	Byte to write.

LCD_WRITE_A1

Description

Writes a byte to LCD controller with A0 - line high.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A1(Byte)
```

Parameter	Meaning
Byte	Byte to write.

LCD_WRITEM_A1

Description

Writes several bytes to the LCD controller with A0 - line high.

Type

Function replacement

Prototype

```
#define LCD_WRITEM_A1(paBytes, NumberOfBytes)
```

Parameter	Meaning
paBytes	Placeholder for the pointer to the first data byte.
NumberOfBytes	Number of data bytes to be written.

27.9.2 Sample routines for connection to I/O pins

An example can be found in the folder `Sample\LCD_X`:

- `LCD_X_SERIAL.c`, port routines for a serial interface

This sample can be used directly. All you need to do is to define the port access macros listed at the top of the example and to map them in your `LCDConf.h` in a similar manner to that shown below:

```
void LCD_X_Write00(char c);
void LCD_X_Write01(char c);
void LCD_X_WriteM01(char * pData, int NumBytes);
#define LCD_WRITE_A0(Byte) LCD_X_Write00(Byte)
#define LCD_WRITE_A1(Byte) LCD_X_Write01(Byte)
#define LCD_WRITEITEM_A1(data, NumBytes) LCD_X_WriteM01(data, NumBytes)
```

27.10 I2C bus interface configuration

27.10.1 Macros used by a I2C bus interface

The following macros are used for LCD controllers with I2C bus interface.

LCD_READ_A0

Description

Reads a status byte from LCD controller.

Type

Function replacement

Prototype

```
#define LCD_READ_A0(Result)
```

Parameter	Meaning
Result	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

LCD_READ_A1

Description

Reads a data byte from LCD controller.

Type

Function replacement

Prototype

```
#define LCD_READ_A1(Result)
```

Parameter	Meaning
Result	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

LCD_WRITE_A0

Description

Writes a instruction byte to LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A0(Byte)
```

Parameter	Meaning
Byte	Byte to write.

LCD_WRITE_A1

Description

Writes a data byte to LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A1(Byte)
```

Parameter	Meaning
Byte	Byte to write.

LCD_WRITE_A1

Description

Writes several data bytes to the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A1(paBytes, NumberOfBytes)
```

Parameter	Meaning
paBytes	Placeholder for the pointer to the first data byte.
NumberOfBytes	Number of data bytes to be written.

27.10.2 Sample routines for connection to I/O pins

An example can be found in the folder Sample\LCD_X:

- LCD_X_I2CBUS.c, port routines for a serial interface

This sample can be used directly. All you need to do is to define the port access macros listed at the top of the example and to map them in your LCDConf.h in a similar manner to that shown below:

```
void LCD_X_Write00(char c);
void LCD_X_Write01(char c);
char LCD_X_Read00(void);
char LCD_X_Read01(void);
#define LCD_WRITE_A1(Byte) LCD_X_Write01(Byte)
#define LCD_WRITE_A0(Byte) LCD_X_Write00(Byte)
#define LCD_READ_A1(Result) Result = LCD_X_Read01()
#define LCD_READ_A0(Result) Result = LCD_X_Read00()
```

27.11 Full bus interface configuration

27.11.1 Macros used by a full bus interface

The following macros are used for LCD controllers with a full bus interface.

LCD_READ_MEM

Description

Reads the video memory of the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_READ_MEM(Index)
```

Parameter	Meaning
Index	Index of video memory of controller.

Add. information

This macro defines how to read the video memory of the LCD controller.

In order to configure this switch correctly, you need to know the base address of the video memory, the spacing and if 8/16- or 32-bit accesses are permitted. You should also know the correct syntax for your compiler because this kind of hardware access is not defined in ANSI "C" and is therefore different for different compilers.

LCD_READ_REG

Description

Reads the register of the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_READ_REG(Index)
```

Parameter	Meaning
Index	Index of the register to read.

Add. information

This macro defines how to read the registers of the LCD controller. Usually, the registers are memory-mapped. In this case, the macro can normally be written as a single line.

In order to configure this switch correctly, you need to know the address the registers are mapped to, the spacing and if 8/16- or 32-bit accesses are permitted. You should also know the correct syntax for your compiler because this kind of hardware access is not defined in ANSI "C" and is therefore different for different compilers. However, the syntax shown below works with the majority of them.

Example

If the registers are mapped to a memory area starting at 0xc0000, the spacing is 2 and 16-bit accesses should be used; with most compilers the define should look as follows:

```
#define LCD_READ_REG(Index) *( (U16*) (0xc0000+(Off<<1)) )
```

LCD_WRITE_MEM

Description

Writes data to the video memory of the LCD controller.

Type

Function replacement

Prototype

```
LCD_WRITE_MEM(Index, Data)
```

Parameter	Meaning
Index	Index of video memory of controller.
Data	Data to write to the register

Add. information

This macro defines how to write to the video memory of the LCD controller.

In order to configure this switch correctly, you need to know the base address of the video memory, the spacing and if 8/16- or 32-bit accesses are permitted, as well as the correct syntax for your compiler.

With 8-bit accesses, a value of 1 indicates byte 1.

With 16-bit accesses, a value of 1 indicates word 1.

LCD_WRITE_REG

Description

Writes data to a specified register of the LCD controller

Type

Function replacement

Prototype

```
LCD_WRITE_REG(Index, Data)
```

Parameter	Meaning
Index	Index of the register to write to
Data	Data to write to the register

Add. information

This macro defines how to write to the registers of the LCD controller. If the registers are memory-mapped, the macro can normally be written as a single line.

In order to configure this switch correctly, you need to know the address the registers are mapped to, the spacing and if 8/16- or 32-bit accesses are permitted, as well as the correct syntax for your compiler.

With 8-bit accesses, a value of 1 indicates byte 1.

With 16-bit accesses, a value of 1 indicates word 1.

Example

If the registers are mapped to a memory area starting at 0xc0000, the spacing is 4 and 8-bit access should be used; with most compilers the define should look as follows:

```
#define LCD_WRITE_REG(Index,Data) *((U8 volatile *) (0xc0000+(Off<<2)))=data
```

LCD_BUSWIDTH

Description

Defines bus-width of LCD controller/CPU interface (external display access).

Type

Selection switch
 8: 8-bit wide VRAM
 16: 16-bit wide VRAM (default)

Add. information

Since this completely depends on your hardware, you will have to substitute these macros. The Epson SED1352 distinguishes between memory and register access; memory is the video memory of the LCD controller and registers are the 15 configuration registers. The macros define how to access (read/write) VRAM and registers.

LCD_ENABLE_REG_ACCESS**Description**

Enables register access and sets the M/R signal to high.

Type

Function replacement

Prototype

```
#define LCD_ENABLE_REG_ACCESS() MR = 1
```

Add. information

Only used for Epson SED1356 and SED1386 controllers.

After using this macro, `LCD_ENABLE_MEM_ACCESS` must also be defined in order to switch back to memory access after accessing the registers.

LCD_ENABLE_MEM_ACCESS**Description**

Switches the M/R signal to memory access. It is executed after register access functions and sets the M/R signal to low.

Type

Function replacement

Prototype

```
#define LCD_ENABLE_MEM_ACCESS() MR = 0
```

Add. information

Only used for Epson SED1356 and SED1386 controllers.

LCD_SWAP_BYTE_ORDER**Description**

Inverts the endian mode (swaps the high and low bytes) between CPU and LCD controller when using a 16-bit bus interface.

Type

Binary switch
 0: inactive, endian modes not swapped (default)
 1: active, endian modes swapped

27.11.2 Configuration example

The example assumes the following:

Base address video memory	0x80000
Base address registers	0xc0000
Access to video RAM	16-bit

Access to register	16-bit
Distance between adjacent video memory locations	2 bytes
Distance between adjacent register locations	2 bytes

```
#define LCD_READ_REG(Index)      *((U16*)(0xc0000+(Off<<1)))
#define LCD_WRITE_REG(Index,data) *((U16*)(0xc0000+(Off<<1)))=data
#define LCD_READ_MEM(Index)       *((U16*)(0x80000+(Off<<1)))
#define LCD_WRITE_MEM(Index,data) *((U16*)(0x80000+(Off<<1)))=data
```

27.12 Hardware access for multiple LCD controllers

Multiple LCD controllers (not multiple displays or multiple layers!) are only supported for LCD's connected by a simple bus interface, a 3 or 4 pin SPI interface or a I2C bus interface. If more than one LCD controller is used for the LCD, you must define access macros for each of them individually, according to your hardware. Four macros are needed per LCD controller. The macros for the additional controllers are often very similar to those for the first one. With a direct bus connection, usually only the addresses are different. When I/O pins are used, the sequence for the access is the same except for the CHIP-SELECT signal.

When using more than one controller, add an 'C' underscore and the index of the controller as the postfix for controllers > 0. For example:

Controller #0: LCD_READ_A0
 Controller #1: LCD_READ_A0C1
 and so on.

Note that the first controller is considered to be controller #0, so that a second controller would be defined as #1, etc. The macros for additional LCD controllers are listed as follows.

Second LCD controller

Type	Macro	Explanation
F	LCD_READ_A0C1(Result)	LCD controller 1: Read a byte with A0 = 0.
F	LCD_READ_A1C1(Result)	LCD controller 1: Read a byte with A0 = 1.
F	LCD_WRITE_A0C1(Byte)	LCD controller 1: Write a byte with A0 = 0.
F	LCD_WRITE_A1C1(Byte)	LCD controller 1: Write a byte with A0 = 1.

Third LCD controller

Type	Macro	Explanation
F	LCD_READ_A0C2(Result)	LCD controller 2: Read a byte with A0 = 0.
F	LCD_READ_A1C2(Result)	LCD controller 2: Read a byte with A0 = 1.
F	LCD_WRITE_A0C2(Byte)	LCD controller 2: Write a byte with A0 = 0.
F	LCD_WRITE_A1C2(Byte)	LCD controller 2: Write a byte with A0 = 1.

Fourth LCD controller

Type	Macro	Explanation
F	LCD_READ_A0C3(Result)	LCD controller 3: Read a byte with A0 = 0.
F	LCD_READ_A1C3(Result)	LCD controller 3: Read a byte with A0 = 1.
F	LCD_WRITE_A0C3(Byte)	LCD controller 3: Write a byte with A0 = 0.
F	LCD_WRITE_A1C3(Byte)	LCD controller 3: Write a byte with A0 = 1.

27.13 Virtual display support

LCD_VXSIZE, LCD_VYSIZE

Description

Define the horizontal and vertical resolution (respectively) of the virtual display.

Type

Numerical values

Add. information

The values are logical sizes; X-direction specifies the direction which is used as X-direction by all routines of the LCD driver.

If a virtual display is not used, these values should be the same as the values for `LCD_XSIZE`, `LCD_YSIZE` (these are the default settings).

The virtual display feature requires hardware support and is not available with all drivers.

LCD_XORG<n>, LCD_YORG<n>

Description

Define the horizontal and vertical origin of the display (respectively) controlled by the configured driver.

Type

Numerical values

Add. information

In a single display system, both macros are usually set to 0 (the default value).

27.14 LCD controller configuration: common/segment lines

For most LCDs, the setup of common (COM) and segment (SEG) lines is straightforward and neither special settings for COM/SEG lines nor the configuration macros in this section are required.

This section explains how the LCD controller(s) is physically connected to your display. The direction does not matter; it is only assumed that continuous COM and SEG lines are used. If the direction of SEGs or COMs is reversed, use `LCD_MIRROR_X`/`LCD_MIRROR_Y` to set them in the direction required by your application. If non-continuous COM/SEG lines have been used, you have to modify the driver (putting in a translation table will do) or -- even better -- go back to the hardware (LCD module) designer and ask him/her to start over.

LCD_XORG<n>, LCD_YORG<n>

Description

Define the horizontal and vertical origin of the display (respectively) controlled by the configured driver.

Type

Numerical values

Add. information

In a single display system, both macros are usually set to 0 (the default value).

LCD_FIRSTSEG<n>

Description

Controller <n>: first segment line used.

Type

Numerical value

LCD_LASTSEG<n>

Description

Controller <n>: last segment line used.

Type

Numerical value

LCD_FIRSTCOM<n>

Description

Controller <n>: first common line used.

