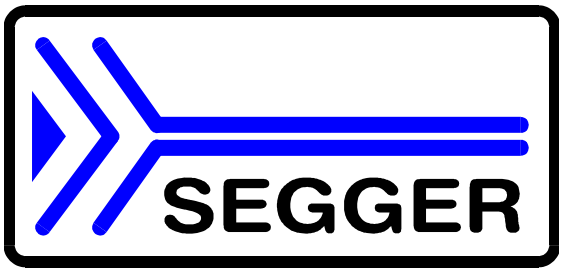
emWin

Graphic Library with Graphical User Interface

Version 5.00

Manual Rev. 1



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.

© 2009 SEGGER Microcontroller GmbH & Co. KG, 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 GmbH & Co. KG
In den Weiden 11
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: 4/9/09

Manual version	Date	By	Explanation
5.00R0	090326	JE	Software has been completely revised. For the version history of earlier versions please refer to older documents.
5.00R1	090409	JE	Chapter 3 'Simulator': - Completely revised. Chapter 8 'Displaying bitmap files' - PNG support added.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

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

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

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

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

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

EMBEDDED SOFTWARE (Middleware)

emWin
Graphics software and GUI



emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display. Starterkits, eval- and trial-versions are available.

embOS
Real Time Operating System



embOS is an RTOS designed to offer the benefits of a complete multitasking system for hard real time applications with minimal resources. The profiling PC tool embOSView is included.

emFile
File system



emFile is an embedded file system with FAT12, FAT16 and FAT32 support. emFile has been optimized for minimum memory consumption in RAM and ROM while maintaining high speed. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.

USB-Stack
USB device stack



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

SEGGER TOOLS

Flasher
Flash programmer

Flash Programming tool primarily for microcontrollers.

J-Link
JTAG emulator for ARM cores

USB driven JTAG interface for ARM cores.

J-Trace
JTAG emulator with trace

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

J-Link / J-Trace Related Software

Add-on software to be used with SEGGER's industry standard JTAG emulator, this includes flash programming software and flash breakpoints.



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Table of Contents

1 Introduction to emWin 17

1.1 Purpose of this document 17
1.2 Assumptions 17
1.3 Requirements 18
1.3.1 Target system (hardware) 18
1.3.2 Development environment (compiler) 18
1.4 Features 18
1.5 Samples and demos 19
1.6 Starter kits 20
1.7 How to use this manual 20
1.8 Typographic conventions for syntax 20
1.9 Screen and coordinates 20
1.10 How to connect the display to the microcontroller 21
1.11 Data types 21

2 Getting Started 23

2.1 Recommended directory structure 24
2.1.1 Subdirectories 24
2.1.2 Include directories 24
2.2 Adding emWin to the target program 24
2.3 Creating a library 25
2.3.1 Adapting the library batch files to a different system 25
2.4 "C" files to include in the project 27
2.5 Configuring emWin 27
2.6 Initializing emWin 28
2.7 Using emWin with target hardware 28
2.8 The "Hello world" sample program 29

3 Simulator 31

3.1 Using the simulator 32
3.1.1 Using the simulator with the trial version of emWin 32
3.1.1.1 Directory structure 32
3.1.1.2 Visual C++ workspace 32
3.1.1.3 Compiling the demo program 33
3.1.1.4 Compiling the samples 33
3.1.2 Using the simulator with the emWin source 34
3.1.2.1 Directory structure 34
3.1.2.2 Visual C++ workspace 34
3.1.2.3 Compiling the application 34
3.1.3 Advanced features of the simulation 35
3.1.3.1 Pause and Resume 35
3.1.3.2 View system info 35
3.1.3.3 Copy to clipboard 35
3.2 Device simulation 36
3.2.1 Generated frame view 37
3.2.2 Custom bitmap view 37
3.2.3 Window view 38
3.3 Device simulator API 39
3.4 Hardkey simulation 43
3.4.1 Hardkey simulator API 44

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

3.5 Integrating the emWin simulation into an existing simulation 47
3.5.1 Directory structure 47
3.5.2 Using the simulation library 47
3.5.2.1 Modifying WinMain 47
3.5.2.2 Sample application 47
3.5.3 Integration into the embOS Simulation 49
3.5.3.1 WinMain 49
3.5.3.2 Target program (main) 49
3.5.4 GUI simulation API 51

4 Viewer 55

4.1 Using the viewer 56
4.1.1 Using the simulator and the viewer 56
4.1.2 Using the viewer with virtual pages 57
4.1.3 Always on top 57
4.1.4 Open further windows of the display output 57
4.1.5 Zooming 57
4.1.6 Copy the output to the clipboard 58
4.1.7 Using the viewer with multiple displays 58
4.1.8 Using the viewer with multiple layers 59

5 Displaying Text 61

5.1 Basic routines 62
5.2 Text API 62
5.3 Routines to display text 63
5.4 Selecting text drawing modes 69
5.5 Selecting text alignment 71
5.6 Setting the current text position 72
5.7 Retrieving the current text position 73
5.8 Routines to clear a window or parts of it 73

6 Displaying Values 75

6.1 Value API 76
6.2 Displaying decimal values 76
6.3 Displaying floating-point values 80
6.4 Displaying binary values 82
6.5 Displaying hexadecimal values 83
6.6 Version of emWin 84

7 2-D Graphic Library 85

7.1 Graphic API 86
7.2 Drawing modes 87
7.3 Query current client rectangle 88
7.4 Pen size 89
7.5 Basic drawing routines 89
7.6 Alpha blending 93
7.7 Drawing bitmaps 94
7.8 Drawing lines 97
7.9 Drawing polygons 100
7.10 Drawing circles 104
7.11 Drawing ellipses 106
7.12 Drawing arcs 107
7.13 Drawing graphs 108
7.14 Drawing pie charts 109
7.15 Saving and restoring the GUI-context 110
7.16 Clipping 110

8 Displaying bitmap files 113

8.1 BMP file support 114

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

8.1.1 Supported formats ... 114
8.1.2 BMP file API ... 114
8.2 JPEG file support ... 119
8.2.1 Supported JPEG compression methods ... 119
8.2.2 Converting a JPEG file to 'C' source ... 119
8.2.3 Displaying JPEG files ... 119
8.2.4 Memory usage ... 119
8.2.5 Progressive JPEG files ... 120
8.2.6 JPEG file API ... 120
8.3 GIF file support ... 124
8.3.1 Converting a GIF file to 'C' source ... 124
8.3.2 Displaying GIF files ... 124
8.3.3 Memory usage ... 124
8.3.4 GIF file API ... 124
8.4 PNG file support ... 132
8.4.1 Converting a PNG file to 'C' source ... 132
8.4.2 Displaying PNG files ... 132
8.4.3 Memory usage ... 132
8.4.4 PNG file API ... 132
8.5 Getting data with the ...Ex() functions ... 135

9 Fonts ... 137

9.1 Introduction ... 138
9.2 Font types ... 138
9.3 Font formats ... 139
9.3.1 'C' file format ... 139
9.3.2 System Independent Font (SIF) format ... 140
9.3.3 External Bitmap Font (XBF) format ... 140
9.3.4 TrueType Font (TTF) format ... 140
9.4 Converting a TTF file to 'C' source ... 141
9.5 Declaring custom fonts ... 141
9.6 Selection of a font ... 142
9.7 Font API ... 142
9.8 'C' file related font functions ... 143
9.9 'SIF' file related font functions ... 143
9.10 'TTF' file related font functions ... 145
9.11 'XBF' file related font functions ... 148
9.12 Common font-related functions ... 149
9.13 Character sets ... 153
9.13.1 ASCII ... 153
9.13.2 ISO 8859-1 Western Latin character set ... 153
9.13.3 Unicode ... 155
9.14 Font converter ... 156
9.14.1 Adding fonts ... 156
9.15 Standard fonts ... 157
9.15.1 Font identifier naming convention ... 157
9.15.2 Font file naming convention ... 158
9.15.3 Measurement, ROM-size and character set of fonts ... 158
9.15.4 Proportional fonts ... 159
9.15.4.1 Overview ... 159
9.15.4.2 Measurement, ROM size and used files ... 160
9.15.4.3 Characters ... 161
9.15.5 Monospaced fonts ... 170
9.15.5.1 Overview ... 170
9.15.5.2 Measurement, ROM size and used files ... 170
9.15.5.3 Characters ... 171
9.15.6 Digit fonts (proportional) ... 175
9.15.6.1 Overview ... 175
9.15.6.2 Measurement, ROM size and used files ... 175
9.15.6.3 Characters ... 175
9.15.7 Digit fonts (monospaced) ... 177

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

9.15.7.1 Overview 177
9.15.7.2 Measurement, ROM size and used files 177
9.15.7.3 Characters 177

10 Bitmap Converter 179

10.1 What it does 180
10.2 Loading a bitmap 180
10.2.1 Supported file formats 180
10.2.2 Loading from a file 180
10.2.3 Using the clipboard 180
10.3 Generating "C" files from bitmaps 181
10.3.1 Supported bitmap formats 181
10.3.2 Palette information 181
10.3.3 Transparency 182
10.3.4 Alpha blending 182
10.3.5 Selecting the best format 183
10.3.6 Saving the file 184
10.4 Color conversion 185
10.5 Compressed bitmaps 186
10.6 Using a custom palette 186
10.6.1 Saving a palette file 187
10.6.2 Palette file format 187
10.6.3 Palette files for fixed palette modes 187
10.6.4 Converting a bitmap 187
10.7 Command line usage 188
10.7.1 Format for commands 188
10.7.2 Valid command line options 188
10.8 Example of a converted bitmap 190

11 Colors 193

11.1 Predefined colors 194
11.2 The color bar test routine 194
11.3 Fixed palette modes 195
11.4 Detailed fixed palette mode description 197
11.5 Application defined color conversion 205
11.6 Custom palette mode 206
11.7 Color API 207
11.8 Basic color functions 207
11.9 Index & color conversion 209

12 Memory Devices 211

12.1 Using memory devices: an illustration 212
12.2 Supported color depth (bpp) 212
12.3 Memory devices and the window manager 213
12.4 Memory requirements 213
12.5 Performance 214
12.6 Basic functions 214
12.7 In order to be able to use memory devices... 214
12.8 Multi layer / multi display configurations 215
12.9 Configuration options 215
12.9.1 GUI_USE_MEMDEV_1BPP_FOR_SCREEN 215
12.10 Memory device API 215
12.11 Basic functions 216
12.12 Banding memory device 229
12.13 Auto device object 231
12.14 Measurement device object 233

13 Execution Model: Single Task / Multitask 235

13.1 Supported execution models 236

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

13.2 Single task system (superloop) ... 236
13.2.1 Description ... 236
13.2.2 Superloop example (without emWin) ... 236
13.2.3 Advantages ... 236
13.2.4 Disadvantages ... 236
13.2.5 Using emWin ... 236
13.2.6 Superloop example (with emWin) ... 237
13.3 Multitask system: one task calling emWin ... 237
13.3.1 Description ... 237
13.3.2 Advantages ... 237
13.3.3 Disadvantages ... 237
13.3.4 Using emWin ... 237
13.4 Multitask system: multiple tasks calling emWin ... 238
13.4.1 Description ... 238
13.4.2 Advantages ... 238
13.4.3 Disadvantages ... 238
13.4.4 Using emWin ... 238
13.4.5 Recommendations ... 238
13.4.6 Example ... 238
13.5 GUI configuration macros for multitasking support ... 239
13.6 Kernel interface routine API ... 240