Type

Numerical value

LCD_LASTCOM<n>

Description

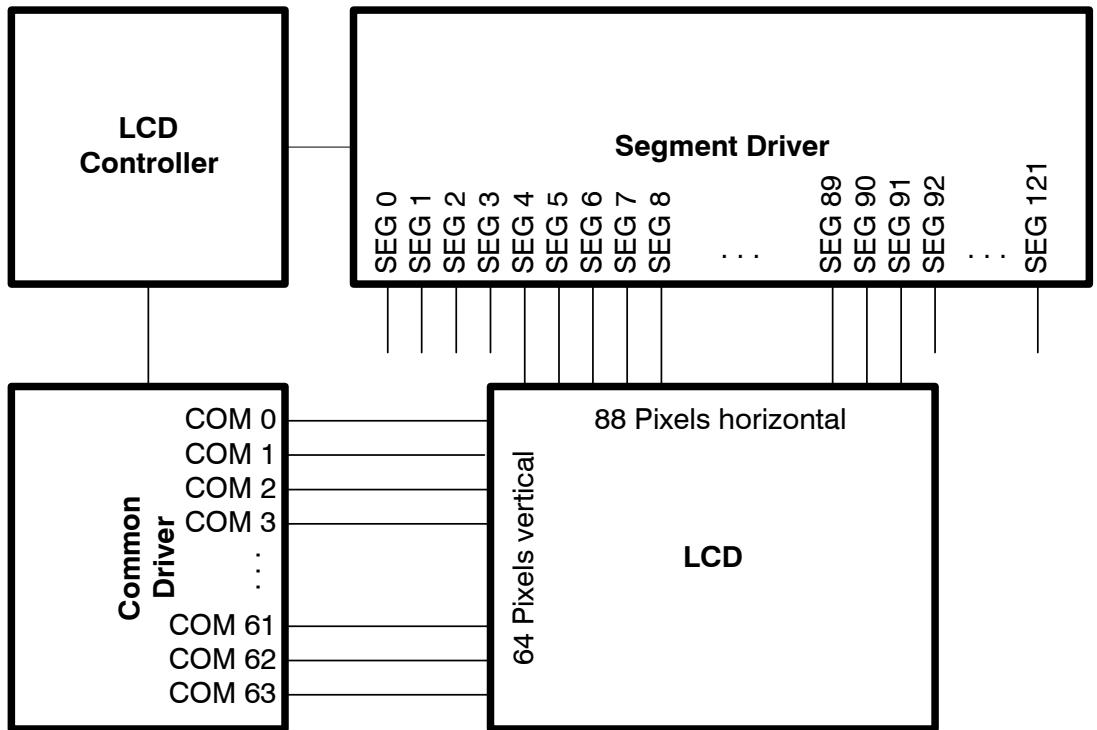
Controller <n>: last common line used.

Type

Numerical value

27.14.1 Single LCD controller configuration

The following block diagram shows a single LCD, controlled by a single LCD controller, using external COM and SEG drivers. All outputs of the common driver (COM0-COM63) are being used, but only some outputs of the segment driver (SEG4-SEG91). Note that for simplicity the video RAM is not shown in the diagram.



Configuration for the above example

```

#define LCD_FIRSTSEG0      (4)      /* Contr.0: first segment line used */
#define LCD_LASTSEG0       (91)     /* Contr.0: last segment line used */
#define LCD_FIRSTCOM0      (0)      /* Contr.0: first com line used */
#define LCD_LASTCOM0       (63)     /* Contr.0: last com line used */

```

Please also note that the above configuration is identical if the COM or SEG lines are mirrored and even if the LCD is built-in sideways (90° turned, X-Y swapped). The same applies if the COM/SEG drivers are integrated into the LCD controller, as is the case for some controllers designed for small LCDs. A typical example for this type of controller would be the Epson SED15XX series.

27.14.2 Configuring additional LCD controllers

emWin offers the possibility of using more than one (up to four) LCD controllers with one LCD. The configuration switches are identical to the switches for the first controller (controller 0), except the index is 1, 2 or 3 instead of 0.

Second LCD controller

Type	Macro	Explanation
N	LCD_FIRSTSEG1	LCD controller 1: first segment line used.
N	LCD_LASTSEG1	LCD controller 1: last segment line used.
N	LCD_FIRSTCOM1	LCD controller 1: first com line used.
N	LCD_LASTCOM1	LCD controller 1: last com line used.
N	LCD_XORG1	LCD controller 1: leftmost (lowest) X-position.
N	LCD_YORG1	LCD controller 1: topmost (lowest) Y-position.

Third LCD controller

Type	Macro	Explanation
N	LCD_FIRSTSEG2	LCD controller 2: first segment line used.
N	LCD_LASTSEG2	LCD controller 2: last segment line used.
N	LCD_FIRSTCOM2	LCD controller 2: first com line used.
N	LCD_LASTCOM2	LCD controller 2: last com line used.
N	LCD_XORG2	LCD controller 2: leftmost (lowest) X-position.
N	LCD_YORG2	LCD controller 2: topmost (lowest) Y-position.

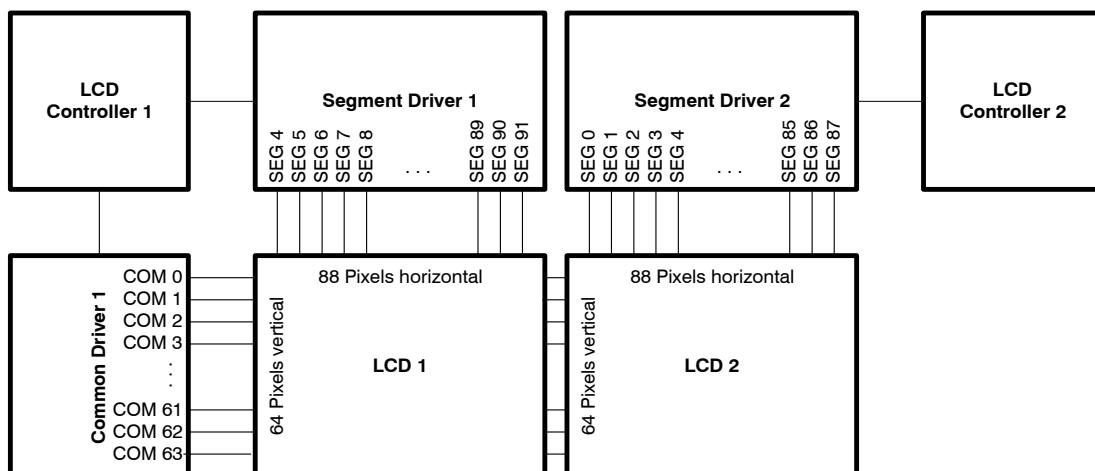
Fourth LCD controller

Type	Macro	Explanation
N	LCD_FIRSTSEG3	LCD controller 3: first segment line used.
N	LCD_LASTSEG3	LCD controller 3: last segment line used.
N	LCD_FIRSTCOM3	LCD controller 3: first com line used.
N	LCD_LASTCOM3	LCD controller 3: last com line used.
N	LCD_XORG3	LCD controller 3: leftmost (lowest) X-position.
N	LCD_YORG3	LCD controller 3: topmost (lowest) Y-position.

When using more than one LCD controller, be sure to remember to define the number of controllers used (see macro `LCD_NUM_CONTROLLERS` in the next section).

Example

The following diagram shows a hardware configuration using two LCD controllers. The COM lines are driven by the common driver connected to controller 1 and are directly connected to the second LCD. LCD 1 is connected to segment driver 1 using SEG lines 4 to 91. LCD 2 is driven by SEG 0 to SEG 87 of segment driver 2.



Configuration for the above example

```
#define LCD_FIRSTSEG0      (4)      /* Contr.0: first segment line used */
#define LCD_LASTSEG0        (91)     /* Contr.0: last segment line used */
#define LCD_FIRSTCOM0        (0)      /* Contr.0: first com line used */
#define LCD_LASTCOM0         (63)     /* Contr.0: last com line used */
#define LCD_XORG0            (0)      /* Contr.0: leftmost (lowest) x-Pos */
#define LCD_YORG0            (0)      /* Contr.0: topmost (lowest) y-Pos */
#define LCD_FIRSTSEG1        (0)      /* Contr.1: first segment line used */
#define LCD_LASTSEG1          (87)    /* Contr.1: last segment line used */
#define LCD_FIRSTCOM1        (0)      /* Contr.1: first com line used */
#define LCD_LASTCOM1         (63)     /* Contr.1: last com line used */
#define LCD_XORG1            (88)     /* Contr.1: leftmost (lowest) x-Pos */
#define LCD_YORG1            (0)      /* Contr.1: topmost (lowest) y-Pos */
```

27.15 COM/SEG lookup tables

When using "chip on glass" technology, it is sometimes very difficult to ensure that the COM and SEG outputs of the controller(s) are connected to the display in a linear fashion. In this case a COM/SEG lookup table may be required in order to inform the driver as to how the COM/SEG lines are connected.

LCD_LUT_COM

Description

Defines a COM lookup table for the controller.

Type

Alias

Example

Let us assume your display contains only 10 COM lines and their connecting order is 0, 1, 2, 6, 5, 4, 3, 7, 8, 9. To configure the LCD driver so that the COM lines are accessed in the correct order, the following macro should be added to your `LCD-Conf.h`:

```
#define LCD_LUT_COM 0, 1, 2, 6, 5, 4, 3, 7, 8, 9
```

If you need to modify the segment order, you should use the macro `LCD_LUT_SEG` in the same manner.

LCD_LUT_SEG

Description

Defines a SEG lookup table for the controller.

Type

Alias

27.16 Miscellaneous

LCD_NUM_CONTROLLERS

Description

Defines the number of LCD controllers used.

Type

Numerical value (default is 1)

LCD_CACHE

Description

Controls caching of video memory in CPU memory.

Type

Binary switch

0: disabled, no display data cache used

1: enabled, display data cache used (default)

Add. information

This switch is not supported by all LCD drivers.

Using a display data cache (which speeds up access) is recommended if access to the video memory is slow, which is usually the case with larger displays and simple bus interfaces (particularly if port-access or serial interfaces are used). Disabling the cache will slow down the speed of the driver.

LCD_USE_BITBLT

Description

Controls usage of hardware acceleration.

Type

Binary switch

0: disabled, BitBLT engine is not used

1: enabled, BitBLT engine is used (default)

Add. information

Disabling the BitBLT engine will instruct the driver not to use the available hardware acceleration.

LCD_SUPPORT_CACHECONTROL

Description

Switch support for the `LCD_L0_ControlCache()` function of the driver.

Type

Binary switch

0: disabled, `LCD_L0_ControlCache()` may not be used (default)

1: enabled, `LCD_L0_ControlCache()` may be used

Add. information

The API function `LCD_L0_ControlCache()` permits locking, unlocking, or flushing of the cache. For more information, refer to Chapter 28: "LCD Driver API".

Please note that this feature is intended only for some LCD controllers with simple bus interface, for which it is important to access the controller as little as possible in order to maximize speed. For other controllers, this switch has no effect.

LCD_TIMERINIT0

Description

Timing value used by an interrupt service routine for displaying pane 0 of a pixel.

Type

Numerical value

Add. information

This macro is only relevant when no LCD controller is used, since it is then the job of the CPU to update the display in an interrupt service routine.

LCD_TIMERINIT1

Description

Timing value used by an interrupt service routine for displaying pane 1 of a pixel.

Type

Numerical value

Add. information

This macro is only relevant when no LCD controller is used, since it is then the job of the CPU to update the display in an interrupt service routine.

LCD_ON

Description

Switches the LCD on.

Type

Function replacement

LCD_OFF

Description

Switches the LCD off.

Type

Function replacement

Chapter 28

High-Level Configuration (GUIConf.h)

High-level configuration is relatively simple. In the beginning, you can normally use the existing configuration files (for example, those used in the simulation). Only if there is a need to fine-tune the system, or to minimize memory consumption, does the high-level configuration file `GUIConf.h` need to be changed. This file is usually located in the `Config` subdirectory of your project's root directory. Use the file `GUIConf.h` for any high-level configuration.

The second thing to do when using emWin on your hardware is to change the hardware-dependent functions, located in the file `Sample\GUI_X\GUI_X.c`.

28.1 General notes

The configuration options explained in this chapter are the available options for the general library. Furthermore exist configuration options for the optional window manager and the widget library. For details please refer to the chapters 'The Window Manager' and 'Window Objects'.

28.2 How to configure the GUI

We recommend the following procedure:

1. Make a copy of the original configuration file.
2. Review all configuration switches.
3. Erase unused sections of the configuration.

28.2.1 Sample configuration

The following is a short sample GUI configuration file (ConfigSample\GUIConf.h):

```
#define GUI_WINSUPPORT          (1) /* Use window manager if true (1) */
#define GUI_SUPPORT_TOUCH        (1) /* Support a touch screen */
#define GUI_ALLOC_SIZE           5000 /* Size of dynamic memory */
#define GUI_DEFAULT_FONT  &GUI_Font6x8 /* This font is used as default */
```

28.3 Available GUI configuration macros

The following table shows the available macros used for high-level configuration of emWin

Type	Macro	Default	Explanation
N	GUI_ALLOC_SIZE	---	Defines the size (number of bytes available) for optional dynamic memory. Dynamic memory is required only for windows and memory devices. This memory will only be used if the GUIAlloc module is linked, which should happen only if dynamic memory is required.
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_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	Define which font is used as 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.
N	GUI_MAXBLOCKS	---	Defines the number of available memory blocks for the memory management of emWin. The maximum number of blocks depends per default on GUI_ALLOC_SIZE.
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 Chapter 13: "Execution Model: Single Task/Multitask").