14 The Window Manager (WM) ... 245

14.1 Explanation of terms ... 246
14.2 Callback mechanism, invalidation and rendering ... 247
14.2.1 Rendering without callbacks ... 247
14.2.2 Rendering using callbacks ... 248
14.2.3 Background window redrawing and callback ... 248
14.2.4 Invalidation ... 249
14.2.5 Rendering of transparent windows ... 249
14.3 Messages ... 250
14.3.1 Message structure ... 250
14.3.2 List of messages ... 250
14.3.3 System-defined messages ... 251
14.3.4 Pointer input device (PID) messages ... 254
14.3.5 System-defined notification codes ... 257
14.3.6 Application-defined messages ... 257
14.4 Configuration options ... 258
14.5 WM API ... 259
14.5.1 Using the WM API functions ... 261
14.6 Basic functions ... 261
14.7 Memory device support (optional) ... 289
14.8 Timer related functions ... 290
14.9 Widget related functions ... 292
14.10 Example ... 296

15 Window Objects (Widgets) ... 299

15.1 Some basics ... 300
15.1.1 Available widgets ... 300
15.1.2 Understanding the redrawing mechanism ... 301
15.1.3 How to use widgets ... 302
15.2 Configuration options ... 303
15.3 General widget API ... 304
15.3.1 API reference: WM routines used for widgets ... 304
15.3.2 API reference: routines common to all widgets ... 304
15.3.3 User drawn widgets ... 307
15.4 BUTTON: Button widget ... 309
15.4.1 Configuration options ... 309
15.4.2 Notification codes ... 310
15.4.3 Keyboard reaction ... 310

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

15.4.4 BUTTON API 310
15.4.5 Examples 322
15.5 CHECKBOX: Checkbox widget 323
15.5.1 Configuration options 323
15.5.2 Notification codes 323
15.5.3 Keyboard reaction 324
15.5.4 CHECKBOX API 324
15.5.5 Examples 336
15.6 DROPDOWN: Dropdown widget 337
15.6.1 Configuration options 337
15.6.2 Notification codes 337
15.6.3 Keyboard reaction 338
15.6.4 DROPDOWN API 338
15.6.5 Examples 349
15.7 EDIT: Edit widget 350
15.7.1 Configuration options 350
15.7.2 Notification codes 350
15.7.3 Keyboard reaction 351
15.7.4 EDIT API 351
15.7.5 Examples 366
15.8 FRAMEWIN: Frame window widget 367
15.8.1 Structure of the frame window 368
15.8.2 Configuration options 369
15.8.3 Keyboard reaction 369
15.8.4 FRAMEWIN API 369
15.8.5 Examples 389
15.9 GRAPH: Graph widget 391
15.9.1 Structure of the graph widget 391
15.9.2 Creating and deleting a graph widget 392
15.9.3 Drawing process 392
15.9.4 Supported types of curves 392
15.9.4.1 GRAPH_DATA_XY 393
15.9.4.2 GRAPH_DATA_YT 393
15.9.5 Configuration options 393
15.9.5.1 Graph widget 393
15.9.5.2 Scale object 393
15.9.6 Keyboard reaction 393
15.9.7 GRAPH API 393
15.9.7.1 Common routines 394
15.9.7.2 GRAPH_DATA_YT related routines 403
15.9.7.3 GRAPH_DATA_XY related routines 406
15.9.7.4 Scale related routines 409
15.9.8 Examples 414
15.10 HEADER: Header widget 415
15.10.1 Configuration options 415
15.10.2 Notification codes 415
15.10.3 Keyboard reaction 416
15.10.4 HEADER API 416
15.10.5 Examples 427
15.11 ICONVIEW: Icon view widget 428
15.11.1 Configuration options 428
15.11.2 Notification codes 429
15.11.3 Keyboard reaction 429
15.11.4 ICONVIEW API 429
15.11.5 Examples 433
15.12 LISTBOX: List box widget 434
15.12.1 Configuration options 434
15.12.2 Notification codes 434
15.12.3 Keyboard reaction 434
15.12.4 LISTBOX API 435
15.12.5 Examples 451

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

15.13 LISTVIEW: Listview widget ... 452
15.13.1 Configuration options ... 453
15.13.2 Notification codes ... 453
15.13.3 Keyboard reaction ... 453
15.13.4 LISTVIEW API ... 454
15.13.5 Example ... 474
15.14 LISTWHEEL: Listwheel widget ... 476
15.14.1 Configuration options ... 476
15.14.2 Notification codes ... 476
15.14.3 Keyboard reaction ... 476
15.14.4 LISTWHEEL API ... 477
15.15 MENU: Menu widget ... 489
15.15.1 Menu messages ... 490
15.15.2 Data structures ... 491
15.15.3 Configuration options ... 492
15.15.4 Keyboard reaction ... 492
15.15.5 MENU API ... 493
15.15.6 Example ... 506
15.16 MESSAGEBOX: Message box widget ... 507
15.16.1 Configuration options ... 507
15.16.2 Keyboard reaction ... 507
15.16.3 MESSAGEBOX API ... 507
15.17 MULTIEDIT: Multi line text widget ... 509
15.17.1 Configuration options ... 510
15.17.2 Notification codes ... 510
15.17.3 Keyboard reaction ... 510
15.17.4 MULTIEDIT API ... 510
15.17.5 Example ... 520
15.18 MULTIPAGE: Multiple page widget ... 521
15.18.1 Configuration options ... 522
15.18.2 Notification codes ... 522
15.18.3 Keyboard reaction ... 522
15.18.4 MULTIPAGE API ... 522
15.18.5 Example ... 533
15.19 PROGBAR: Progress bar widget ... 534
15.19.1 Configuration options ... 534
15.19.2 Keyboard reaction ... 534
15.19.3 PROGBAR API ... 534
15.19.4 Examples ... 538
15.20 RADIO: Radio button widget ... 540
15.20.1 Configuration options ... 540
15.20.2 Notification codes ... 540
15.20.3 Keyboard reaction ... 541
15.20.4 RADIO API ... 541
15.20.5 Examples ... 550
15.21 SCROLLBAR: Scroll bar widget ... 551
15.21.1 Configuration options ... 551
15.21.2 Notification codes ... 551
15.21.3 Keyboard reaction ... 551
15.21.4 SCROLLBAR API ... 552
15.21.5 Examples ... 558
15.22 SLIDER: Slider widget ... 559
15.22.1 Configuration options ... 559
15.22.2 Notification codes ... 559
15.22.3 Keyboard reaction ... 559
15.22.4 SLIDER API ... 559
15.22.5 Examples ... 565
15.23 TEXT: Text widget ... 566
15.23.1 Configuration options ... 566
15.23.2 Keyboard reaction ... 566
15.23.3 TEXT API ... 566

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

15.23.4 Examples 571
15.24 TREEVIEW: Treeview widget 572
15.24.1 Explanation of terms 573
15.24.2 Configuration options 574
15.24.3 Notification codes 574
15.24.4 Keyboard reaction 574
15.24.5 TREEVIEW API 575
15.24.5.1 Common functions 575
15.24.5.2 Item related routines 587
15.24.6 Examples 591
15.25 WINDOW: Window widget 592
15.25.1 Configuration options 592
15.25.2 Keyboard reaction 592
15.25.3 WINDOW API 592

16 Dialogs 593

16.1 Dialog basics 594
16.2 Creating a dialog 594
16.2.1 Resource table 594
16.2.2 Dialog procedure 595
16.2.2.1 Initializing the dialog 596
16.2.2.2 Defining dialog behavior 597
16.3 Dialog API 598
16.4 Dialog boxes 598

17 Virtual screen / Virtual pages 601

17.1 Introduction 602
17.2 Requirements 602
17.3 Configuration 603
17.4 Samples 603
17.4.1 Basic sample 603
17.4.2 Real time sample using the window manager 605
17.4.3 Dialog sample using the window manager 606
17.5 Virtual screen API 608

18 Multi layer / multi display support 609

18.1 Introduction 610
18.1.1 Selecting a layer for drawing operations 610
18.1.2 Selecting a layer for a window 610
18.1.2.1 Moving a window from one layer to an other 611
18.2 Using multi layer support 613
18.2.1 Transparency 613
18.2.2 Alpha blending 614
18.2.3 Hardware cursors 615
18.2.4 Multi layer sample 615
18.3 Using multi display support 615
18.3.1 Enabling multi display support 615
18.3.2 Runtime screen rotation 616
18.3.3 Multi display sample 617
18.4 Configuring multi layer support 617
18.5 Configuring multi display support 618
18.6 Multi layer API 618

19 Pointer Input Devices 623

19.1 Description 624
19.2 Pointer input device API 625
19.3 Mouse driver 627
19.3.1 Generic mouse API 627
19.3.2 PS2 mouse driver 629

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

19.3.2.1 Using the PS2 mouse driver ... 629
19.3.2.2 PS2 mouse driver API ... 629
19.4 Touch-screen drivers ... 630
19.4.1 Generic touch-screen API ... 630
19.4.2 The analog touch screen driver ... 631
19.4.2.1 Setting up the analog touch screen ... 631
19.4.2.2 Runtime calibration ... 633
19.4.2.3 Hardware routines ... 634
19.4.2.4 Driver API for analog touch-screens ... 635
19.4.2.5 Configuring the analog touch-screen driver ... 637
19.5 Joystick input sample ... 638

20 Keyboard Input ... 639

20.1 Description ... 640
20.1.1 Driver layer API ... 640
20.1.2 Application layer API ... 641

21 Sprites ... 643

21.1 Introducton ... 644
21.2 Sprite API ... 644

22 Cursors ... 649

22.1 Available cursors ... 650
22.2 Cursor API ... 650

23 Antialiasing ... 653

23.1 Introduction ... 654
23.1.1 Quality of antialiasing ... 654
23.1.2 Antialiased Fonts ... 654
23.1.3 High-resolution coordinates ... 655
23.2 Antialiasing API ... 656
23.3 Control functions ... 656
23.4 Drawing functions ... 657
23.5 Examples ... 661

24 Foreign Language Support ... 667

24.1 Unicode ... 668
24.1.1 UTF-8 encoding ... 668
24.1.2 Unicode characters ... 668
24.1.3 UTF-8 strings ... 669
24.1.3.1 Using U2C.exe to convert UTF-8 text into "C"-code ... 669
24.1.4 Unicode API ... 670
24.1.4.1 UTF-8 functions ... 670
24.1.4.2 Double byte functions ... 672
24.2 Arabic language support ... 674
24.2.1 Notation forms ... 674
24.2.2 Ligatures ... 675
24.2.3 Bidirectional text alignment ... 675
24.2.4 Requirements ... 676
24.2.5 How to enable Arabic support ... 676
24.2.6 Sample ... 676
24.2.7 Font files used with Arabic text ... 676
24.3 Thai language support ... 677
24.3.1 Requirements ... 677
24.3.2 How to enable Thai support ... 677
24.3.3 Sample ... 677
24.3.4 Font files used with Thai text ... 677
24.4 Shift JIS support ... 678
24.4.1 Creating Shift JIS fonts ... 678