Type	Macro	Default	Explanation
F	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.
B	GUI_OS	0	Activate to enable multitasking support with multiple tasks calling emWin (see Chapter 13: "Execution Model: Single Task/Multitask").
B	GUI_SUPPORT_ARABIC	0	Activates the Arabic language support (refer to Chapter 23.2: "Arabic language support").
B	GUI_SUPPORT_LARGE_BITMAPS	0	If a system with a 16 bit CPU (sizeof(int) == 2) should display bitmaps >64Kb this configuration macro should be set to 1.
B	GUI_SUPPORT_MEMDEV	1	Enables optional memory device support. Not using memory devices will save about 40 bytes of RAM for function pointers and will slightly accelerate execution.
B	GUI_SUPPORT_MOUSE	0	Enables the optional mouse support.
B	GUI_SUPPORT_ROTATION		Enables text rotation support (default if compiler supports unlimited function pointer calls).
B	GUI_SUPPORT_TOUCH	0	Enables optional touch-screen support.
B	GUI_SUPPORT_UNICODE	1	Enables support for Unicode characters embedded in 8-bit strings. Please note: Unicode characters may always be displayed, as character codes are always treated as 16-bit.
B	GUI_TRIAL_VERSION	0	Marks the compiler output as evaluation version.
B	GUI_WINSUPPORT	0	Enables optional window manager support.

28.3.1 GUI_ALLOC_SIZE

The dynamic memory is used by the window manager/widget library (optional) and the memory devices (optional). The following gives you an overview of the memory requirements of these modules.

Memory requirement of the window manager

If the window manager is used, approximately 50 bytes are used per application defined window and approximately 100 bytes per widget. Typical applications using the window manager/widgets requires app. 2500 bytes.

Memory requirement of the memory devices

Depending on the color depth used one pixel of a memory device requires 1 or 2 bytes of RAM. Configurations with a color depth between 1 and 8bpp (LCD_BITSPERPIXEL between 1 and 8) uses 1 byte of RAM. A color depth >8 and <=16bpp uses 2 bytes of RAM. If using the memory device module to prevent the display from flickering when using drawing operations for overlapping items a minimum of 4000 bytes is recommended. If enough RAM is available the allocated size should be sufficient to store the pixels of the largest window plus the memory requirement of the window manager.

Examples

- If the current configuration is 16bpp and the largest window is 320x240 pixels
 $320 \times 240 \times 2 = 153600$ bytes are useful for the memory device module.
- If the current configuration is 16bpp and the largest window is 320x240 pixels
 $320 \times 240 = 76800$ bytes are useful for the memory device module.

28.3.2 GUI_MAXBLOCKS

Defines the maximum number of available memory blocks for the memory management system. Per default it depends on `GUI_ALLOC_SIZE` and is calculated as follows:

```
GUI_MAXBLOCKS = (2 + GUI_ALLOC_SIZE / 32)
```

For the most applications the default is a good value. If an application requires more or less memory blocks it can be configured using the following define (where `xxx` is the maximum number of blocks):

```
#define GUI_MAXBLOCKS xxx
```

28.3.3 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. Please 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 software
contains an eval-
build of emWin.

A license is
required to use
it in a product.

www.segger.com
```

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.

Sample

```
#define GUI_TRIAL_VERSION 1
```

28.4 GUI_X routine reference

When using emWin with your hardware, there are several hardware-dependent functions which must exist in your project. When using the simulation, the library already contains them. A sample file can be found under `Sample\GUI_X\GUI_X.c`. The following table lists the available hardware-dependent functions in alphabetical order within their respective categories. Detailed description of the routines can be found in the sections that follow.

Routine	Explanation
Init routine	
<code>GUI_X_Init()</code>	Called from <code>GUI_Init()</code> ; can be used to initialize hardware.
Timing routines	
<code>GUI_X_Delay()</code>	Return after a given period.
<code>GUI_X_ExecIdle()</code>	Called only from non-blocking functions of window manager.
<code>GUI_X_GetTime()</code>	Return the system time in milliseconds.

Routine	Explanation
Kernel interface routines	
<code>GUI_X_InitOS()</code>	Initialize the kernel interface module (create a resource semaphore/mutex).
<code>GUI_X_GetTaskId()</code>	Return a unique, 32-bit identifier for the current task/thread.
<code>GUI_X_Lock()</code>	Lock the GUI (block resource semaphore/mutex).
<code>GUI_X_Unlock()</code>	Unlock the GUI (unblock resource semaphore/mutex).
Debugging	
<code>GUI_X_Log()</code>	Return debug information; required if logging is enabled.

28.5 Init routine

`GUI_X_Init()`

Description

Called from `GUI_Init()`; can be used to initialize hardware.

Prototype

```
void GUI_X_Init(void);
```

28.6 Timing routines

`GUI_X_Delay()`

Description

Returns after a specified time period in milliseconds.

Prototype

```
void GUI_X_Delay(int Period)
```

Parameter	Meaning
<code>Period</code>	Period in milliseconds.

`GUI_X_ExecIdle()`

Description

Called only from non-blocking functions of the window manager.

Prototype

```
void GUI_X_ExecIdle(void);
```

Add. 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.

28.7 Kernel interface routines

Detailed descriptions for these routines may be found in Chapter 13: "Execution Model: Single Task/Multitask".

28.8 Debugging

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	Meaning
s	Pointer to the string to be sent.

Add. 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	Explanation
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.

28.9 Dynamic memory

emWin contains its own memory management system. But it is also possible to use your own memory management system. The following table shows the available macros used for dynamic memory configuration of emWin:

Type	Macro	Explanation
F	GUI_ALLOC_ALLOC(Size)	Used to allocate a memory block, returns a memor handle.
F	GUI_ALLOC_FREE(pMem)	Used to release a memory block.
F	GUI_ALLOC_GETMAXSIZE()	Returns the maximum number of bytes of available memory.
F	GUI_ALLOC_H2P(hMem)	Converts a memory handle to a memory pointer.
A	GUI_HMEM	Type of a memory handle.

GUI_ALLOC_ALLOC

Description

Allocates a memory block and returns a handle to it.

Type

Function replacement.

Prototype

```
#define GUI_ALLOC_ALLOC(Size)
```

Parameter	Meaning
Size	Size of required memory block

Example

```
#define GUI_ALLOC_ALLOC(Size) malloc(Size)
```

GUI_ALLOC_FREE

Description

Releases a memory block.

Type

Function replacement.

Prototype

```
#define GUI_ALLOC_FREE(pMem)
```

Parameter	Meaning
pMem	Pointer to memory block to be released.

Example

```
#define GUI_ALLOC_FREE(pMem) free(pMem)
```

GUI_ALLOC_GETMAXSIZE

Description

Returns the maximum number of bytes of available memory.

Type

Function replacement.

Prototype

```
#define GUI_ALLOC_GETMAXSIZE()
```

Example

```
#define GUI_ALLOC_GETMAXSIZE() 10000
```

GUI_ALLOC_H2P**Description**

Converts a memory handle to a memory pointer.

Type

Function replacement.

Prototype

```
#define GUI_ALLOC_H2P(hMem)
```

Parameter	Meaning
hMem	Memory handle to be converted to a memory pointer.

Example

```
#define GUI_ALLOC_H2P(hMem) hMem
```

GUI_HMEM**Description**

Defines the type of a memory handle.

Type

Text replacement.

Example

```
#define GUI_HMEM void *
```

Example

The following sample is an excerpt from the GUIConf.h which uses the standard dynamic memory system:

```
#include <malloc.h>
#include <memory.h>

#define GUI_HMEM           void *
#define GUI_ALLOC_ALLOC(Size)    malloc(Size)
#define GUI_ALLOC_FREE(pMem)     free(pMem)
#define GUI_ALLOC_H2P(hMem)      hMem
#define GUI_ALLOC_GETMAXSIZE()   10000
```

**28.10 Special considerations for certain Compilers/
CPUs****28.10.1 AVR with IAR-Compiler**

When using an Atmel AVR CPU and a IAR compiler and you need to put const data in flash ROM you need to add 2 additional configuration macros to GUIConf.h:

Type	Macro	Default	Explanation
A	GUI_UNI_PTR	---	Define "universal pointer".
A	GUI_CONST_STORAGE	const	Define const storage.

GUI_UNI_PTR

Description

Defines a "universal pointer" which can point to RAM and flash ROM. On some systems it can be necessary since a default pointer can access RAM only, not the built-in Flash.

Type

Alias.

Example

```
#define GUI_UNI_PTR __generic
```

GUI_CONST_STORAGE

Description

Defines the const storage. On some systems it can be necessary since otherwise constants are copied into RAM.

Type

Alias.

Example

```
#define GUI_CONST_STORAGE __flash const
```

28.10.2 8051 Keil compiler and other 8-bit CPU compilers

Keils 8051 Compiler (tested in V5 & V6) has limitation as far as function pointers are concerned. The compiler is limited in respect to the number of parameters which can be passed to a function called thru a function pointer (indirect function call). Some other 8-bit compilers (for 6502 type architectures, such as ST7, but possibly also other chips) may also have a similar limitation. The config switch below allows to circumvent most of these limitations by avoiding function calls with multiple parameters.

Type	Macro	Default	Explanation
A	GUI_COMPILER_SUPPORTS_FP	1	Set to 0 if the compiler does not support complex function calls via function pointers.

GUI_COMPILER_SUPPORTS_FP

Description

Used to enable/disable the use of complex function pointers.

Type

Alias.

Example

```
#define GUI_COMPILER_SUPPORTS_FP 0 /* Disable the use of complex function pointers */
```

Add. information

Disabling this config switch will make it possible to use the software even with very limited "C"-compilers for small chips. However, this comes at a price:

The available functionality is limited as well. The following limitations apply in this case:

- Text rotation can not be used
- Compressed bitmaps can not be used
- Higher level software, such as memory devices, window manager & VNC server can not be used

- Antialiasing can not be used
- Some other (smaller) restrictions may apply.

Chapter 29

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 64kb ROM and 2kb RAM can be supported by emWin. The actual performance and resource usage depends on many factors (CPU, compiler, memory model, optimization, configuration, interface to LCD controller, 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.

29.1 Performance benchmark

We use a benchmark test to measure the speed of the software 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

All values are in $\mu\text{s}/\text{pixel}$.

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
M16C/60 (16 bit), 16MHz	SED1560 (1560)	1	1.68	11.0	4.81	8.96	34.8	35.5	21.5	8.39
M16C/60 (16 bit), 16MHz	T6963 (6963)	1	5.62	21.4	6.59	7.19	30.2	34.6	51.3	72.6
M16C/80 (16 bit), 20MHz	S1D13705 (1375)	1	0.11	6.12	1.99	1.34	8.37	8.43	13.7	7.13
M16C/80 (16 bit), 20MHz	S1D13705 (1375)	4	0.39	5.41	1.94	2.22	8.73	3.86	13.7	1.62
M16C/80 (16 bit), 20MHz	S1D13705 (1375)	8	0.63	5.45	2.30	3.26	7.65	3.23	2.81	1.61
V850SB1 (32 bit), 20MHz	S1D13806 (1375)	8	0.06	2.92	0.63	0.66	4.16	2.18	12.1	0.80
V850SB1 (32 bit), 20MHz	S1D13806 (1375)	16	0.12	3.08	0.69	0.67	2.56	2.58	4.67	1.24
ARM720T (32 bit), 50MHz	internal LCD-Controller (3200)	16	0.14	1.72	0.54	0.51	1.44	1.55	2.44	0.34

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 1bpp bitmaps. An area of 58*8 pixels is filled with a 1bpp bitmap.

Bench 5: Bitmap 2bpp

Bench the speed of 2bpp bitmaps. An area of 32*11 pixels is filled with a 2bpp bitmap.

Bench6: Bitmap 4bpp

Bench the speed of 4bpp bitmaps. An area of 32*11 pixels is filled with a 4bpp bitmap.

Bench7: Bitmap 8bpp

Bench the speed of 8bpp 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.

29.2 Memory requirements of the GUI on ARM7

Due to the increasing popularity of the ARM processor, we have detailed the memory requirements of the individual GUI components below. The values have been obtained on a system as follows:

CPU	ARM7
Toolchain	IAR embedded workbench version 4.20a
Processor mode	Thumb
Compile options	Size optimisation high

Results:

Component	ROM [bytes]	RAM [bytes]	Stack [bytes]	Explanation
Core	5248	73	600	Memory requirements of a typical 'Hello world' application without using additional software items.
Window manager	+ 6260	+ 2541	+ 600	Additional memory requirements of a 'Hello world' application when using the window manager. The configured GUI_ALLOC_SIZE for the measurement was 2000.
Memory devices	+ 4764	+ 7121	+ 200	Additional memory requirements of a 'Hello world' application when using memory devices. The configured GUI_ALLOC_SIZE for the measurement was 8000.
Antialiasing	+ 4500	+ 2 * LCD_XSIZE	-	Additional memory requirements for the antialiasing software item.
Driver	+ 2 - 8kB	(see explanation)	-	The memory requirements of the driver depend on the configured driver and if a data cache is used or not. Without a data cache the driver uses app. < 20 bytes. For details please refer to the driver documentation.
Multilayer	+ 2 - 8kB	(see explanation)		If working with a multi layer or a multi display configuration additional memory for each additional layer is required. Each layer

29.3 Memory requirements of typical applications

The following shows the memory requirements for 2 projects. The first project works with a minimum configuration. The second project is a "typical" application using the window manager and different widgets.

29.3.1 Minimum project ("Hello world")

This project uses only the basic functions for displaying text. The used application can be found under Sample\GUI\BASIC_HelloWorld.c.

29.3.1.1 Configuration

The following excerpts of the files GUIConf.h and LCDConf.h shows the used configuration options.

GUIConf.h:

```
#define GUI_WINSUPPORT          0
#define GUI_SUPPORT_TOUCH        0
#define GUI_SUPPORT_MOUSE        0
#define GUI_SUPPORT_MEMDEV       0
#define GUI_SUPPORT_AA           0
#define GUI_DEFAULT_FONT         &GUI_Font6x8
```

LCDConf.h:

```
#define LCD_CONTROLLER          6963
#define LCD_BITSPERPIXEL         1
#define LCD_MAX_LOG_COLORS       2
#define LCD_CACHE                0
```

29.3.1.2 System

The following table shows the hardware and the toolchain details of the project:

Detail	Description
CPU	M16C
Toolchain	IAR embedded workbench version 1.36A
Memory model	near
Compile options	Size optimisation (-z9)

29.3.1.3 Memory requirements

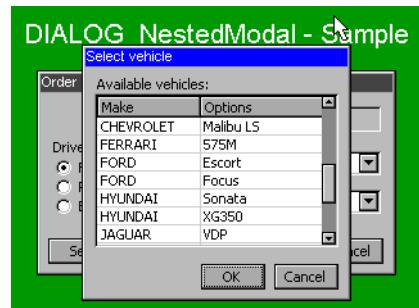
The following table shows the memory requirement of the project:

Description	ROM [bytes]	RAM [bytes]
GUI code and const data	5898	62
GUI data (fonts)	1828	--
Application	38	--
Stack	--	272
Startup code	258	--
Library	71	--
Total		
	8093	334

29.3.2 Window application

The application initially opens a dialog. This dialog contains interface elements (widgets). One of these (a button labeled "car selection") opens an other dialog which contains a list view and is used to select the vehicle. The resource usage of this application should be more or less typical for applications using the window manager and some, but not all widgets.

The application can be found under Sample\GUI\Dialog_NestedModal.c.



29.3.2.1 Configuration

The following excerpts of the files GUIConf.h and LCDConf.h shows the used configuration options.

GUIConf.h:

```
#define GUI_WINSUPPORT          1
#define GUI_SUPPORT_TOUCH        0
#define GUI_SUPPORT_MOUSE        1
#define GUI_SUPPORT_MEMDEV       0
#define GUI_SUPPORT_AA           0
#define GUI_MAXBLOCKS            160
#define GUI_ALLOC_SIZE            3500
#define BUTTON_FONT_DEFAULT      &GUI_Font13_ASCII
#define DROPODOWN_FONT_DEFAULT   &GUI_Font13_ASCII
#define EDIT_FONT_DEFAULT         &GUI_Font13_ASCII
#define HEADER_FONT_DEFAULT      &GUI_Font13_ASCII
#define LISTBOX_FONT_DEFAULT     &GUI_Font13_ASCII
#define LISTVIEW_FONT_DEFAULT    &GUI_Font13_ASCII
#define TEXT_FONT_DEFAULT         &GUI_Font13_ASCII
#define GUI_DEFAULT_FONT          &GUI_Font6x8
```

LCDConf.h:

```
#define LCD_CONTROLLER           1375
#define LCD_BITSPERPIXEL          8
```

29.3.2.2 System

The following table shows the hardware and the toolchain details of the project:

Detail	Description
CPU	ARM AT91
Toolchain	IAR embedded workbench version 3.40A
Code model	small
Compile options	Size optimization high

29.3.2.3 Memory requirements

The following table shows the memory requirement of the project:

Description	ROM [bytes]	RAM [bytes]
GUI code and const data	42731	5224
GUI data (fonts)	12412	--
Application	2731	40
Stack	--	1400
Startup code	304	--
Library	1480	388
Total		
	59658	7052

Chapter 30

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 the emWin support.

30.1 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.

30.1.1 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"

30.1.2 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

This kind 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 loose significant bits"
 "Statement not reached"
 "Meaningless statements were deleted during optimization"
 "Condition is always true/false"
 "Unreachable code"

Most compilers offers a way to supress 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 sample:

```
#define GUI_USE_PARA(para) para=para;
```

emWin uses this macro wherever necessary to avoid this type of warning.

30.1.3 Compiler errors

emWin assumes that the used compiler is ANSI 'C' compatible. The compiler should cover the international standard ISO/IEC 9899:1990, ISO/IEC 9899:1999 or ANSI/ISO 9899:1990. If it is compatible there should be no problem to translate the code.

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 sample 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'.

30.1.4 Linker problems

Undefined externals

If your linker shows the error message "Undefined external symbols..." please 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? For details about this please take a look to the chapter 'Low-Level Configuration'.
- One of the files located in the folder Sample\GUI_X? For details about this please take a look to the chapter 'High-Level Configuration'.

Executable to large

Some linkers are not able to link only the modules/functions referenced by the project. This results in 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 please take a look at the chapter 'Getting started'.

30.2 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?

Please make sure there have been configured enough stack. We can not estimate exactly how much stack will be used by your configuration and with your compiler. If you want to

Initialisation of the display wrong?

If the LCD_INIT_CONTROLLER macro is needed by your driver please check, if this macro 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 please 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.

30.3 Problems with API functions

If your tool chain and your hardware works fine but the API functions do not function as documented, please make a small sample as described later under 'Contacting Support'. This allows us to easily reproduce the problem and solve it quickly.

30.4 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 testcode 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 please contact our support, we should be able to optimize the code.

Slow display controller?

Also please take a look to the chapter 'Display drivers'. If using a slow display controller like the Epson SED1335 this chapter may answer the question, why the driver works slow.

30.5 Contacting support

If you need to contact the emWin support, please send the following information to the support:

- A detailed description of the problem may written as comment in the sample code.
- The configuration file `GUIConf.h`.
- The configuration file `LCDConf.h`.
- A sample 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 please 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 please 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. Please also fill in the CPU, the used tool chain and the problem description. It can be found under `Sample\GUI\ProblemReport.c`:

```
*****
*          SEGGER MICROCONTROLLER SYSTEME GmbH
*          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 */
}

```

30.6 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

Numerics

- 3 pin SPI, configuration 725
- 4 pin SPI, configuration 726–727

A

- Access addresses, defining 61
- Access routines, defining 61
- Active window 252
- Additional software 59
- Alias macro 60
- Aliasing 597
- ANSI 49–50, 730
- Antialiasing 597–609
 - of fonts 597–599
 - qualities 597–598
 - software for 597
 - with high-resolution coordinates 597, 599–600
- API reference
 - antialiasing 600
 - BMP file support 136
 - BUTTON widget 313
 - CHECKBOX widget 331
 - colors 208
 - cursors 594
 - device simulator 68, 72
 - dialogs 552
 - EDIT widget 345, 356, 414, 450
 - fonts 146
 - FRAMEWIN widget 374
 - generic touch-screen 580
 - GIF file support 142
 - GRAPH widget 395
 - graphics 114
 - JPEG file API 140
 - kernel interface routines 246
 - LCD driver 691
 - LCD layer 691
 - LISTBOX widget 427
 - memory devices 219
 - MENU widget 469
 - MESSAGEBOX widget 482
 - Multi layer 571

- MULTIEDIT widget 485
- MULTIPAGE widget 495
- PROGBAR widget 506
- RADIO widget 515
- routines common to all widgets 307
- SCROLLBAR widget 525
- SLIDER widget 533
- text 90
- TEXT widget 540
- timing and execution 708
- values 104
- VNC 703
- window manager 265
- WINDOW widget 545
- WM routines used for widgets 306
- Application program interface (API) 52, 690
- Arcs, drawing 131–132
- ASCII 90, 145, 152, 154
- Auto device object 233–237

B

- Background window 254
- Banding memory device 230–233
- Best palette option 191, 193, 195
- Binary switch macro 60
- Binary values, displaying 110–111
- Bitmap converter 51, 187–197
 - command line usage 193–195
 - supported input formats 188
 - using for color conversion 191–192
- Bitmap file support 136
- Bitmaps 187–197
 - color conversion of 191–192
 - device-dependent (DDB) 189
 - device-independent (DIB) 189
 - drawing 119–143
 - full-color mode 189, 191
 - generating "C" files from 187–191
 - manipulating 188
 - RLE compressed 189, 192, 196
- Blocking dialog 548
- BmpCvt.exe 193–195
- Bottom window 253

- Button widget 304, 312–329
BUTTON_3D_MOVE_X 312
BUTTON_3D_MOVE_Y 312
BUTTON_ALIGN_DEFAULT 312
BUTTON_BI_DISABLED 315, 319
BUTTON_BI_PRESSED 315, 319
BUTTON_BI_UNPRESSED 315, 319
BUTTON_BKCOLOR0_DEFAULT 312
BUTTON_BKCOLOR1_DEFAULT 312
BUTTON_CI_DISABLED 316–322
BUTTON_CI_PRESSED 316–322
BUTTON_CI_UNPRESSED 316–322
BUTTON_Create 314
BUTTON_CreateEx 315
BUTTON_CreateIndirect 315
BUTTON_FOCUSCOLOR_DEFAULT 312
BUTTON_FONT_DEFAULT 312
BUTTON_GetBitmap 315
BUTTON_GetBkColor 316
BUTTON_GetDefaultBkColor 316
BUTTON_GetDefaultFont 316
BUTTON_GetDefaultTextAlign 317
BUTTON_GetDefaultTextColor 317
BUTTON_GetFont 317
BUTTON_GetText 317
BUTTON_GetTextColor 318
BUTTON_IsPressed 318
BUTTON.REACT_ON_LEVEL 312
BUTTON_SetBitmap 318
BUTTON_SetBitmapEx 319
BUTTON_SetBkColor 319
BUTTON_SetBMP 320
BUTTON_SetBMPEX 320
BUTTON_SetDefaultBkColor 320
BUTTON_SetDefaultFocusColor 321
BUTTON_SetDefaultFont 321
BUTTON_SetDefault TextAlign 321
BUTTON_SetDefaultTextColor 322
BUTTON_SetFocusColor 322
BUTTON_SetFocussable 322
BUTTON_SetFont 323
BUTTON_SetPressed 323
BUTTON_SetStreamedBitmap 323
BUTTON_SetStreamedBitmapEx 323
BUTTON_SetText 324
BUTTON_SetTextAlign 324
BUTTON_SetTextColor 324
BUTTON_TEXTCOLOR0_DEFAULT 312
BUTTON_TEXTCOLOR1_DEFAULT 312
- C**
- "C" compiler 50, 63, 189
 "C" files
 converting bitmaps into 187–191
 converting fonts into 155
 inclusion of in emWin 59
 "C" programming language 49
 Callback mechanism 51, 253–263
 Callback routines 71, 252
 using 254–263
 Character sets 152–154
 Check box widget 304, 330–343
CHECKBOX_ALIGN_DEFAULT 330
CHECKBOX_BKCOLOR_DEFAULT 330
CHECKBOX_BKCOLOR0_DEFAULT 330
CHECKBOX_BKCOLOR1_DEFAULT 330
- CHECKBOX_Check** 332
CHECKBOX_Create 332
CHECKBOX_CreateEx 332
CHECKBOX_CreateIndirect 332
CHECKBOX_FGCOLOR0_DEFAULT 330
CHECKBOX_FGCOLOR1_DEFAULT 330
CHECKBOX_FOCUSCOLOR_DEFAULT 330
CHECKBOX_FONT_DEFAULT 330
CHECKBOX_GetDefaultBkColor 333
CHECKBOX_GetDefaultFont 333
CHECKBOX_GetDefaultSpacing 334
CHECKBOX_GetDefaultTextAlign 334
CHECKBOX_GetDefaultTextColor 334
CHECKBOX_GetState 334
CHECKBOX_GetText 335
CHECKBOX_IMAGE0_DEFAULT 330
CHECKBOX_IMAGE1_DEFAULT 330
CHECKBOX_IsChecked 335
CHECKBOX_SetBkColor 335
CHECKBOX_SetBoxBkColor 336
CHECKBOX_SetDefaultBkColor 336
CHECKBOX_SetDefaultFocusColor 337
CHECKBOX_SetDefaultFont 337
CHECKBOX_SetDefaultImage 337
CHECKBOX_SetDefaultSpacing 338
CHECKBOX_SetDefaultTextAlign 338
CHECKBOX_SetDefaultTextColor 338
CHECKBOX_SetFocusColor 339
CHECKBOX_SetImage 339
CHECKBOX_SetNumStates 340
CHECKBOX_SetSpacing 340
CHECKBOX_SetState 340
CHECKBOX_SetText 341
CHECKBOX_SetTextAlign 341
CHECKBOX_SetTextColor 342
CHECKBOX_SPACING_DEFAULT 330
CHECKBOX_TEXTCOLOR_DEFAULT 330
CHECKBOX_Uncheck 342
 Child window 252, 271, 548
 Circles, drawing 129–130
 Client area, of windows 252
 Clip area, of windows 252
 Clipping 113, 252
 Color bar test routine 200–201
 Color conversion, of bitmaps 188, 191–192
 Color lookup table (LUT) 208
 Color palettes
 best palette option 191, 193, 195
 custom 192–193, 208
 fixed 191, 201–207
 Colors 199–213
 converting 199
 logical 199
 physical 199
 predefined 200
 COM/SEG lines
 configuration 736–740
 lookup tables for 740
 Command line usage
 of bitmap converter 193–195
 Compile time switches 51
 Compiling, with simulator
 demo program 64, 66
 for your application 66
 samples 65
 Config folder 60, 66, 586, 711
 Configuration, of emWin 60

high-level 743–751
 low-level 711–742
 Control characters 90, 145
 Controls (see Widgets)
 Coordinates 53, 252, 599–600
 high-resolution 597, 599–600
 CPU load, when used as LCD controller 644
 Cursors 593–596
 available styles 594
 Custom palettes
 defining for hardware 208
 file formats, for color conversion 192
 for color conversion 192–193

D

Data types 54
 Decimal values, displaying 104–107
 Demos 51
 Depth coordinate 253
 Desktop coordinates 252
 Desktop window 252
 Device simulation ??– 70
 Device.bmp 68, 71
 Device1.bmp 68, 71
 Device-dependent bitmap (DDB) 189
 Device-independent bitmap (DIB) 189
 Dialog messages 548
 Dialog procedure 548–551
 Dialogs 547–553
 blocking 548
 creating 548–551
 defining behavior of 551
 initialization of 550–551
 messages sent to 548
 non-blocking 548
 Directories, inclusion of 56
 Directory structure
 for emWin 56
 for simulator 66
 for Visual C++ workspace 66
 Display driver 625–698
 Displaying bitmap files 135–143
 Drawing modes 115–116
 Dropdown widget 304, 344–354
DROPDOWN_AddString 345
DROPDOWN_ALIGN_DEFAULT 344
DROPDOWN_BKCOLOR0_DEFAULT 344
DROPDOWN_BKCOLOR1_DEFAULT 344
DROPDOWN_BKCOLOR2_DEFAULT 344
DROPDOWN_CF_AUTOSCROLLBAR 347
DROPDOWN_CF_UP 347
DROPDOWN_Collapse 346
DROPDOWN_Create 346
DROPDOWN_CreateEx 346
DROPDOWN_DecSel 347
DROPDOWN_Expand 347
DROPDOWN_FONT_DEFAULT 344
DROPDOWN_GetNumItems 348
DROPDOWN_GetSel 348
DROPDOWN_IncSel 348
DROPDOWN_InsertString 348
DROPDOWN_KEY_EXPAND 344
DROPDOWN_KEY_SELECT 344
DROPDOWN_SetAutoScroll 349
DROPDOWN_SetBkColor 349, 351, 440
DROPDOWN_SetColor 350

DROPDOWN_SetDefaultColor 350
DROPDOWN_SetDefaultFont 350
DROPDOWN_SetDefaultScrollbarColor 351
DROPDOWN_SetFont 351
DROPDOWN_SetItemSpacing 352
DROPDOWN_SetScrollbarWidth 352
DROPDOWN_SetSel 352
DROPDOWN_SetTextAlign 353
DROPDOWN_SetTextColor 353
DROPDOWN_SetTextHeight 353
DROPDOWN_TEXTCOLOR0_DEFAULT 344
DROPDOWN_TEXTCOLOR1_DEFAULT 344
DROPDOWN_TEXTCOLOR2_DEFAULT 344
DROPROWN_DeleteItem 347

E

Edit widget 304, 355–371
EDIT_AddKey 357, 486, 488–493
EDIT_ALIGN_DEFAULT 355
EDIT_BKCOLOR0_DEFAULT 355
EDIT_BKCOLOR1_DEFAULT 355
EDIT_BORDER_DEFAULT 355
EDIT_Create 357
EDIT_CreateAsChild 357
EDIT_CreateEx 358
EDIT_CreateIndirect 359
EDIT_FONT_DEFAULT 355
EDIT_GetDefaultBkColor 359
EDIT_GetDefaultFont 359
EDIT_GetDefaultTextAlign 359
EDIT_GetDefaultTextColor 360
EDIT_GetFloatValue 360
EDIT_GetNumChars 360
EDIT_GetText 360
EDIT_GetValue 361
EDIT_SetBinMode 361
EDIT_SetBkColor 361
EDIT_SetCursorAtChar 362
EDIT_SetCursorAtPixel 362
EDIT_SetDecMode 362
EDIT_SetDefaultBkColor 363
EDIT_SetDefaultFont 363
EDIT_SetDefaultTextAlign 363
EDIT_SetDefaultTextColor 363
EDIT_SetFloatMode 363
EDIT_SetFloatValue 364
EDIT_SetFont 364
EDIT_SetHexMode 364
EDIT_SetInsertMode 365
EDIT_SetMaxLen 365
EDIT_SetpfAddKeyEx 365
EDIT_SetSel 366
EDIT_SetText 366
EDIT_SetTextAlign 367
EDIT_SetTextColor 367
EDIT_SetTextMode 367
EDIT_SetValue 367
EDIT_TEXTCOLOR0_DEFAULT 355
EDIT_TEXTCOLOR1_DEFAULT 355
EDIT_XOFF 355
 Ellipses, drawing 130–131
 embOS 241
 kernel interface routines for 248
 emWin
 as trial version 64–65
 configuration of 60

data types used (see Data types)
 directory structure for 56
 features of 50–51
 in multitask environments 61, 305
 initialization of 60
 memory requirements 756–757
 performance benchmark 754
 updating to newer versions 56
 Execution model 241–249
 supported types 242

F

Fixed color palettes 191
 Fixed palette modes 201–207
 Flickering of display 215, 296
 Floating-point calculations 113
 Floating-point values, displaying ... 108–110
 Font converter 51, 146, 155, 599, 621
 Font editor 155
 Font files
 linking 146, 155
 naming convention 157
 Fonts 51, 145–185
 adding 155
 antialiased 146, 597–599
 available 146
 converting (see Font converter)
 creating additional 146
 declaring 146, 155
 default 146
 defining 51
 editing 155
 file naming convention 157