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

24.4.2 Example 678

25 Display drivers 681

25.1 Available drivers and supported display controllers 682
25.2 CPU / Display controller interface 685
25.2.1 Full bus interface....685
25.2.1.1 Typical block diagram 685
25.2.2 Simple bus interface 685
25.2.2.1 Typical block diagram 686
25.2.2.2 Macros used by a simple bus interface 686
25.2.2.3 Sample routines for connection to I/O pins 687
25.2.3 4 pin SPI interface....688
25.2.3.1 Typical block diagram 688
25.2.3.2 Macros used by a 4 pin SPI interface....688
25.2.3.3 Sample routines for connection to I/O pins 689
25.2.4 3 pin SPI interface....689
25.2.4.1 Typical block diagram 689
25.2.4.2 Macros used by a 3 pin SPI interface....689
25.2.4.3 Sample routines for connection to I/O pins 690
25.2.5 I2C bus interface....690
25.2.5.1 Typical block diagram 690
25.2.5.2 Macros used by a I2C bus interface 690
25.2.5.3 Sample routines for connection to I/O pins 690
25.2.6 Non readable displays....691
25.3 Display orientation 692
25.3.1 Runtime configuration....692
25.3.2 Compile time configuration 692
25.3.3 Display orientations....692
25.4 Display driver callback function 694
25.4.1 Commands passed to the callback function 694
25.5 Detailed display driver descriptions....696
25.5.1 GUIDRV_Lin 696
25.5.2 GUIDRV_CompactColor_16....699
25.5.3 LCDTemplate driver....703
25.5.4 LCDNull driver 704
25.6 LCD layer and display driver API 705
25.6.1 Display driver API 705
25.6.2 Callback routines....705
25.6.3 LCD layer routines....706
25.6.3.1 "Get" group....706
25.6.3.2 Configuration group....709

26 VNC server 711

26.1 Introduction 712
26.1.1 Requirements 712
26.1.2 Notes on this implementation 712
26.2 The VNC viewer 713
26.2.1 Starting the VNC viewer....713
26.3 emWin VNC server 714
26.3.1 Starting the emWin VNC server....714
26.3.2 How the server starts ... .714
26.3.3 Integration of the VNC server on the target 714
26.4 Required resources....715
26.5 Configuration options....715
26.6 VNC API....715

27 Timing and Execution-Related Functions 719

27.1 Timing and execution API....720

28 Configuration....723

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

28.1 What needs to be configured? ... 724
28.2 Runtime- and compile time configuration ... 724
28.3 Initialization process of emWin ... 724
28.4 Runtime configuration ... 725
28.4.1 Customizing GUIConf.c ... 725
28.4.1.1 API functions to be used in GUI_X_Config() ... 725
28.4.2 Customizing LCDConf.c ... 726
28.4.2.1 API functions to be used in LCD_X_Config() ... 728
28.4.3 Customizing GUI_X.c ... 729
28.4.3.1 Timing routines ... 729
28.4.3.2 Debugging routines ... 730
28.4.3.3 Kernel interface routines ... 730
28.5 Compile time configuration ... 731
28.5.1 Customizing GUIConf.h ... 731
28.5.1.1 Configuring the available features of emWin ... 731
28.5.1.2 Default font and default color configuration ... 731
28.5.1.3 Advanced GUI configuration options ... 732
28.5.2 Customizing LCDConf.h ... 733

29 Performance and Resource Usage ... 735

29.1 Performance ... 736
29.1.1 Driver benchmark ... 736
29.1.2 Image drawing performance ... 737
29.2 Memory requirements ... 738
29.2.1 Memory requirements of the GUI components ... 738
29.2.2 Stack requirements ... 739
29.3 Memory requirements of sample applications ... 739
29.3.1 "Hello world" project ... 740
29.3.1.1 Configuration ... 740
29.3.1.2 System ... 740
29.3.1.3 Memory requirements ... 740
29.3.2 Window application ... 741
29.3.2.1 Configuration ... 741
29.3.2.2 System ... 741
29.3.2.3 Memory requirements ... 741

30 Support ... 743

30.1 Problems with tool chain (compiler, linker) ... 744
30.1.1 Compiler crash ... 744
30.1.2 Compiler warnings ... 744
30.1.3 Compiler errors ... 744
30.1.4 Linker problems ... 745
30.2 Problems with hardware/driver ... 745
30.3 Problems with API functions ... 746
30.4 Problems with the performance ... 747
30.5 Contacting support ... 747
30.6 FAQ’s ... 748

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 display (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: 600 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: 1200-1800 bytes (depending on the functionality used)
- 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 complying with at least one of the following international standard is required:

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

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

General

- Any (monochrome, grayscale or color) display with any controller supported (if the right driver is available).
- May work without display controller on smaller displays.
- Any interface supported using configuration macros.
- Display-size configurable.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

- Characters and bitmaps may be written at any point on the display, not just on even-numbered byte addresses.
- Routines are optimized for both size and speed.
- Compile time switches allow for different optimizations.
- For slower display controllers, display can be cached in memory, reducing access to a minimum and resulting in very high speed.
- Clear structure.
- Virtual display support; the virtual display can be larger than the actual display.

Graphic library

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

Fonts

- A variety of different fonts are shipped with the basic software: 4*6, 6*8, 6*9, 8*8, 8*9, 8*16, 8*17, 8*18, 24*32, and proportional fonts with pixel-heights of 8, 10, 13, 16. For more information, see chapter 'Fonts'.
- New fonts can be defined and simply linked in.
- Only the fonts used by the application are actually linked to the resulting executable, resulting in minimum ROM usage.
- 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. 50 bytes per window).

Optional widgets for PC look and feel

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

Touch-screen & mouse support

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

PC tools

- Simulation 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

1.6 Starter kits

Complete starter kits including a demoboard with a display, a "C" compiler and a sample project are available. 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:

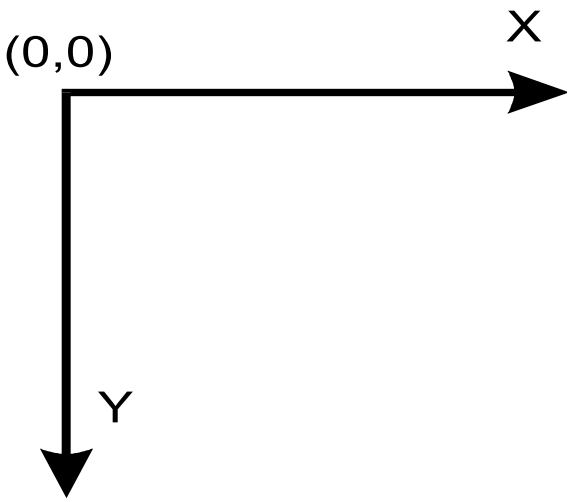
- Copy the emWin files to your computer.
- Go through Chapter 2: "Getting Started".
- Use the simulator in order to become more familiar with what the software can do (refer to Chapter 3: "Simulator).
- Expand your program using the rest of the manual for reference.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

1.10 How to connect the display to the microcontroller

emWin handles all access to the display. Virtually any display controller can be supported, independently of how it is accessed. For details, please refer to Chapter 28: "Configuration". Also, please get in contact with us if your display controller is not supported. We are currently writing drivers for all display controllers available on the market and may already have a proven driver for the display controller that you intend to use. It is usually very simple to write the routines (or macros) used to access the display in your application. SEGGER Microcontroller Systeme GmbH offers the service of making these customizations for you, if necessary with your target hardware.
It does not really matter how the display is connected to the system as long as it is somehow accessible by software, which may be accomplished in a variety of ways. Most of these interfaces are supported by a driver which is supplied in source code form. This driver does not normally require modifications, but is configured for your hardware by making changes in the file `LCDConf.h`. Details about how to customize a driver to your hardware as necessary are explained in Chapter 25: "Display Drivers". The most common ways to access the display are described as follows. If you simply want to understand how to use emWin, you may skip this section.

display with memory-mapped display controller:

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

display with display controller connected to port / buffer

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

Proprietary solutions: display without display controller

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Data type	Definition	Explanation
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 file `Global.h`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

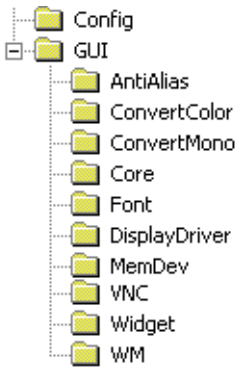
Chapter 2

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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\Font	Font files
GUI\DisplayDriver	Display 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

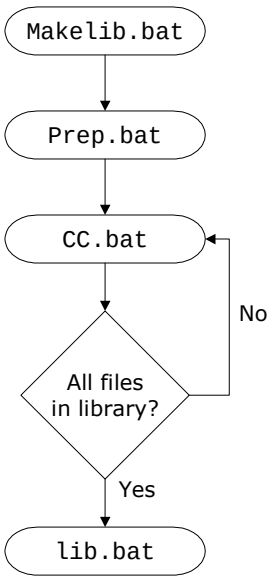
You basically have a choice between including only the source files that you are actually going to use in your project, which will then be compiled and linked, or creating a library and linking the library file. If your tool chain supports "smart" linking (linking in only the modules that are referenced and not those that are not referenced),

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 an other way.



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

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
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
REM ************************************************************
REM   Set the variable PATH to be able to call the tools

SET PATH=%TOOLPATH%\BIN;%TOOLPATH%\LIB308;%PATH%

REM ************************************************************
REM   Set the tool internal used variables

SET BIN308=%TOOLPATH%\BIN
SET INC308=%TOOLPATH%\INC308
SET LIB308=%TOOLPATH%\LIB308
SET TMP308=%TOOLPATH%\TMP
```

Adapting CC.bat

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

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

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
-fFRAM  : Changes the default attribute of RAM data to far
-fETI   : Performs operation after extending char-type data to the int type
          (Extended according to ANSI standards)
:START

REM ************************************************************
REM   Compile the passed source file with the Mitsubishi NC308 compiler

NC308 -silent -M82 -c -IInc -dir Temp\Output -OS -fFRAM -fETI Temp\Source\%1.c

REM ************************************************************
REM   Pause if any problem occurs

IF ERRORLEVEL 1 PAUSE

REM ************************************************************
REM   Add the file name of the object file to the link list

ECHO Temp\Output\%1.R30>>Temp\Output\Lib.dat
```

Adapting Lib.bat

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

```
@ECHO OFF
GOTO START
REM ************************************************************
REM   Explanation of the used options:
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
-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 display driver, all font files you plan to use and any optional modules you have ordered with emWin:

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

Additional software packages

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

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

Target specifics

- For port/buffer-accessed displays, interface routines must be defined. Samples of the required routines are available under `Samples\LCD_X` or on our website in the download area.

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 all configuration files. The chapter ’Configuration’ explains in detail how emWin should be configured.
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 a display.

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

Alias "A"

A macro which operates like a simple text substitute. An example 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 display) which is highly hardware-dependent. This type of macro is always declared using brackets (and optional parameters).

2.6 Initializing emWin

To get emWin initialized the function GUI_Init() should be called. The process of initialization is explained in detail in Chapter 28: "Configuration".

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: Configuring emWin

The first step is usually to customize emWin. For details about the configuration please refer to Chapter 28: "Configuration"

Step 2: Defining access addresses or access routines

For memory-mapped display controllers, the access addresses of the display simply need to be defined in the configuraton file of the display controller. For port/buffer-accessed display controllers, 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: "Configuration".

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 display controller 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:

```
#include "GUI.h"

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

Adding functionality to the "Hello world" program

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

```
#include "GUI.h"

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

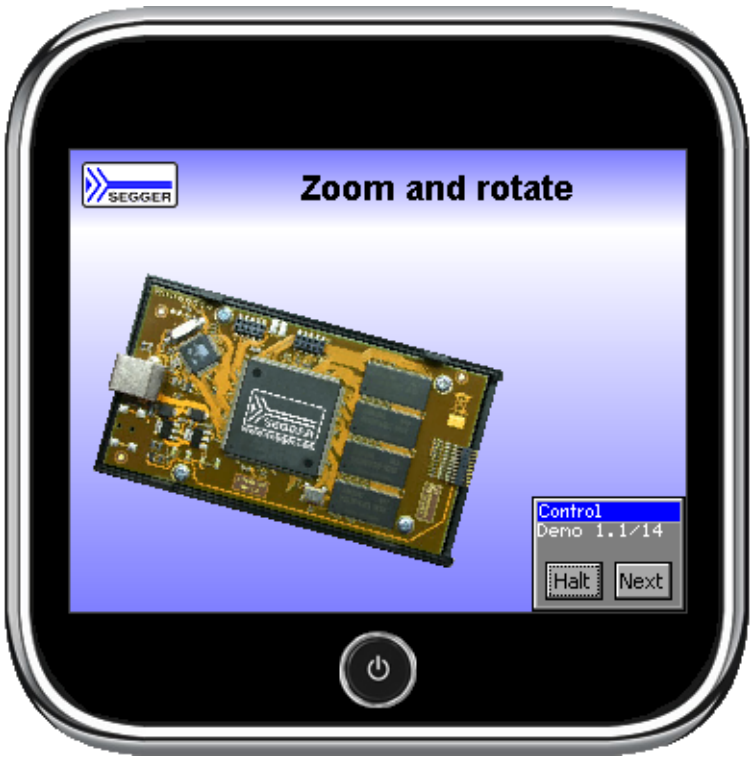
Chapter 3

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.



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.1 Using the simulator

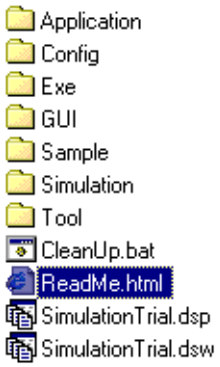
The emWin simulator requires Microsoft Visual C++ (version 6.00 or higher) and the integrated development environment (IDE) which comes with it. You will see a simulation of your LCD on your PC screen, which has the same resolution in X and Y and can display the exact same colors as your LCD once it has been properly configured. The entire graphic library API and window manager API of the simulation are identical to those on your target system; all functions will behave in the very same way as on the target hardware since the simulation uses the same "C" source code as the target system. The difference lies only in the lower level of the software: the 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 with the trial version of emWin

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

3.1.1.1 Directory structure

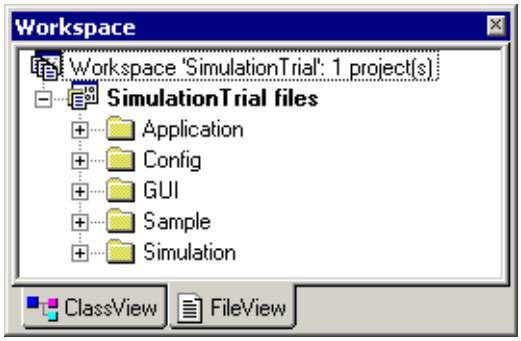
The directory structure of the simulator in the trial version is shown at the right side. The table below explains the contents of the folders:



Directory	Contents
Application	Source of the demo program.
Config	Configuration files used to build the library. Please note that changes at the header files do not have any efect on the precompiled library!
Exe	Ready-to-use demo program.
GUI	Library files and include files needed to use the library.
Sample	Simulation samples.
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.



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

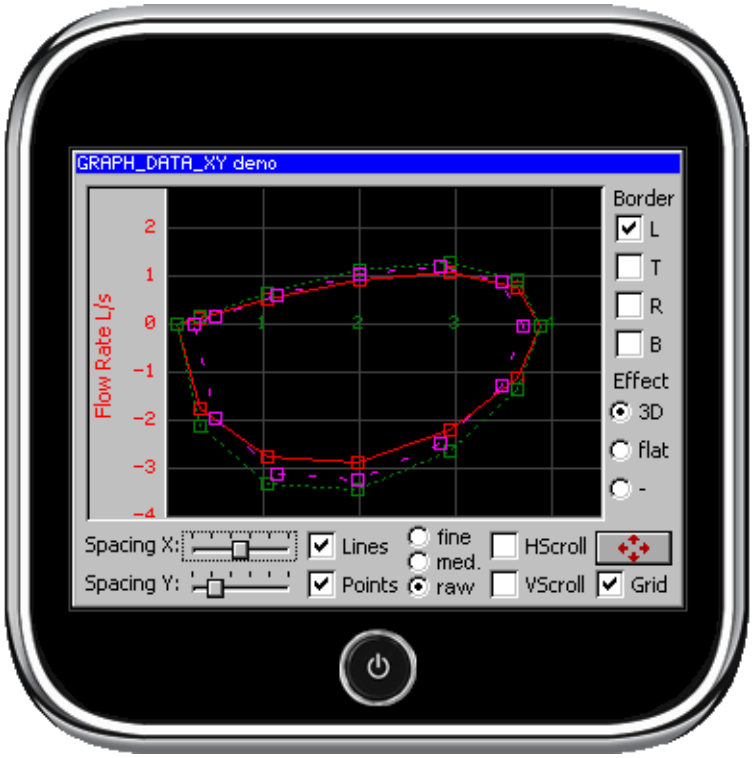
- Step 1: Open the Visual C++ workspace by double-clicking on Simulation-Trial.dsw.
- Step 2: Rebuild the project by choosing Build/Rebuild All from the menu (or by pressing F7).
- Step 3: Start the simulation by choosing Build/Start Debug/Go from the menu (or by pressing F5).

The demo project will begin to run and may be closed 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'.
- Step 3: If the sample contains its own configuration files (LCDConf.c and/or SIM-Conf.c) the default configuration files located in the config folder need to be excluded from the build process.
- Step 4: Rebuild the sample by choosing Build/Rebuild All from the menu (or by pressing F7).
- Step 5: Start the simulation by choosing Build/Start Debug/Go from the menu (or by pressing F5). The result of the sample selected above is pictured below:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

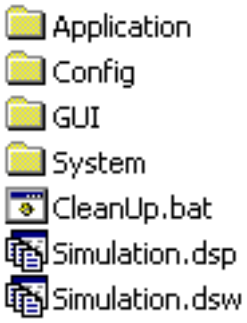
3.1.2 Using the simulator with the emWin source

3.1.2.1 Directory structure



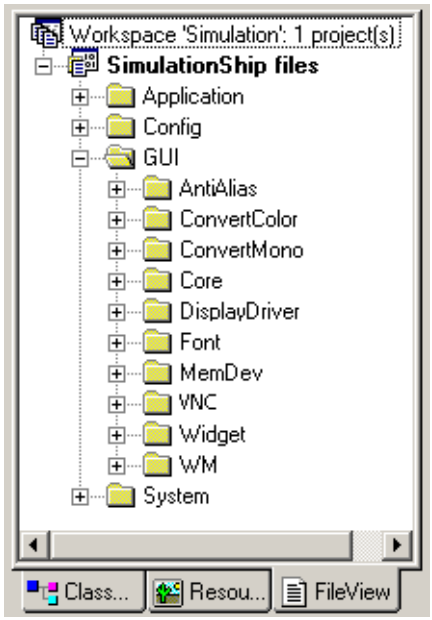
The root directory of the simulator can be anywhere on your PC, e.g. `C:\work\emWinSim`. The directory structure will appear as shown to the right. This structure is very similar to that which we recommend for your target application (see Chapter 3: "Getting Started" for more information).
The following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



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

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

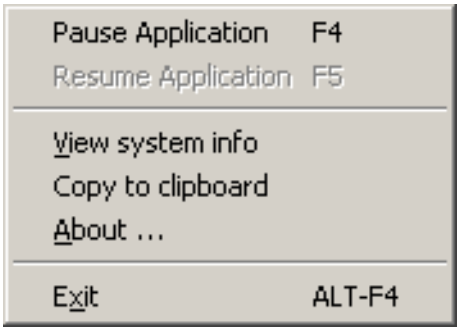
- Step 1: Open the Visual C++ workspace by double-clicking on `Simulation.dsw`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

3.1.3 Advanced features of the simulation

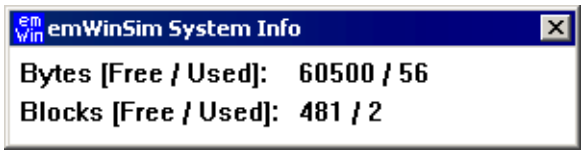
Clicking the right mouse button shows a context menu with several advanced functions:



3.1.3.1 Pause and Resume

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

3.1.3.2 View system info



This menu item opens a further window with information of the memory currently used by the application. The window continuously shows the current status of memory consumption by showing the free and used bytes and the free and used number of memory blocks. For details about the memory manager please refer to the chapter "Configuration" on page 723.

3.1.3.3 Copy to clipboard

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

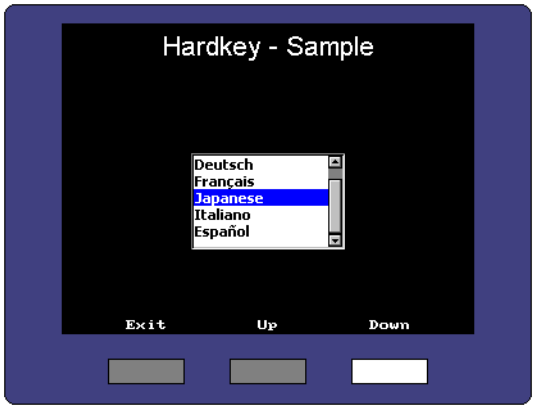
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.2 Device simulation

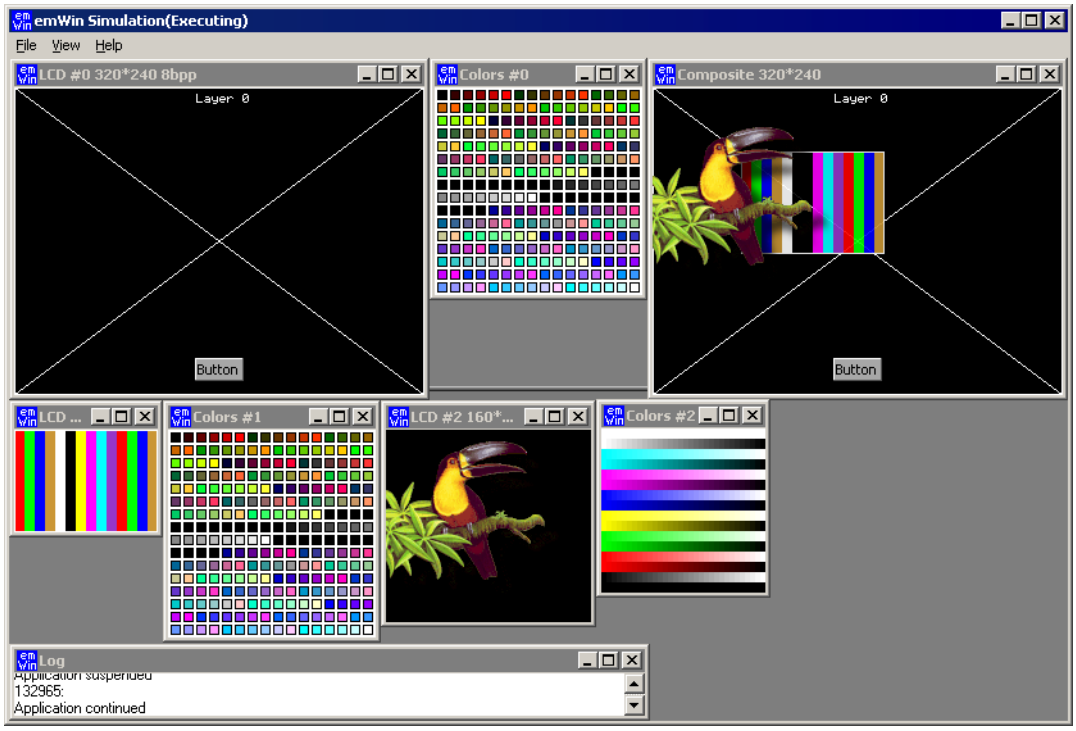
The device simulation supports 3 views:

- Generated frame view
- Custom bitmap view
- Window view

The table below shows the different views:

Generated frame view	Custom bitmap view
	

Window view



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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.2.1 Generated frame view

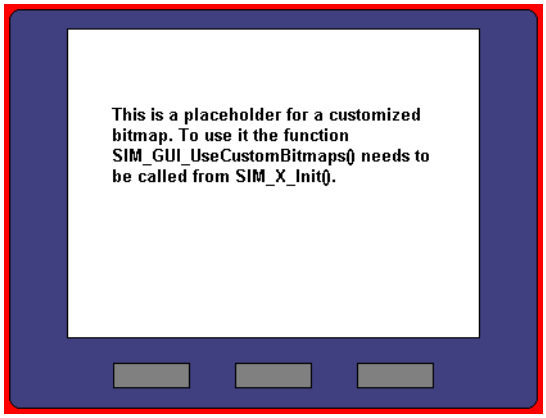


The simulator shows the display inside an automatically generated frame surrounding the display. The frame contains a small button which per default closes the application. This is the default behavior for simulating a single display with one layer. This is the default behavior of the simulation for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

3.2.2 Custom bitmap view

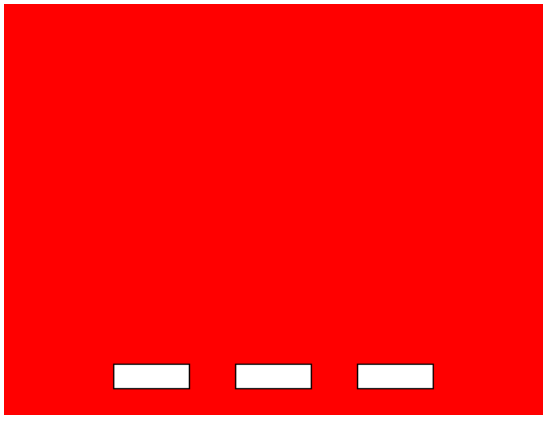
The simulator can show the simulated display in a bitmap of your choice, typically your target device. The bitmap can be used to simulate the behavior of the entire target device. In order to simulate the appearance of the device, bitmaps are required.

Device bitmap



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

Hardkey bitmap



The second bitmap file is required for defining the hardkeys and must be named `Device1.bmp`. It contains the buttons in pressed state. The non hardkey area has to be filled with the transparent color. This is only a short description. For more details about how to simulate hardkeys please refer to the subchapter "Hardkey simulation" on page 43 which should explain in detail how to use hardkeys.

Using separate files

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

Adding the bitmap to the application resources

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

This section can be used to set custom device files. For more information, please refer to the Win32 documentation.

3.2.3 Window view

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.3 Device simulator API

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