 generating "C" files from 155
 included with emWin 51, 145
 monospaced 146, 156, 176
 naming convention 156–157
 proportional 146, 156, 158–175
 scaling 51
 selecting 146–147
 usage of 146
 Foreign Language Support 611–623
 Frame window widget 304, 372
 FRAMEWIN_AddButton 375
 FRAMEWIN_AddCloseButton 376
 FRAMEWIN_AddMaxButton 376
 FRAMEWIN_AddMenu 377, 389
 FRAMEWIN_AddMinButton 378
 FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT ..
 374
 FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT
 374
 FRAMEWIN_BORDER_DEFAULT 374
 FRAMEWIN_CLIENTCOLOR_DEFAULT ... 374
 FRAMEWIN_Create 378
 FRAMEWIN_CreateAsChild 379
 FRAMEWIN_CreateEx 379
 FRAMEWIN_DEFAULT_FONT 374
 FRAMEWIN_FRAMECOLOR_DEFAULT 374
 FRAMEWIN_GetBorderSize 380
 FRAMEWIN_GetDefaultBarColor 380
 FRAMEWIN_GetDefaultBorderColor 381
 FRAMEWIN_GetDefaultClientColor 381
 FRAMEWIN_GetDefaultFont 381
 FRAMEWIN_GetDefaultTextColor ... 381, 387
 FRAMEWIN_GetDefaultTitleSize 382

FRAMEWIN_GetTitleHeight 382
 FRAMEWIN_IBORDER_DEFAULT 374
 FRAMEWIN_IsMaximized 382
 FRAMEWIN_IsMinimized 382
 FRAMEWIN_Maximize 383
 FRAMEWIN_Minimize 383
 FRAMEWIN_Restore 384
 FRAMEWIN_SetActive 384
 FRAMEWIN_SetBarColor 385
 FRAMEWIN_SetBorderSize 386
 FRAMEWIN_SetClientColor 386
 FRAMEWIN_SetDefaultBarColor 386
 FRAMEWIN_SetDefaultBorderSize 387
 FRAMEWIN_SetDefaultClient Color 387
 FRAMEWIN_SetDefaultFont 387
 FRAMEWIN_SetDefaultTitleHeight 388
 FRAMEWIN_SetFont 388
 FRAMEWIN_SetMoveable 388
 FRAMEWIN_SetText 389
 FRAMEWIN_Set.TextAlign 390
 FRAMEWIN_SetTextColor 390
 FRAMEWIN_SetTextColorEx 391
 FRAMEWIN_SetTitleHeight 391
 FRAMEWIN_SetTitleVis 392
 FRAMEWIN_TITLEHEIGHT_DEFAULT 374
 Full bus interface, configuration 730–732
 Full-color mode, of bitmaps 189, 191
 Function replacement macro 60
 Function-level linking 57

G

GIF file support 142
 Graph widget 304–412
 GRAPH_AttachData 396
 GRAPH_AttachScale 397
 GRAPH_CreateEx 397
 GRAPH_DATA_XY_AddPoint 406
 GRAPH_DATA_XY_Create 406
 GRAPH_DATA_XY_SetOffX 407
 GRAPH_DATA_XY_SetPenSize 407
 GRAPH_DATA_YT_AddValue 404
 GRAPH_DATA_YT_Create 404
 GRAPH_DATA_YT_SetOffY 405
 GRAPH_SCALE_Create 408
 GRAPH_SCALE_SetFactor 409
 GRAPH_SCALE_SetFont 409
 GRAPH_SCALE_SetNumDecs 410
 GRAPH_SCALE_SetOff 410
 GRAPH_SCALE_SetPos 411
 GRAPH_SCALE_SetTextColor 411
 GRAPH_SCALE_SetTickDist 412
 GRAPH_SetBorder 398
 GRAPH_SetColor 398
 GRAPH_SetGridFixedX 399
 GRAPH_SetGridOffY 400
 GRAPH_SetGridSpacingX 399
 GRAPH_SetGridSpacingY 399
 GRAPH_SetGridVis 401
 GRAPH_SetLineStyleH 401
 GRAPH_SetLineStyleV 401
 GRAPH_SetUserDraw 403
 GRAPH_SetVSizeX 402
 GRAPH_SetVSizeY 402
 Graphic library 51, 113–134, 707
 Grayscale 191, 199
 GUI configuration 744–746

GUI subdirectories	56, 66
GUI_AA_DisableHiRes	600
GUI_AA_DrawArc	601
GUI_AA_DrawLine	602
GUI_AA_DrawPolyOutline	602
GUI_AA_DrawPolyOutlineEx	603
GUI_AA_EnableHiRes	600
GUI_AA_FillCircle	603
GUI_AA_FillPolygon	604
GUI_AA_GetFactor	601
GUI_AA_SetFactor	601
GUI_ALLOC_ALLOC	749
GUI_ALLOC_FREE	749
GUI_ALLOC_GETMAXSIZE	749
GUI_ALLOC_H2P	749
GUI_ALLOC_SIZE	744
GUI_AUTODEV	233
GUI_AUTODEV_INFO	234
GUI_BITMAP structures	188
GUI_BMP_Draw	136
GUI_BMP_GetXSize	136
GUI_BMP_GetYSize	137
GUI_BMP_Serialize	137
GUI_BMP_SerializeEx	137
GUI_CalcColorDist	211
GUI_CalcVisColorError	211
GUI_Clear	100
GUI_ClearKeyBuffer	591
GUI_ClearRect	117
GUI_Color2Index	211
GUI_Color2VisColor	212
GUI_ColorIsAvailable	212
GUI_COMPILER_SUPPORTS_FP	751
GUI_CONST_STORAGE	750–751
GUI_CreateDialogBox	548, 552
GUI_CURSOR_Hide	594
GUI_CURSOR_Select	595
GUI_CURSOR_SetPosition	595
GUI_CURSOR_Show	595
GUI_DEBUG_LEVEL	744
GUI_DEFAULT_BKCOLOR	744
GUI_DEFAULT_COLOR	744
GUI_DEFAULT_FONT	744
GUI_Delay	707–708
GUI_DispBin	110
GUI_DispBinAt	111
GUI_DispCEOL	100
GUI_DispChar	91
GUI_DispCharAt	92
GUI_DispChars	92
GUI_DispDec	104
GUI_DispDecAt	105
GUI_DispDecMin	105
GUI_DispDecShift	106
GUI_DispDecSpace	106
GUI_DispFloat	108
GUI_DispFloatFix	109
GUI_DispFloatMin	109
GUI_DispHex	111
GUI_DispHexAt	112
GUI_DispNextLine	92
GUI_DispSDec	106
GUI_DispSDecShift	107
GUI_DispSFloatFix	110
GUI_DispSFloatMin	110
GUI_DispString	93
GUI_DispStringAt	93
GUI_DispStringAtCEOL	93
GUI_DispStringHCenterAt	94
GUI_DispStringInRect	94
GUI_DispStringInRectEx	94
GUI_DispStringLen	95
GUI_DrawArc	131
GUI_DrawBitmap	119
GUI_DrawBitmapEx	120
GUI_DrawBitmapExp	120
GUI_DrawBitmapMag	121
GUI_DrawCircle	129
GUI_DrawEllipse	130
GUI_DrawGraph	132
GUI_DrawHLine	121
GUI_DrawLine	122
GUI_DrawLineRel	122
GUI_DrawLineTo	122
GUI_DRAWMODE_XOR	115
GUI_DrawPixel	117
GUI_DrawPoint	117
GUI_DrawPolygon	125
GUI_DrawPolyLine	123
GUI_DrawRect	118
GUI_DrawStreamedBitmap	121
GUI_DrawVLine	123–124
GUI>EditBin	368
GUI>EditDec	368
GUI>EditHex	368
GUI>EditString	369
GUIEndDialog	553
GUIEnlargePolygon	125
GUIExec	708
GUIExec1	708
GUICreateCreatedDialog	552
GUIEexecDialogBox	548, 552
GUIFillCircle	129
GUIFillEllipse	130
GUIFillPolygon	126
GUIFillRect	118
GUIFillRectEx	118
GUIFONT structures	155
GUIFont10_1	159
GUIFont10_ASCII	158
GUIFont10S_1	159
GUIFont10S_ASCII	159
GUIFont13_1	160
GUIFont13_ASCII	160
GUIFont13H_1	161
GUIFont13H_ASCII	160
GUIFont13HB_1	161
GUIFont13HB_ASCII	161
GUIFont16_1	162
GUIFont16_1HK	163
GUIFont16_ASCII	162
GUIFont16_HK	163
GUIFont16B_1	164
GUIFont16B_ASCII	164
GUIFont24_1	166
GUIFont24_ASCII	165
GUIFont24B_1	167
GUIFont24B_ASCII	167
GUIFont32_1	168
GUIFont32_ASCII	168
GUIFont32B_1	170
GUIFont32B_ASCII	169
GUIFont4x6	176
GUIFont6x8	176

GUI_Font6x9	176	GUI_MAXTASK	245, 744
GUI_Font8_1	158	GUI_MEASDEV_ClearRect	238
GUI_Font8_ASCII	158	GUI_MEASDEV_Create	238
GUI_Font8x10_ASCII	177	GUI_MEASDEV_Delete	238
GUI_Font8x12_ASCII	178	GUI_MEASDEV_GetRect	238
GUI_Font8x13_1	178	GUI_MEASDEV_Select	238
GUI_Font8x13_ASCII	178	GUI_MEMDEV_Clear	220
GUI_Font8x15B_1	179	GUI_MEMDEV_CopyFromLCD	220
GUI_Font8x15B_ASCII	179	GUI_MEMDEV_CopyToLCD	220
GUI_Font8x16	179	GUI_MEMDEV_CopyToLCDAA	221
GUI_Font8x16x1x2	181	GUI_MEMDEV_CopyToLCDAT	221
GUI_Font8x16x2x2	181	GUI_MEMDEV_Create	221-222
GUI_Font8x16x3x3	182	GUI_MEMDEV_CreateAuto	233
GUI_Font8x17	180	GUI_MEMDEV_CreateFixed	222
GUI_Font8x18	180	GUI_MEMDEV_Delete	224
GUI_Font8x8	177	GUI_MEMDEV_DeleteAuto	233
GUI_Font8x9	177	GUI_MEMDEV_Draw	230
GUI_FontComic18B_1	165	GUI_MEMDEV_DrawAuto	234
GUI_FontComic18B_ASCII	164	GUI_MEMDEV_GetDataPtr	224
GUI_FontComic24B_1	172	GUI_MEMDEV_GetYSize	225
GUI_FontComic24B_ASCII	171	GUI_MEMDEV_ReduceYSize	225
GUI_FontD24x32	183	GUI_MEMDEV_Select	225
GUI_FontD32	173	GUI_MEMDEV_SetOrg	226
GUI_FontD36x48	183	GUI_MEMDEV_Write	226
GUI_FontD48	173	GUI_MEMDEV_WriteAlpha	226
GUI_FontD48x64	184	GUI_MEMDEV_WriteAlphaAt	226
GUI_FontD60x80	185	GUI_MEMDEV_WriteAt	227
GUI_FontD64	174-175	GUI_MEMDEV_WriteEx	227
GUI_GetBkColor	209	GUI_MEMSET	745
GUI_GetBkColorIndex	209	GUI_MessageBox	482
GUI_GetCharDistX	149	GUI_MOUSE_DRIVER_PS2_Init	579
GUI_GetClientRect	116	GUI_MOUSE_DRIVER_PS2_OnRx	579
GUI_GetColor	209	GUI_MOUSE_GetState	577
GUI_GetColorIndex	210	GUI_MoveRel	124
GUI_GetDispPosX	100	GUI_OS	245, 745
GUI_GetDispPosY	100	GUI_PID_GetState	575
GUI_GetDrawMode	115	GUI_PID_STATE	575
GUI_GetFont	147	GUI_PID_StoreState	575
GUI_GetFontDistY	149	GUI_RestoreContext	133
GUI_GetFontInfo	149	GUI_RotatePolygon	127
GUI_GetFontSizeY	150	GUI_SaveContext	133
GUI_GetKey	592	GUI_SelectLCD	228
GUI_GetLineStyle	123	GUI_SelLayer	571
GUI_GetOrg	562	GUI_SendKeyMsg	591
GUI_SetStringDistX	150	GUI_SetBkColor	210
GUI_GetTextAlign	98	GUI_SetBkColorIndex	210
GUI_GetTextExtend	150	GUI_SetClipRect	134
GUI_GetTime	709	GUI_SetColor	210
GUI_GetVersionString	112	GUI_SetColorIndex	211
GUI_GetYDistOfFont	150	GUI_SetDrawMode	115
GUI_GetYSizeOfFont	150	GUI_SetFont	147
GUI_GIF_Draw	142	GUI_SetLBorder	98
GUI_GIF_GetXSize	143	GUI_SetLineStyle	124
GUI_GIF_GetYSize	143	GUI_SetLUTColor	213
GUI_GotoX	99	GUI_SetLUTColorEx	571
GUI_GotoXY	99	GUI_SetLUTEEntry	213
GUI_GotoY	99	GUI_SetOrg	562
GUI_HMEM	750	GUI_Set.TextAlign	98
GUI_Index2Color	212	GUI_Set.TextMode	97
GUI_Init	60	GUI_SetTextStyle	98
GUI_InitLUT	212	GUI_SIF_CreateFont	147
GUI_InvertRect	119	GUI_SIF_DeleteFont	148
GUI_IsInFont	151	GUI_StoreKey	592
GUI_JPEG_Draw	140	GUI_StoreKeyMsg	590
GUI_JPEG_GetInfo	140	GUI_SUPPORT_LARGE_BITMAPS	745
GUI_MagnifyPolygon	126	GUI_SUPPORT_MEMDEV	745
GUI_MAXBLOCKS	744	GUI_SUPPORT_MOUSE	745

GUI_SUPPORT_ROTATION 745
 GUI_SUPPORT_TOUCH 745
 GUI_SUPPORT_UNICODE 745
 GUI_TEXTMODE_NORMAL 97, 99
 GUI_TEXTMODE_REVERSE 97, 99
 GUI_TEXTMODE_TRANSPARENT 97, 99
 GUI_TEXTMODE_XOR 97, 99
 GUI_TOUCH_AD_BOTTOM 587
 GUI_TOUCH_AD_LEFT 586
 GUI_TOUCH_AD_RIGHT 586
 GUI_TOUCH_AD_TOP 586
 GUI_TOUCH_Calibrate 585
 GUI_TOUCH_Exec 586
 GUI_TOUCH_GetState 580
 GUI_TOUCH_MIRROR_X 586
 GUI_TOUCH_MIRROR_Y 586
 GUI_TOUCH_SetDefaultCalibration 586
 GUI_TOUCH_StoreState 577, 580
 GUI_TOUCH_StoreStateEx 581
 GUI_TOUCH_SWAP_XY 586
 GUI_TOUCH_X_ActivateX 584
 GUI_TOUCH_X_ActivateY 584
 GUI_TOUCH_X_MeasureX 584
 GUI_TOUCH_X_MeasureY 584
 GUI_TOUCH_XSIZE 587
 GUI_TOUCH_YSIZE 587
 GUI_TRIAL_VERSION 745
 GUI_UC_ConvertUC2UTF8 614
 GUI_UC_ConvertUTF82UC 614
 GUI_UC_DispString 616
 GUI_UC_Encode 615
 GUI_UC_GetCharCode 615
 GUI_UC_GetCharSize 615
 GUI_UC_SetEncodeNone 616
 GUI_UC_SetEncodeUTF8 616
 GUI_UNI_PTR 750
 GUI_VNC_AttachToLayer 703
 GUI_VNC_Process 704
 GUI_VNC_X_StartServer 705
 GUI_WaitKey 592
 GUI_WINSUPPORT 745
 GUI_X_Delay 708, 747
 GUI_X_ExecIdle 747
 GUI_X_GetTaskID 247
 GUI_X_GetTime 709, 747
 GUI_X_Init 747
 GUI_X_InitOS 247
 GUI_X_Lock 247
 GUI_X_Log 748
 GUI_X_SIGNAL_EVENT 245
 GUI_X_SignalEvent 247
 GUI_X_Unlock 248
 GUI_X_WAIT_EVENT 246
 GUI_X_WaitEvent 248
 GUIConf.h 146, 218, 743

H

Handle, of a window 252
 Hardkey simulation 71–74
 Header widget 304, 393, 413–425
 HEADER_AddItem 414
 HEADER_Create 415
 HEADER_CreateAttached 415
 HEADER_CreateEx 416, 452
 HEADER_CreateIndirect 416
 HEADER_GetDefaultBkColor 416

HEADER_GetDefaultBorderH 417
 HEADER_GetDefaultBorderV 417
 HEADER_GetDefaultCursor 417
 HEADER_GetDefaultFont 417
 HEADER_GetDefaultTextColor 417
 HEADER_GetHeight 418
 HEADER_GetItemWidth 418
 HEADER_GetNumColumns 418
 HEADER_SetBitmap 418
 HEADER_SetBitmapEx 419
 HEADER_SetBkColor 419
 HEADER_SetBMP 420
 HEADER_SetBMPEx 420
 HEADER_SetDefaultBkColor 420
 HEADER_SetDefaultBorderH 421
 HEADER_SetDefaultBorderV 421
 HEADER_SetDefaultCursor 421
 HEADER_SetDefaultFont 422
 HEADER_SetDefaultTextColor 422
 HEADER_SetFont 422
 HEADER_SetHeight 422
 HEADER_SetItemText 423
 HEADER_SetItemWidth 423
 HEADER_SetStreamedBitmap 424
 HEADER_SetStreamedBitmapEx 424
 HEADER_Set.TextAlign 423
 HEADER_SetTextColor 424
 Hello world program 61
 Hexadecimal values, displaying 111–112
 Hiding windows 253
 High-resolution coordinates .. 597, 599–600

I

I/O pins, connection to . 723, 725–726, 729
 I2C bus interface, configuration728–729
 Initializing emWin 60
 Input devices 573, 589–592
 keyboard ??- 592
 mouse ??- 579
 Input focus 548
 Interrupt service routines 242–244, 586
 Invalidation, of windows 253
 ISO 8859-1 145, 152, 154

J

JPEG file support 139

K

Kernel interface routines 242–244, 246–249
 Keyboard input support ??- 592

L

LCD 639
 caching in memory 51
 configuration of 199, 711–741
 connecting to microcontroller 53–54
 initialization of 60
 magnifying 721
 simulated 68
 without LCD controller 54, 644
 LCD controller
 configuration of 711–742
 configuring additional 737–739
 connected to port/buffer 54
 initialization of 716

memory-mapped	54	LCD_NUM_COM0	655
support for	53, 626	LCD_NUM_CONTROLLERS	741
using CPU as	644	LCD_NUM_DUMMY_READS	685
with LUT hardware	212, 740	LCD_NUM_SEG0	655
LCD driver	64	LCD_OFF 634, 636, 639, 642, 646, 649, 742	
availability/selection	626	LCD_ON 634, 636, 639, 642, 646, 649, 742	
customization of	53	LCD_PHYSCOLORS	720
LCD_BITSPERPIXEL	714	LCD_PHYSCOLORS_IN_RAM	213
LCD_BUSWIDTH	639, 731	LCD_READ_A0 652, 657, 664, 669, 671, 673- 674, 682, 686, 722, 728	
LCD_CACHE . 652, 662, 668, 671, 673, 683, 687, 741		LCD_READ_A1 652, 655, 657, 661, 664, 668-.. 669, 671, 673-674, 677, 683, 687, 722, 728	
LCD_CNF4	639	LCD_READ_MEM	634, 638, 659, 730
LCD_CONTROLLER	628, 714	LCD_READ_REG	635, 639, 659, 730
LCD_DAD	650	LCD_READM_A1	666, 685
LCD_ENABLE_MEM_ACCESS	639, 732	LCD_REVERSE	720
LCD_ENABLE_REG_ACCESS	639, 732	LCD_SERIAL_ID	685
LCD_ENDIAN_BIG	642	LCD_SET_LUT_ENTRY	639, 642, 720
LCD_EXTENDED_WAIT	657	LCD_SUPPORT_CACHECONTROL ... 652, 741	
LCD_FILL_RECT	642-643	LCD_SWAP_BYTE_ORDER	639, 732
LCD_FIRSTCOM ... 652, 655, 664, 677, 736		LCD_SWAP_RB	639, 721
LCD_FIRSTSEG	736	LCD_SWAP_XY	719
LCD_FIRSTSEG0	652, 655, 664, 677	LCD_TIMERINIT0	646, 649, 742
LCD_FIXEDPALETTE	714	LCD_TIMERINIT1	646, 649, 742
LCD_GET_BUSY	677	LCD_USE_BITBLT	639, 660, 741
LCD_GetBitsPerPixel	696	LCD_USE_BITBLT_1BPP	660
LCD_GetBitsPerPixelEx	696	LCD_USE_BITBLT_FILL	660
LCD_GetFixedPalette	696	LCD_USE_PARALLEL_16	685
LCD_GetFixedPaletteEx	696	LCD_USE_SERIAL_3PIN	685
LCD_GetNumColors	697	LCD_VRAM_ADR	642
LCD_GetNumColorsEx	697	LCD_VXSIZE	735
LCD_GetNumDisplays	571	LCD_VYSIZE	735
LCD_GetVXSize	697	LCD_WAIT	677
LCD_GetVXSizeEx	697	LCD_WRITE	725
LCD_GetVYSize	697	LCD_WRITE_A0 ... 652, 655, 657, 661, 664, 668-.. 669, 671, 673-674, 677-678, 681, 683, 687, 722, 726, 728	
LCD_GetVYSizeEx	697	LCD_WRITE_A1 ... 650, 652, 655, 657, 661, 664, .. 666, 668-669, 671, 673-674, 677-.. 678, 681, 683, 687, 722, 726, 728	
LCD_GetXMagEx	698	LCD_WRITE_BUFFER_SIZE	666, 685
LCD_GetXSize	698	LCD_WRITE_MEM	634, 639, 659, 731
LCD_GetXSizeEx	698	LCD_WRITE_REG	635, 639, 659, 731
LCD_GetYMagEx	698	LCD_WITEM	725
LCD_GetYSize	698	LCD_WITEM_A0	666, 685
LCD_GetYSizeEx	698	LCD_WITEM_A1 650, 652, 655, 661, 664, 666, .. 668, 673, 677, 681, 685, 723, 726, 729	
LCD_INIT_CONTROLLER 634-635, 638-639, 642, .. 650, 652, 655, 657, 659, 661, 664, .. 666, 668-669, 671, 673-674, 677-.. 678, 681-682, 685-686, 716		LCD_XMAG	721
LCD_L0_ControlCache	652, 664, 695	LCD_XORG	735-736
LCD_L0_DrawBitMap	692	LCD_XSIZE	714
LCD_L0_DrawHLine	693	LCD_YMAG	721
LCD_L0_DrawPixel	693	LCD_YORG	735-736
LCD_L0_DrawVLine	693	LCD_YSIZE	714
LCD_L0_FillRect	693	LCD0323 driver	661-662
LCD_L0_GetPixelIndex	694	LCD07X1 driver	663-664
LCD_L0_Init	691	LCD1200 driver	665
LCD_L0_Off	692	LCD13701 driver	667-668
LCD_L0_On	692	LCD159A driver	669
LCD_L0_ReInit	692	LCD15E05 driver	670-671
LCD_L0_SetLUTEEntry	695	LCD1611 driver	672-673
LCD_L0_SetPixelIndex	694	LCD161620 driver	674-675
LCD_L0_XorPixel	694	LCD1781 driver ??- 677	
LCD_LASTCOM	736	LCD501 driver	676, 678-679
LCD_LASTSEG	736	LCD6331 driver	680-681
LCD_LIN_SWAP	642		
LCD_LUT_COM	740		
LCD_LUT_SEG	740		
LCD_MAX_LOG_COLORS	720		
LCD_MIRROR_X	718		
LCD_MIRROR_Y	718		

LCD6642X driver	682–683
LCD66750 driver	686–687
LCD667XX driver	684–685
LCDColorOnMono driver	633–634
LCDConf.h	53–54, 60, 208, 625, 711
LCDDummy driver	688
LCDFujitsu driver	635–636
LCDLin driver	637–639
LCDLin32 driver	640–643
LCDMem driver	644–646
LCDMemC driver	647–649
LCDNoritake driver	650
LCDNull driver	689
LCDPage1bpp driver	651–654
LCDPage4bpp driver ??-	655
LCDSLin driver	656–663
LCDTemplate driver ??-	688
LCDVesa driver ??-	657658
LCDXylon driver	659–660
Library, creating	57
Linearization	213
Lines, drawing	121–123
Linking source files	57
List box widget	304, 426–447
LISTBOX_	436
LISTBOX_AddString	428
LISTBOX_BKCOLOR0_DEFAULT	426
LISTBOX_BKCOLOR1_DEFAULT	426
LISTBOX_BKCOLOR2_DEFAULT	426
LISTBOX_Create	428
LISTBOX_CreateAsChild	428
LISTBOX_CreateEx	429
LISTBOX_CreateIndirect	429
LISTBOX_DecSel	430
LISTBOX_DeleteItem	430
LISTBOX_FONT_DEFAULT	426
LISTBOX_GetDefaultBkColor	430
LISTBOX_GetDefaultFont	430
LISTBOX_GetDefaultScrollStepH	431
LISTBOX_GetDefault TextAlign	431
LISTBOX_GetDefaultTextColor	431
LISTBOX_GetFont	431
LISTBOX_GetItemDisabled	432
LISTBOX_GetItemSel	432
LISTBOX_GetItemText	432
LISTBOX_GetMulti	433
LISTBOX_GetNumItems	433
LISTBOX_GetScrollStepH	433
LISTBOX_GetSel	433
LISTBOX_SetTextAlign	434
LISTBOX_IncSel	434
LISTBOX_InsertString	434
LISTBOX_InvalidateItem	434
LISTBOX_OwnerDraw	435
LISTBOX_SetAutoScrollH	435
LISTBOX_SetBkColor	436
LISTBOX_SetDefaultBkColor	436
LISTBOX_SetDefaultFont	437
LISTBOX_SetDefaultScrollStepH	437
LISTBOX_SetDefaultTextColor	437
LISTBOX_SetFont	438
LISTBOX_SetItemDisabled	438
LISTBOX_SetItemSel	438
LISTBOX_SetItemSpacing	439
LISTBOX_SetMulti	439
LISTBOX_SetOwnerDraw	439
LISTBOX_SetScrollbarWidth	441
LISTBOX_SetScrollStepH	441
LISTBOX_SetSel	441
LISTBOX_SetString	441
LISTBOX_Set.TextAlign	442
LISTBOX_SetTextColor	442
LISTBOX_TEXTCOLOR0_DEFAULT	426
LISTBOX_TEXTCOLOR1_DEFAULT	426
LISTBOX_TEXTCOLOR2_DEFAULT	426
Listview widget	304, 448–464
LISTVIEW_	454, 456
LISTVIEW_AddColumn	450
LISTVIEW_AddRow	451
LISTVIEW_Create	451
LISTVIEW_CreateAttached	452
LISTVIEW_CreateIndirect	453
LISTVIEW_DecSel	453
LISTVIEW_DeleteColumn	453
LISTVIEW_DeleteRow	453
LISTVIEW_GetBkColor	455
LISTVIEW_GetFont	455
LISTVIEW_GetHeader	455
LISTVIEW_GetNumColumns	456
LISTVIEW_GetNumRows	456
LISTVIEW_GetSel	456
LISTVIEW_GetTextColor	457
LISTVIEW_IncSel	457
LISTVIEW_SetBkColor	457
LISTVIEW_SetColumnWidth	458
LISTVIEW_SetDefaultBkColor	458
LISTVIEW_SetDefaultFont	458
LISTVIEW_SetDefaultGridColor	459
LISTVIEW_SetDefaultTextColor	459
LISTVIEW_SetFont	459
LISTVIEW_SetGridVis	460
LISTVIEW_SetItemBkColor	460
LISTVIEW_SetItemText	461
LISTVIEW_SetItemTextColor	461
LISTVIEW_SetLBorder	462
LISTVIEW_SetRBorder	462
LISTVIEW_SetRowHeight	462
LISTVIEW_SetSel	463
LISTVIEW_Set.TextAlign	463
LISTVIEW_SetTextColor	464
Lookup table (LUT)	207–208, 212–213, 690, 740

M

Measurement device object	238–239
Memory devices	215–302
auto	233–237
banding	230–233
basic usage of	218
disabling of	218, 296
software for	215
with window manager	296
Memory, reducing consumption of	188, 191
MENU	304
MENU_AddItem	469
MENU_Attach	470
MENU_CreateEx	470
MENU_DeleteItem	471
MENU_DisableItem	472
MENU_EnableItem	472
MENU_GetDefaultBkColor	473
MENU_GetDefaultBorderSize	473
MENU_GetDefaultEffect	473

MENU_GetDefaultFont	474
MENU_GetDefaultTextColor	474
MENU_GetItem	474
MENU_GetItemText	475
MENU_GetNumItems	475
MENU_IF_DISABLED	467
MENU_IF_SEPARATOR	467
MENU_InsertItem	475
MENU_MSG_DATA	466
MENU_ON_INITMENU	466
MENU_ON_ITEMSELECT	466
MENU_SetBkColor	476
MENU_SetBorderSize	477
MENU_SetDefaultBkColor	477
MENU_SetDefaultBorderSize	478
MENU_SetDefaultEffect	478
MENU_SetDefaultFont	478
MENU_SetDefaultTextColor	479
MENU_SetFont	479
MENU_SetItem	480
MENU_SetOwner	480
MENU_SetTextColor	480
Message box widget ??-	481-482-483
MESSAGEBOX_Create	483
Messages, sent by callback routines	256
Monospaced fonts (see Fonts)	
Mouse support ??-	579
MULTIEDIT	304
Multiedit widget	484-493
MULTIEDIT_AddKey	486
MULTIEDIT_Create	486
MULTIEDIT_CreateEx	487
MULTIEDIT_GetPrompt	488
MULTIEDIT_GetText	488
MULTIEDIT_GetTextSize	488
MULTIEDIT_SetAutoScrollH	488
MULTIEDIT_SetAutoScrollV	489
MULTIEDIT_SetBkColor	489
MULTIEDIT_SetBufferSize	490
MULTIEDIT_SetCursorOffset	490
MULTIEDIT_SetFont	490
MULTIEDIT_SetInsertMode	490
MULTIEDIT_SetMaxNumChars	491
MULTIEDIT_SetPasswordMode	491
MULTIEDIT_SetPrompt	491
MULTIEDIT_SetReadOnly	492
MULTIEDIT_SetText	492