```
#include "LCD_SIM.h"

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

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

Routine	Explanation
SIM_GUI_SetCompositeColor()	Sets the background color of the composite window. (Only used with multi layer systems)
SIM_GUI_SetCompositeSize()	Sets the size of the composite window. (Only used with multi layer systems)
SIM_GUI_SetLCDColorBlack()	Set the color to be used as black (color monochrome displays).
SIM_GUI_SetLCDColorWhite()	Set the color to be used as white (color monochrome displays).
SIM_GUI_SetLCDPos()	Set the position for the simulated LCD within the target device bitmap.
SIM_GUI_SetMag()	Set magnification factors for X and/or Y axis.
SIM_GUI_SetTransColor()	Set the color to be used for transparent areas (default: 0xFF0000).
SIM_GUI_UseCustomBitmaps()	Tells the simulation to use the custom bitmaps from the application resource file.

SIM_GUI_SetCompositeColor()

Description

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

Prototype

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

Parameter	Meaning
Color	Background color to be used.

SIM_GUI_SetCompositeSize()

Description

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

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

Parameter	Meaning
xSize	Horizontal size in pixels.
ySize	Vertical size in pixels.

Sample

The following shows a typicall use (with a multi layer system):

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

SIM_GUI_SetLCDColorBlack(), SIM_GUI_SetLCDColorWhite()

Description

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

Prototypes

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

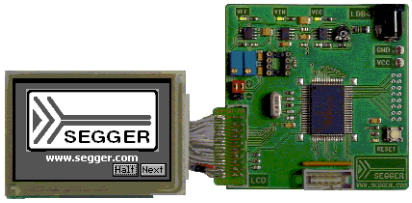
Parameter	Meaning
DisplayIndex	Reserved for future use; must be 0.
Color	RGB value of the color.

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 0xFFFFFF.

Example using default settings

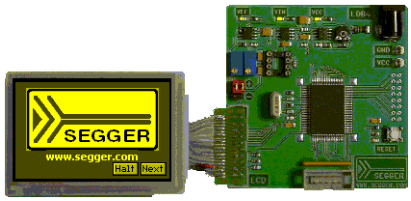
```
void SIM_X_Config() {
  SIM_GUI_SetLCDPos(14,84);                 // Define the position of the LCD
                                            // in the bitmap
  SIM_GUI_SetLCDColorBlack (0, 0x000000);  // Define the color used as black
  SIM_GUI_SetLCDColorWhite (0, 0xFFFFFF);  // Define the color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



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example using yellow instead of white

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



SIM_GUI_SetLCDPos()

Description

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

Prototype

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

Parameter	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).

Add. information

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

SIM_GUI_SetMag()

Description

Sets magnification factors for X and/or Y axis.

Prototype

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

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 00000000RRRRRRRRGGGGGGGGBBBBBBBB.

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.

SIM_GUI_UseCustomBitmaps()

Description

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

Prototype

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

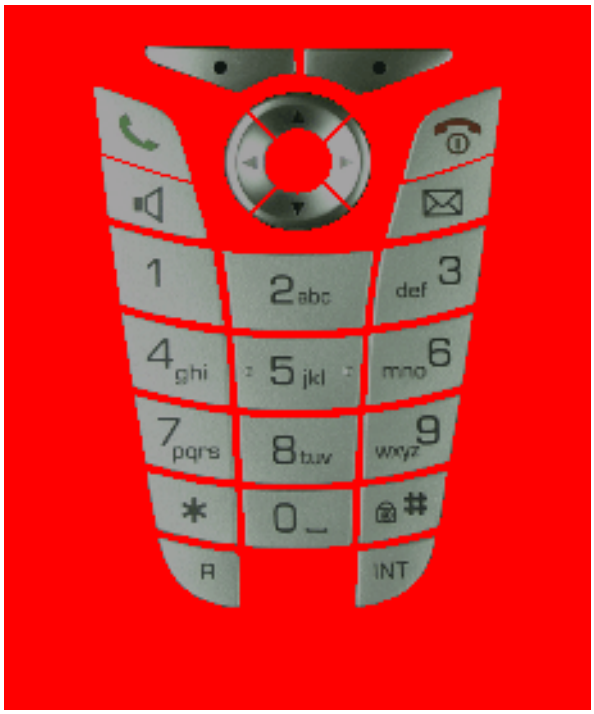
Add. information

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.4 Hardkey simulation

The hardkey simulation can only be used in the custom bitmap view. Hardkeys may also be simulated as part of the device, and may be selected with the mouse pointer. The idea is to be able to distinguish whether a key or button on the simulated device is pressed or unpressed. A hardkey is considered "pressed" as long as the mouse button is held down; releasing the mouse button or moving the pointer off of the hardkey "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). As described earlier in this chapter, this bitmap can be in a separate file in the directory, or included as a resource in the executable. Hardkeys may be any shape, as long as they are exactly the same size in pixels in both `Device.bmp` and `Device1.bmp`. The following example illustrates this:

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

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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.4.1 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
SIM_HARDKEY_GetNum()	Return the number of available hardkeys.
SIM_HARDKEY_GetState()	Return the state of a specified hardkey (0: unpressed, 1: pressed).
SIM_HARDKEY_SetCallback()	Set a callback routine to be executed when the state of a specified hardkey changes.
SIM_HARDKEY_SetMode()	Set the behavior for a specified hardkey (default = 0: no toggle).
SIM_HARDKEY_SetState()	Set the state for a specified hardkey (0: unpressed, 1: pressed).

SIM_HARDKEY_GetNum()

Description

Returns the number of available hardkeys.

Prototype

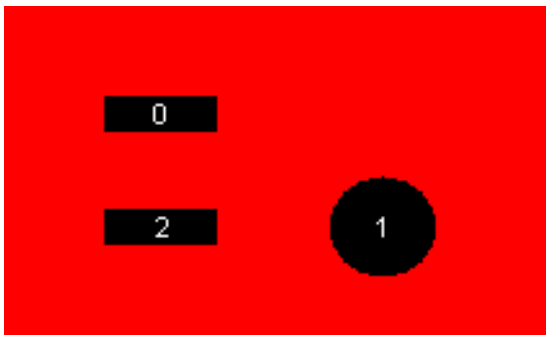
```
int SIM_HARDKEY_GetNum(void);
```

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

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

Prototype

```
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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).

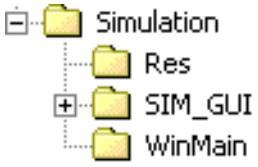
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.5 Integrating the emWin simulation into an existing simulation

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

3.5.1 Directory structure

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



Directory	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.

3.5.2 Using the simulation library

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

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

3.5.2.1 Modifying WinMain

Every windows WIN32 program starts with `WinMain()` (contrary to a normal "C" program from the command line, which starts with `main()`. All that needs to be done is to add a few lines of code to this routine.
The following function calls need to be added (normally in this order as show in the following application code sample):

- SIM_GUI_Init
- SIM_GUI_CreateLCDWindow
- CreateThread
- SIM_GUI_Exit

3.5.2.2 Sample application

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
#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();
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.5.3 Integration into the embOS Simulation

3.5.3.1 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;
}
```

3.5.3.2 Target program (main)

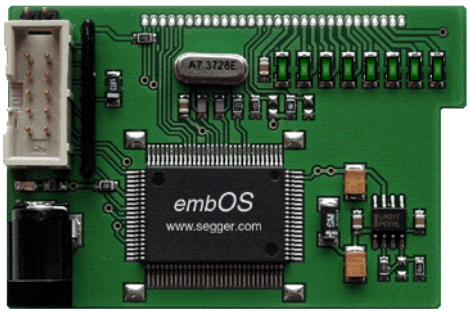
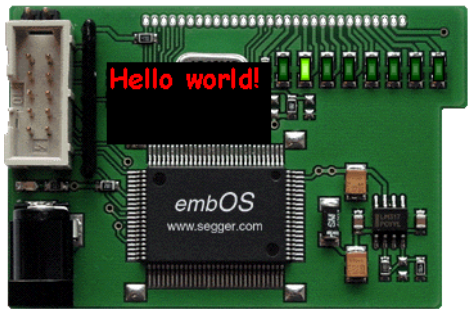
The emWIn API can be called from one or more target threads. Without RTOS, the WIN32 API function `CreateThread` is normally used to create a target thread which calls the emWin API; within an RTOS simulation, a target task/thread (Created by the simulated RTOS) is used to call the emWin API. In other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

Below a modified embOS start application:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
#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
	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

3.5.4 GUI simulation API

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                int x, int y, int xSize, int ySize
                                int LayerIndex);
```

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

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

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
pfHook	Pointer to hook function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 4

Viewer

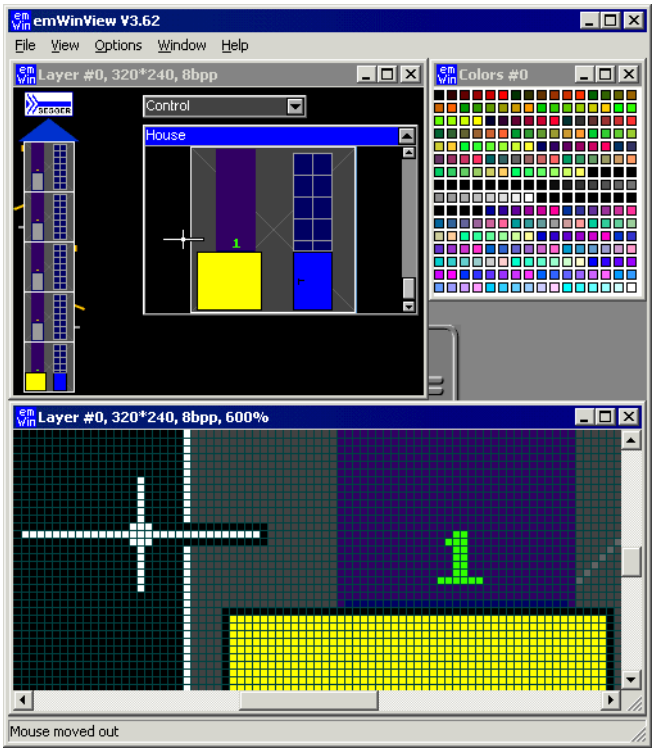
If you use the simulator when debugging your application, you cannot see the display output when stepping through the source code. The primary purpose of the viewer is to solve this problem. It shows the contents of the simulated display(s) while debugging in the simulation.

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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:

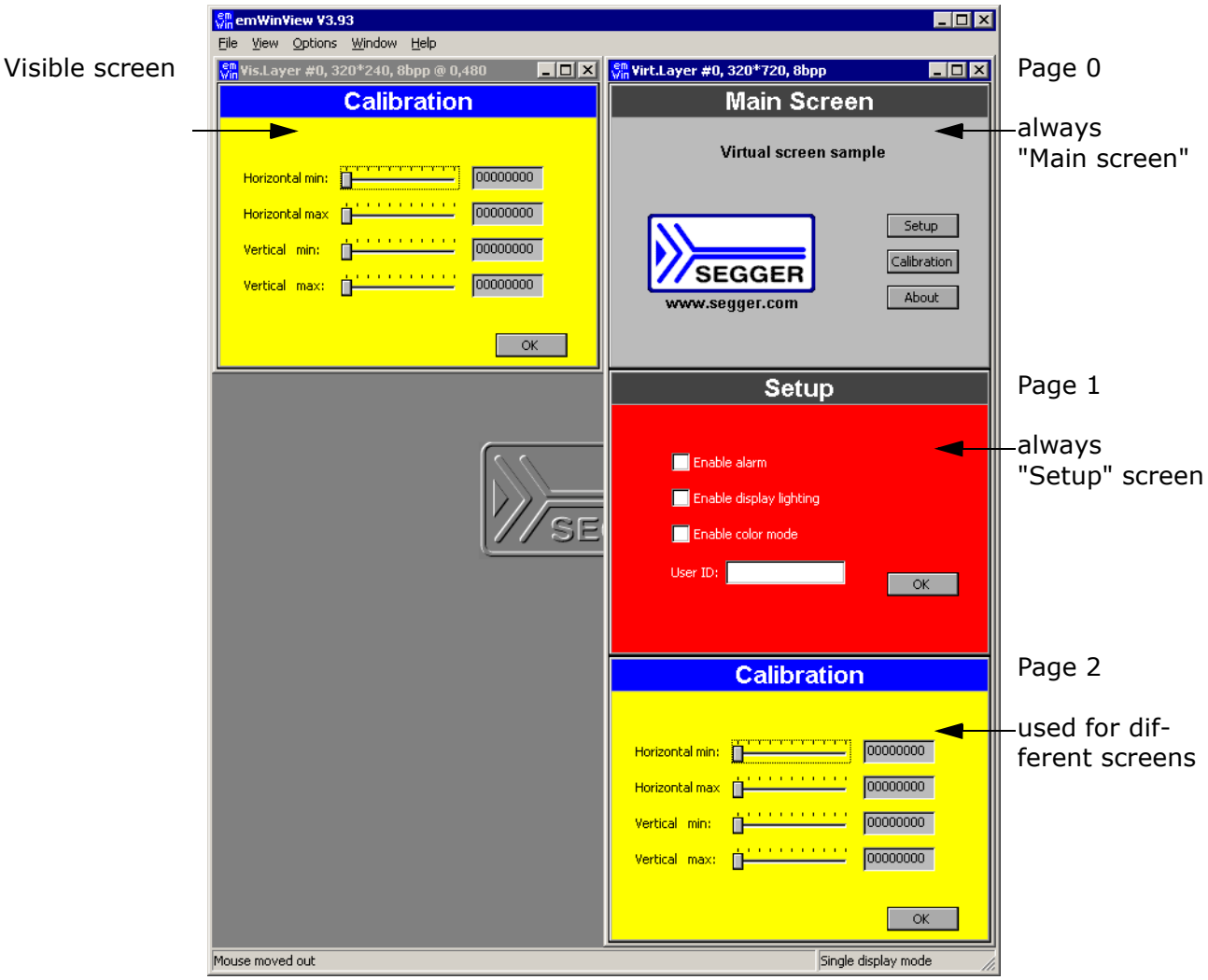
- Step 1: Start the viewer. No display- or color window is shown until the simulation has been started.
- Step 2: Open the Visual C++ workspace.
- Step 3: Compile and run the application program.
- Step 4: Debug the application as described previously.

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

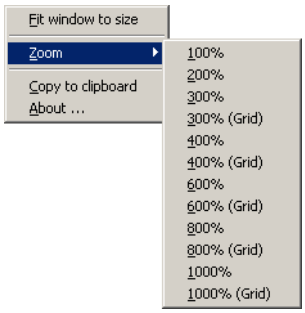
If you want to show a magnified area of the LCD output or the composite view of a multi layer configuration it could be useful to open more than one output window. You can do this by `View/Visible Layer/Layer (1...4)`, `View/Virtual Layer/Layer (1...4)` or `View/Composite`.

4.1.5 Zooming

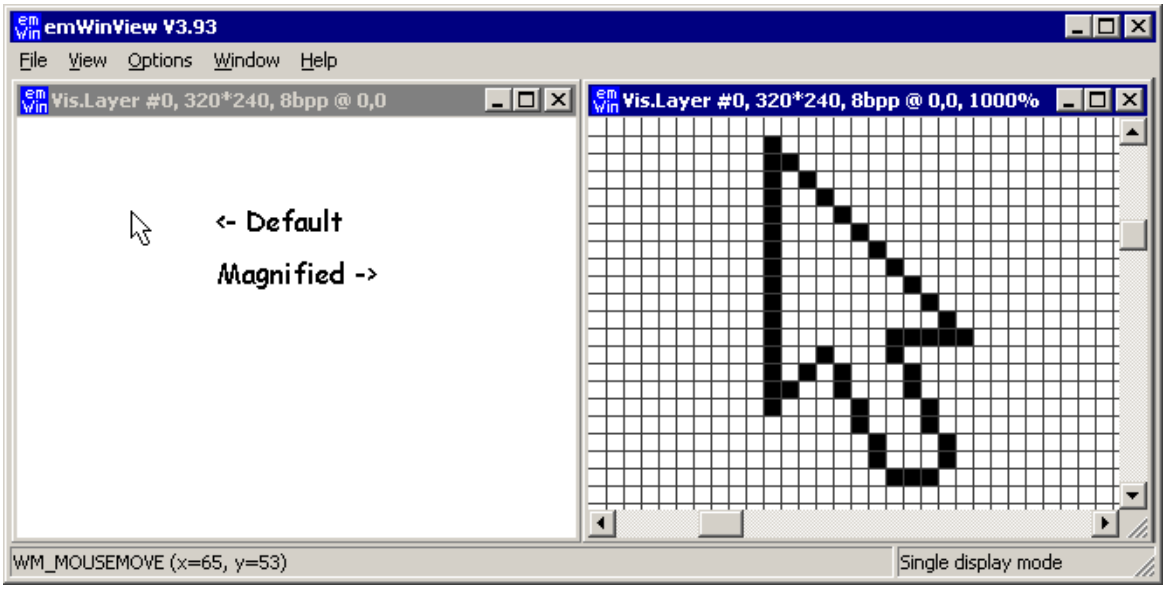
Zooming in or out is easy:
Right-click on a layer or composite window opens the `Zoom` popup menu.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Choose one of the zoom options:



Using the grid



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

Adapting the size of the window

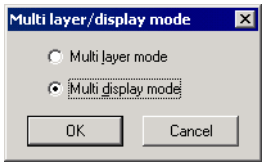
If you want to adapt the size of the window to the magnification choose `Fit window to size` from the first popup menu.