MULTIEDIT_SetTextColor	492
MULTIEDIT_SetWrapNone	493
MULTIEDIT_SetWrapWord	493
MULTIPAGE	305
Multipage widget	494-505
MULTIPAGE_AddPage	496
MULTIPAGE_CreateIndirect	497
MULTIPAGE_DeletePage	497
MULTIPAGE_DisablePage	497
MULTIPAGE_EnablePage	498
MULTIPAGE_SetDefaultAlign	498
MULTIPAGE_SetDefaultBkColor	499
MULTIPAGE_SetDefaultFont	500
MULTIPAGE_SetDefaultTextColor	500
MULTIPAGE_SetSelection	500
MULTIPAGE_SetWindow	500
MULTIPAGE_IsPageEnabled	501
MULTIPAGE_SelectPage	501
MULTIPAGE_SetAlign	501
MULTIPAGE_SetBkColor	502
MULTIPAGE_SetDefaultAlign	503
MULTIPAGE_SetDefaultBkColor	503
MULTIPAGE_SetDefaultFont	503
MULTIPAGE_SetDefaultTextColor	503
MULTIPAGE_SetFont	504
MULTIPAGE_SetText	504
MULTIPAGE_SetTextColor	505
Multiple layer support	563-571
Multitask environments	242-245, 305
multiple tasks call emWin	242, 244-245
one task calls emWin	242-243
N	
Non-blocking dialog	548
NORMAL drawing mode	115
Normal text	96
Numerical value macro	60
O	
Optional software	59
P	
Palettes (see Color palettes)	
Parent window	252
Performance	753-758
Pixels	53
Pointer input device support	
data structure for	575
mouse input ??-	579
Polygons, drawing	125-128
PROGBAR_Create	506
PROGBAR_CreateAsChild	507
PROGBAR_CreateEx	507
PROGBAR_CreateIndirect	508
PROGBAR_DEFAULT_BARCOLOR0	506
PROGBAR_DEFAULT_BARCOLOR1	506
PROGBAR_DEFAULT_FONT	506
PROGBAR_DEFAULT_TEXTCOLOR0	506
PROGBAR_DEFAULT_TEXTCOLOR1	506
PROGBAR_SetBarColor	508
PROGBAR_SetFont	508
PROGBAR_SetMinMax	508
PROGBAR_SetText	509
PROGBAR_Set TextAlign	509
PROGBAR_SetTextPos	509
PROGBAR_SetValue	510
Progress bar widget	305, 506-513
Proportional fonts (see Fonts)	
R	
Radio button widget	305, 514-524
RADIO_Create	515
RADIO_CreateEx	516
RADIO_CreateIndirect	516
RADIO_Dec	517
RADIO_SetDefaultFont	517
RADIO_SetDefaultTextColor	517
RADIO_SetText	518
RADIO_SetValue	518
RADIO_Inc	518
RADIO_SetBkColor	519
RADIO_SetDefaultFocusColor	519
RADIO_SetDefaultFont	520
RADIO_SetDefaultImage	520
RADIO_SetDefaultTextColor	520

RADIO_SetFocusColor 521
 RADIO_SetFont 521
 RADIO_SetGroupID 522
 RADIO_SetImage 522
 RADIO_SetText 523
 RADIO_SetTextColor 523
 RADIO_SetValue 524
 Real bus-interface 723
 Redrawing mechanism 305
 resource 68
 Resource file 68, 71
 Resource semaphore 246
 Resource table, for dialogs 548–549
 Resource usage 753–758
 Reverse text 96
 RLE compression, of bitmaps .189, 192, 196

S

S1D13806 controller 639
 Sample programs 51, 61
 Scroll bar widget 305, 525–532
 SCROLLBAR_AddValue 526
 SCROLLBAR_Create 526
 SCROLLBAR_CreateAttached 527
 SCROLLBAR_CreateEx 527
 SCROLLBAR_CreateIndirect 528
 SCROLLBAR_Dec 528
 SCROLLBAR_GetDefaultWidth 528
 SCROLLBAR_GetValue 529
 SCROLLBAR_Inc 529
 SCROLLBAR_SetColor 529
 SCROLLBAR_SetDefaultColor 530
 SCROLLBAR_SetDefaultWidth 530
 SCROLLBAR_SetNumItems 530
 SCROLLBAR_SetPageSize 531
 SCROLLBAR_SetState 531
 SCROLLBAR_SetValue 531
 SCROLLBAR_SetWidth 531
 SED1386 controller 639
 Selection switch macro 60
 Shift JIS ??- 623
 creating fonts 621
 Showing windows 253
 Sibling window 252
 SIM_GUI_CreateLCDInfoWindow() 79
 SIM_GUI_CreateLCDWindow 79
 SIM_GUI_Exit 80
 SIM_GUI_Init 80
 SIM_GUI_SetLCDColorBlack 69
 SIM_GUI_SetLCDColorWhite 69
 SIM_GUI_SetLCDPos 70
 SIM_GUI_SetLCDWindowHook 80
 SIM_GUI_SetMag 70
 SIM_GUI_SetTransColor 70
 SIM_HARDKEY_GetNum 72
 SIM_HARDKEY_GetState 72
 SIM_HARDKEY_SetCallback 73
 SIM_HARDKEY_SetMode 71, 73
 SIM_HARDKEY_SetState 74
 SIM_SetTransColor 68
 Simple bus interface, configuration 722–724
 Simulator 51, 63–81
 directory structure for 66
 usage of with emWin source 66–67
 usage of with emWin trial version .. 64–65
 Single task system 242–243

Slider widget 305, 533–539
 SLIDER_BKCOLOR0_DEFAULT 533
 SLIDER_COLOR0_DEFAULT 533
 SLIDER_Create 534
 SLIDER_CreateEx 534
 SLIDER_CreateIndirect 535
 SLIDER_Dec 535
 SLIDER_FOCUSCOLOR_DEFAULT 533
 SLIDER_GetValue 535
 SLIDER_Inc 535
 SLIDER_SetBkColor 536
 SLIDER_SetDefaultFocusColor 536
 SLIDER_SetFocusColor 537
 SLIDER_SetNumTicks 537
 SLIDER_SetRange 537
 SLIDER_SetValue 538
 SLIDER_SetWidth 538
 Source files, linking 57
 Sprintf 103
 Standard fonts 145
 Subdirectories, of GUI 56
 Superloop 242–243
 Support 759–763
 Syntax, conventions used 53

T

Text
 alignment 98–99
 displaying 89–96
 modes 96–97
 positioning 90, 99–100
 Text widget 305, 540–544
 TEXT_Create 540
 TEXT_CreateAsChild 541
 TEXT_CreateEx 541
 TEXT_CreateIndirect 542
 TEXT_FONT_DEFAULT 540
 TEXT_GetDefaultFont 542
 TEXT_SetBkColor 542
 TEXT_SetDefaultFont 543
 TEXT_SetDefaultTextColor 543
 TEXT_SetFont 543
 TEXT_SetText 543
 TEXT_Set.TextAlign 544
 TEXT_SetTextColor 544
 Tick 707–708
 Time-related functions 707–709
 Toggle behavior, of hardkeys 71, 74
 Top window 253
 TOUCH_X_ActivateY 584
 TOUCH_X_MeasureY 584
 Touch-screens 51
 Transparency 253
 in device simulation 68
 Transparent reversed text 96
 Transparent text 96
 Trial version, of emWin 64–65
 Tutorial 60–61

U

uC/OS 241
 kernel interface routines for 248
 Unicode 145, 154
 API reference 614
 displaying characters in 612
 User drawn widgets 310

UTF-8 strings 613

V

Validation, of windows 253
 Values, displaying 103
 Vectorized symbols 125
 Viewer 51–88
 Virtual display 51
 Virtual screen support 555–562
 Visual C++ 64, 66
 directory structure for 66
 VNC support ??- 705

W

Western Latin character set (see ISO 8859-1)

WIDGET_DRAW_ITEM_FUNC 310
WIDGET_GetDefaultEffect 308
WIDGET_ITEM_DRAW 311
WIDGET_ITEM_DRAW_INFO 141
WIDGET_ITEM_GET_XSIZE 311
WIDGET_ITEM_GET_YSIZE 311
WIDGET_SetDefaultEffect 308
WIDGET_SetDefaultEffect_3D 309
WIDGET_SetDefaultEffect_None 309
WIDGET_SetDefaultEffect_Simple 310
WIDGET_SetEffect 310
 Widgets 51, 303–545, 707
 currently available 304
 defining behavior of 551
 dynamic memory usage 306
 handles of 303, 306
 in dialogs 547
 initialization of 548, 550–551
 member functions of 305
 using 305

Win32, kernel interface routines for 249

Window coordinates 253

Window manager 51, 251–303, 708

Window objects (see Widgets)

Window widget 545

WINDOW_BKCOLOR_DEFAULT 545
WINDOW_SetDefaultBkColor 545

Windows 251–302
 clearing 100–101
 properties of 252
 terms associated with 252–253

WM_Activate 267
WM_AttachWindow 267
WM_AttachWindowAt 268
WM_BringToBottom 268
WM_BringToFront 268
WM_BroadcastMessage 268
WM_CF_ANCHOR_BOTTOM 270
WM_CF_ANCHOR_LEFT 270
WM_CF_ANCHOR_RIGHT 270
WM_CF_ANCHOR_TOP 270
WM_CF_BGND 270
WM_CF_CONST_OUTLINE 270
WM_CF_FGND 270
WM_CF_HASTRANS 270
WM_CF_HIDE 270
WM_CF_LATE_CLIP 270
WM_CF_MEMDEV 270
WM_CF_MEMDEV_ON_REDRAW 271
WM_CF_SHOW 271
WM_CF_STAYONTOP 271

WM_ClrHasTrans 269
WM_CREATE 256–257
WM_CreateWindow 269
WM_CreateWindowAsChild 271
WM_Deactivate 272
WM_DefaultProc 272
WM_DELETE 256–257
WM_DeleteWindow 272
WM_DetachWindow 273
WM_DisableMemdev 296
WM_DisableWindow 273
WM_EnableMemdev 296
WM_EnableWindow 273
WM_Exec 273, 305, 708
WM_Exec1 274
WM_ForEachDesc 274
WM_GET_ID 256–257
WM_GetActiveWindow 275
WM_GetBackgroundWindow 276
WM_GetCallback 275
WM_GetClientRect 276
WM_GetClientRectEx 276
WM_GetClientWindow 297
WM_GetDesktopWindowEx 276–277
WM_GetDialogItem 277
WM_GetFirstChild 277
WM_GetFocussedWindow 277
WM_GetHasTrans 278
WM_GetId 297
WM_GetInsideRect 297
WM_GetInsideRectEx 298
WM_GetInvalidRect 278
WM_GetNextSibling 278
WM_GetOrgX 279
WM_GetOrgY 279
WM_GetParent 279
WM_GetPrevSibling 279
WM_GetScrollPosH 298
WM_GetScrollPosV 298
WM_GetScrollState 298
WM_GetStayOnTop 279
WM_GetUserData 280
WM_GetWindowOrgX 280
WM_GetWindowOrgY 280
WM_GetWindowRect 280
WM_GetWindowRectEx 281
WM_GetWindowSizeX 281
WM_GetWindowSizeY 281
WM_HasCaptured 281
WM_HasFocus 281
WM_HideWindow 282
WM_INIT_DIALOG 256–257, 548, 550
WM_InvalidateArea 282
WM_InvalidateRect 282
WM_InvalidateWindow 283
WM_IsCompletelyVisible 283
WM_IsEnabled 283
WM_IsVisible 284
WM_IsWindow 284
WM_KEY 256–257
WM_MakeModal 284
WM_MENU 466
WM_MESSAGE 256
WM_MOUSEOVER 256, 260
WM_MOVE 256, 258
WM_MoveChildTo 285
WM_MoveTo 285

WM_MoveWindow 285
 WM_NOTIFICATION_CHILD_DELETED ... 257
 WM_NOTIFICATION_CLICKED 313, 330, 344,
 .. 413, 426, 449, 485, 495, 514, 525,
 533
 WM_NOTIFICATION_MOVED_OUT 313, 330,
 344, ... 413, 426, 449, 485, 495, 514
 WM_NOTIFICATION_RELEASED ... 313, 330,
 344, .. 413, 426, 449, 485, 495, 514,
 525, 533
 WM_NOTIFICATION_SCROLL_CHANGED 344,
 426, 449, 485
 WM_NOTIFICATION_SCROLLBAR_ADDED ..
 525
 WM_NOTIFICATION_SEL_CHANGED 426, 449
 WM_NOTIFICATION_VALUE_CHANGED 330,
 355, 485, 495, 514, 525, 533
 WM_NOTIFY_PARENT ... 256, 258, 306, 548
 WM_NOTIFY_VIS_CHANGED 256, 258
 WM_NotifyParent 285
 WM_PAINT 256, 259
 WM_Paint 286, 305
 WM_PaintWindowAndDescs 286
 WM_PID_STATE_CHANGED 256, 260
 WM_ReleaseCapture 286
 WM_ResizeWindow 286
 WM_SelectWindow 287
 WM_SendMessage 287
 WM_SendToParent 287
 WM_SET_FOCUS 256, 259
 WM_SET_ID 256, 259
 WM_SetCallback 288–289
 WM_SetCapture 288
 WM_SetDesktopColor 287–288
 WM_SetDesktopColorEx 288
 WM_SetFocus 289
 WM_SetHasTrans 289
 WM_SetpfPolIPID 290
 WM_SetScrollPosH 299
 WM_SetScrollPosV 299
 WM_SetScrollState 300
 WM_SetSize 290
 WM_SetStayOnTop 291
 WM_SetTransState 292
 WM_SetUserClipRect 292
 WM_SetUserData 293
 WM_SetWindowPos 291
 WM_SetXSize 291
 WM_SetYSize 291
 WM_ShowWindow 293
 WM_SIZE 256, 259
 WM_SUPPORT_NOTIFY_VIS_CHANGED . 264
 WM_SUPPORT_TRANSPARENCY 264
 WM_TOUCH 256, 260
 WM_TOUCH_CHILD 256, 261
 WM_Update 294
 WM_UpdateWindowAndDescs 294
 WM_USER 257, 263
 WM_ValidateRect 294
 WM_ValidateWindow 294

X

X-axis 53
 XOR drawing mode 115
 XOR text 96

Y

Y-axis 53
 Z
 Z-position 253