4.1.6 Copy the output to the clipboard

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

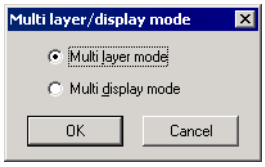


When starting the debugger the viewer will open one display window and one color window for each display:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

4.1.8 Using the viewer with multiple layers

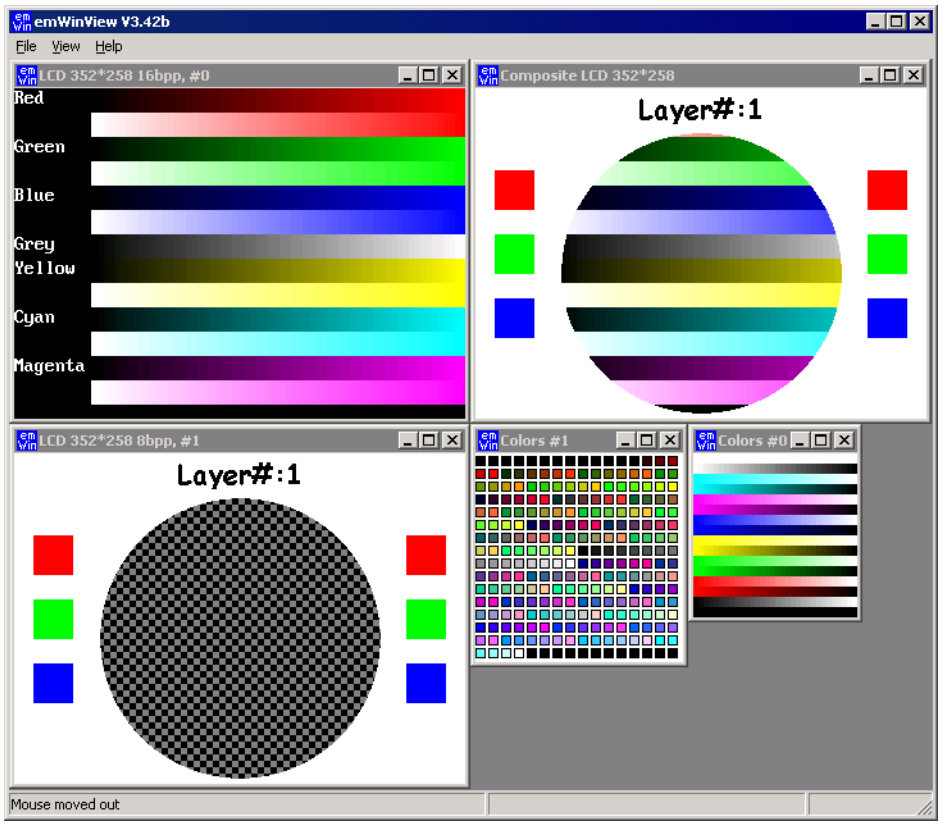
If you are working with multiple layers you should set the viewer into 'Multi layer mode' by using the command `Options/Multi layer/display`.



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.

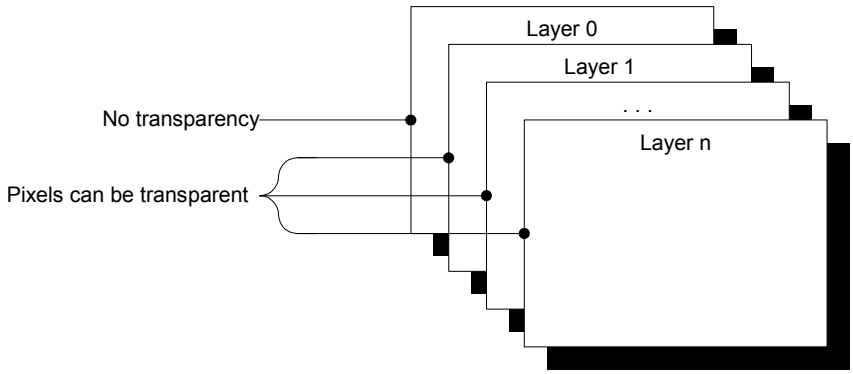
Sample

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



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 GUI_GetFontDistY()).
13	CR	\r	Carriage return. The current text position is changed to the beginning of the current line. Per default, this is: X = 0.

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	
GUI_DispChar()	Displays single character at current position.
GUI_DispCharAt()	Displays single character at specified position.
GUI_DispChars()	Displays character a specified number of times.
GUI_DispNextLine()	Moves the cursor to the beginning of the next line.
GUI_DispString()	Displays string at current position.
GUI_DispStringAt()	Displays string at specified position.
GUI_DispStringAtCEOL()	Displays string at specified position, then clear to end of line.
GUI_DispStringHCenterAt()	Displays string centered horizontally at the given position.
GUI_DispStringInRect()	Displays string in specified rectangle.
GUI_DispStringInRectEx()	Displays string in specified rectangle and optionaly rotates it.
GUI_DispStringInRectWrap()	Displays string in specified rectangle with optional wrapping.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

5.3 Routines to display text

GUI_DispChar()

Description

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

Prototype

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

Parameter	Meaning
c	Character to display.

Add. information

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

Example

Shows a capital A on the display:

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

Related topics

GUI_DispChars(), GUI_DispCharAt()

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_DispCharAt()

Description

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

Prototype

```
void GUI_DispCharAt(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_DispChar().
If the character is not available in the current font, nothing is displayed.

Example

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

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

Related topics

GUI_DispChar(), GUI_DispChars()

GUI_DispChars()

Description

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

Prototype

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

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

Add. information

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

Example

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

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

Related topics

GUI_DispChar(), GUI_DispCharAt()

GUI_DispNextLine()

Description

Moves the cursor to the beginning of the next line.

Prototype

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Related topics

GUI_SetLBorder()

GUI_DispString()

Description

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

Prototype

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

Parameter	Meaning
s	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_DispString("Hello world");// Disp text
GUI_DispString("\nNext line");// Disp text
```

Related topics

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

GUI_DispStringAt()

Description

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

Prototype

```
void GUI_DispStringAt(const char 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.

Example

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

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

Related topics

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

GUI_DispStringAtCEOL()

Description

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_DispStringHCenterAt()

Description

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

Prototype

```
void  GUI_DispStringHCenterAt(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_DispStringInRect()

Description

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

Prototype

```
void GUI_DispStringInRect(const char GUI_FAR *s, 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_DispStringInRect("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_DispString(), GUI_DispStringAtCEOL(), GUI_DispStringLen(),

GUI_DispStringInRectEx()

Description

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

Prototype

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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_DispStringInRectEx(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 configuraton switch GUI_SUPPORT_ROTATION must be activated (default).

GUI_DispStringInRectWrap()

Description

Description

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

Prototype

```
void GUI_DispStringInRectWrap(const char GUI_UNI_PTR * s,
                              GUI_RECT * pRect,
                              int TextAlign,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    GUI_WRAPMODE WrapMode);
```

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.
WrapMode	(see table below)

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

Add. information

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

Example

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

```
int i;
char acText[]      = "This sample demonstrates text wrapping";
GUI_RECT Rect      = {10, 10, 59, 59};
GUI_WRAPMODE aWm[] = {GUI_WRAPMODE_NONE,
                      GUI_WRAPMODE_CHAR,
                      GUI_WRAPMODE_WORD};
GUI_SetTextMode(GUI_TM_TRANS);
for (i = 0; i < 3; i++) {
  GUI_SetColor(GUI_BLUE);
  GUI_FillRectEx(&Rect);
  GUI_SetColor(GUI_WHITE);
  GUI_DispStringInRectWrap(acText, &Rect, GUI_TA_LEFT, aWm[i]);
  Rect.x0 += 60;
  Rect.x1 += 60;
}
```

Screenshot of above example



GUI_DispStringLen()

Description

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

Prototype

```
void GUI_DispStringLen(const char 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispString(), GUI_DispStringAt(), GUI_DispStringAtCEOL(),`

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_REVERSE`.

Example

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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 TextMode (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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 Style	
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_GetTextAlign()

Description

Returns the current text alignment mode.

Prototype

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

GUI_SetLBorder()

Description

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

Prototype

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

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

GUI_SetTextAlign()

Description

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

Prototype

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

Return value

The selected text alignment mode.

Add. information

GUI_SetTextAllign() does not affect the character output routines beginning with GUI_DispChar(). Please note that the settings made with this function are valid only for one string.

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

Shows "(20,20)" at position 20,20 on the display:

```
GUI_GotoXY(20,20)
GUI_DispString("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_DispStringAt("Hello world", 0, 0);// Disp text
GUI_Delay(1000);// Wait 1 second (not part of emWin)
GUI_Clear();// Clear screen
```

GUI_DispCEOL()

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_DispCEOL(void);
```

Example

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
GUI_DispStringAt("Hello world", 0, 0);// Disp text
Delay (1000);
GUI_DispStringAt("Hi", 0, 0);
GUI_DispCEOL();
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

6.2 Displaying decimal values

GUI_DispDec()

Description

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

Prototype

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispString("Min:");
  GUI_DispDec(Min,2);
  GUI_DispString(" Sec:");
  GUI_DispDec(Sec,2);
```

Related topics

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

GUI_DispDecAt()

Description

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

Prototype

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

Parameter	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. 10).

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_DispDecAT(Sec, 200, 0, 2);
```

Related topics

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

GUI_DispDecMin()

Description

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

Prototype

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

Parameter	Meaning
v	Value to display. Minimum: -2147483648 (= -2^31); maximum 2147483647 (= 2^31 -1). Maximum no. of digits displayed is 10.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

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

Related topics

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

GUI_DispDecShift()

Description

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

Prototype

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

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

Add. information

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

GUI_DispDecSpace()

Description

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

Prototype

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

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

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_DispString("The result is :");
GUI_DispDecSpace(Result, 200);
```

Related topics

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

GUI_DispSDec()

Description

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void GUI_DispSDec(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. 10).

Add. information

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

Related topics

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

GUI_DispSDecShift()

Description

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

Prototype

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

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

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_DispStringAt("GUI_DispDecShift:\n",0,0);
  GUI_DispSDecShift(l, 7, 3);
  GUI_SetFont(&GUI_Font6x8);
  GUI_DispStringAt("Press any key",0,GUI_VYSIZE-8);
  WaitKey();
}
```

Screen shot of above example



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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. 10).

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();
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screen shot of above example



GUI_DispFloatFix()

Description

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

Prototype

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

Parameter	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	No. of digits to display (max. 10).
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_DispSFloatFix()

Description

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

Prototype

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

Parameter	Meaning
v	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
Len	No. of digits to display (max. 10).
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_DispSFloatMin()

Description

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

Prototype

```
void GUI_DispSFloatMin(float 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_DispBin()

Description

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

Prototype

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispBinAt()

GUI_DispBinAt()

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_DispBinAt(Input, 0,0, 8);
```

Related topics

GUI_DispBin(), GUI_DispHex()

6.5 Displaying hexadecimal values

GUI_DispHex()

Description

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

Prototype

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

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispString(GUI_GetVersionString());
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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	
GUI_GetDrawMode()	Returns the current drawing mode.
GUI_SetDrawMode()	Sets the drawing mode.
Pen size	
GUI_GetPenSize()	Returns the current pen size in pixels.
GUI_SetPenSize()	Sets the pen size in pixels.
Query current client rectangle	
GUI_GetClientRect()	Returns the current available drawing area.
Basic drawing routines	
GUI_ClearRect()	Fills a rectangular area with the background color.
GUI_DrawGradientV()	Draws a rectangle filled with a vertical color gradient.
GUI_DrawGradientH()	Draws a rectangle filled with a horizontal color gradient.
GUI_DrawPixel()	Draws a single pixel.
GUI_DrawPoint()	Draws a point.
GUI_DrawRect()	Draws a rectangle.
GUI_DrawRectEx()	Draws a rectangle.
GUI_DrawRoundedRect()	Draws a rectangle with rounded corners.
GUI_FillRect()	Draws a filled rectangle.
GUI_FillRectEx()	Draws a filled rectangle.
GUI_FillRoundedRect()	Draws a filled rectangle with rounded corners.
GUI_InvertRect()	Invert a rectangular area.
Alpha blending	
GUI_SetAlpha()	Sets the current alpha blending value.
Drawing bitmaps	
GUI_DrawBitmap()	Draws a bitmap.
GUI_DrawBitmapEx()	Draws a scaled bitmap.
GUI_DrawBitmapHWAlpha()	Draws a bitmap with alpha blending information on a system with hardware alpha blending support.
GUI_DrawBitmapMag()	Draws a magnified bitmap.
Drawing lines	
GUI_DrawHLine()	Draws a horizontal line.
GUI_DrawLine()	Draws a line from a specified startpoint to a specified endpoint (absolute coordinates).
GUI_DrawLineRel()	Draws a line from the current position to an endpoint specified by X- and Y-distances (relative coordinates).
GUI_DrawLineTo()	Draws a line from the current position to a specified endpoint.
GUI_DrawPolyLine()	Draws a polyline.
GUI_DrawVLine()	Draws a vertical line.
GUI_GetLineStyle()	Returns the current line style.
GUI_MoveRel()	Moves the line pointer relative to its current position.
GUI_MoveTo()	Moves the line pointer to the given position.
GUI_SetLineStyle()	Sets the current line style.
Drawing polygons	
GUI_DrawPolygon()	Draws the outline of a polygon.
GUI_EnlargePolygon()	Enlarges a polygon.
GUI_FillPolygon()	Draws a filled polygon.
GUI_MagnifyPolygon()	Magnifys a polygon.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
GUI_RotatePolygon()	Rotates a polygon by a specified angle.
Drawing circles	
GUI_DrawCircle()	Draws the outline of a a circle.
GUI_FillCircle()	Draws a filled circle.
Drawing ellipses	
GUI_DrawEllipse()	Draws the outline of an ellipse.
GUI_FillEllipse()	Draws a filled ellipse.
Drawing arcs	
GUI_DrawArc()	Draws an arc.
Drawing a graph	
GUI_DrawGraph()	Draws a graph.
Drawing a pie chart	
GUI_DrawPie()	Draws a circle sector.
Saving and restoring the GUI-context	
GUI_RestoreContext()	Restores the GUI-context.
GUI_SaveContext()	Saves the GUI-context.
Clipping	
GUI_SetClipRect()	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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_SetDrawMode()

Description

Selects the specified drawing mode.

Prototype

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

Parameter	Meaning
mode	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	
GUI_DM_NORMAL	Default: Draws points, lines, areas, bitmaps.
GUI_DM_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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

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

Parameter	Meaning
pRect	Pointer to GUI_RECT-structure to store result.

7.4 Pen size

The pen size determines the thickness of vector drawing operations line GUI_DrawLine(), GUI_DrawCircle() and so on. Please note that the pen size takes not effect on all drawing functions.

GUI_GetPenSize()

Description

Returns the current pen size.

Prototype

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

GUI_SetPenSize()

Description

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

Prototype

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

Parameter	Meaning
PenSize	Pen size in pixels to be used.

Return value

Previous pen size.

Add information

The pen size should be >= 1.

7.5 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DrawGradientH()

Description

Draws a rectangle filled with a horizontal color gradient.

Prototype

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

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

Example

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

Screenshot of above example



GUI_DrawGradientV()

Description

Draws a rectangle filled with a vertical color gradient.

Prototype

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

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

Screenshot of above example



GUI_DrawPixel()

Description

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

Prototype

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

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DrawRoundedRect()

Description

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

Prototype

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

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

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 rectangle at a specified position in the current window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

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

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

GUI_FillRoundedRect()

Description

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

Prototype

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

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

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()

7.6 Alpha blending

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

GUI_SetAlpha()

Description

Enables software alpha blending for all subsequent drawing operations.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

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

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

Return value

Previous value used for alpha blending.

Add. information

The function sets the alpha value to be used for all subsequent drawing operations. A value of 0 for parameter Alpha means opaque (alpha blending disabled) and a value of 255 means completely transparent (invisible).
Please note that software alpha blending increases the CPU load. Further it is strongly recommended to set the alpha value back to the default value after finishing the drawing operations.

Example

```
extern const GUI_BITMAP _LogoBitmap;

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

Screen shot of above example



7.7 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_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-positiom of the anker point in the bitmap.
yCenter	Y-positiom of the anker point in the bitmap.
xMag	Scale factor of X-direction.
yMag	Scale factor of Y-direction.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DrawBitmapHWAlpha()

Description

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

Prototype

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

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

Add. information

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

GUI_DrawBitmapMag()

Description

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

Prototype

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

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

Add. information

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

7.8 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
y	Y-position.
x0	X-starting position.
x1	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
x0	X-starting position.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Parameter	Meaning
y0	Y-starting position.
x1	X-end position.
y1	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
dx	Distance in X-direction to end of line to draw.
dy	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_DrawVLine()

Description

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

Prototype

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

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

7.9 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. information

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

Example

```
#define GUI_FP_MAXCOUNT 50
```

GUI_MagnifyPolygon()

Description

Magnifies a polygon by a specified factor.

Prototype

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

Parameter	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

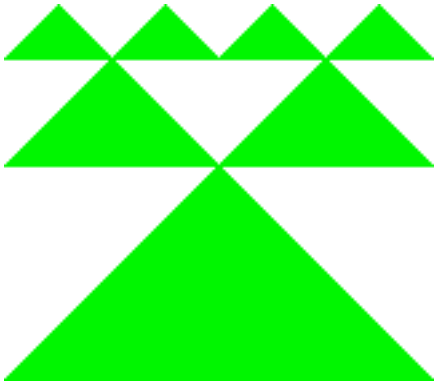
Make sure the destination array of points is equal to or larger than the source array. Note the difference between enlarging and magnifying a polygon. Whereas setting the parameter Len to 1 will enlarge the polygon by one pixel on all sides, setting the parameter Mag to 1 will have no effect.

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;
  }
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

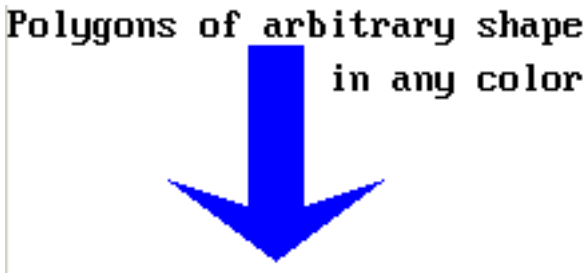
The following example shows how to draw a polygon. It is available as 2DGL_DrawPolygon.c in the samples shipped with emWin.

```
/*********************************************************************
*                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
*
**********************************************************************
*/

static const GUI_POINT aPointArrow[] = {
  {  0,  -5},
  {-40, -35},
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  {-10, -25},
  {-10, -85},
  { 10, -85},
  { 10, -25},
  { 40, -35},
};

/*********************************************************************
*
*               Draws a polygon
*
*********************************************************************
*/

static void DrawPolygon(void) {
  int Cnt =0;
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetFont(&GUI_Font8x16);
  GUI_SetColor(0x0);
  GUI_DispStringAt("Polygons of arbitrary shape ", 0, 0);
  GUI_DispStringAt("in any color", 120, 20);
  GUI_SetColor(GUI_BLUE);
  /* Draw filled polygon */
  GUI_FillPolygon (&aPointArrow[0],7,100,100);
}

/*********************************************************************
*
*                 main
*
*********************************************************************
*/

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

Screen shot of above example



7.10 Drawing circles

GUI_DrawCircle()

Description

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screen shot of above example



7.11 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

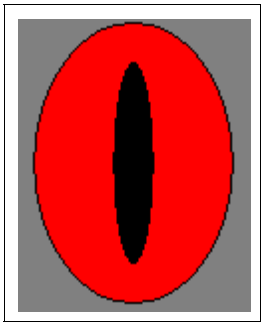
This routine cannot handle a rx/ry parameters in excess of 180.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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



7.12 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.

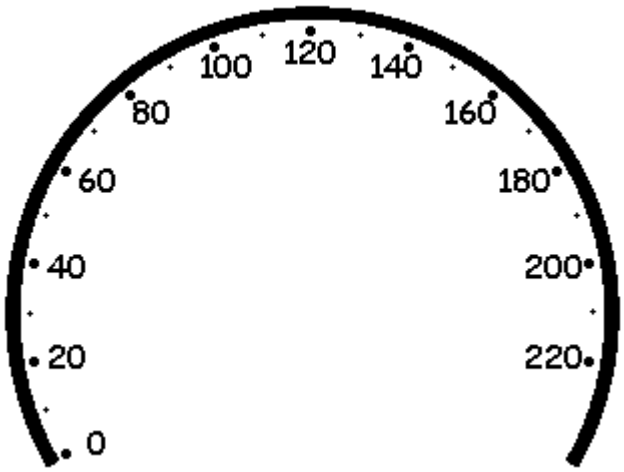
Example

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  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



7.13 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

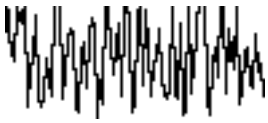
Add. information

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



7.14 Drawing pie charts

GUI_DrawPie()

Description

Draws a circle sector.

Prototype

```
void GUI_DrawPie(int x0, int y0, int r, int a0, int a1, int Type);
```

Parameter	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).
a0	Starting angle (degrees).
a1	End angle (degrees).
Type	(reserved for future use, should be 0)

Example

```
int i, a0, a1;
const unsigned aValues[] = {100, 135, 190, 240, 340, 360};
const GUI_COLOR aColors[] = { GUI_BLUE, GUI_GREEN, GUI_RED,
                              GUI_CYAN, GUI_MAGENTA, GUI_YELLOW };
for (i = 0; i < GUI_COUNTOF(aValues); i++) {
  a0 = (i == 0) ? 0 : aValues[i - 1];
  a1 = aValues[i];
  GUI_SetColor(aColors[i]);
  GUI_DrawPie(100, 100, 50, a0, a1, 0);
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screen shot of above example



7.15 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
pContext	Pointer to a GUI_CONTEXT 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 usefull to save the current state ant to restore it later. For this you can use these functions.

GUI_SaveContext()

Description

The function saves the current GUI-context. (See also GUI_RestoreContext)

Prototype

```
void GUI_SaveContext(GUI_CONTEXT* pContext);
```

Parameter	Meaning
pContext	Pointer to a GUI_CONTEXT structure for saving the current context.

7.16 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 usefull to use a smaller clipping rectangle, which can be set using this function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 8

Displaying bitmap files

The recommended and most efficient way to display a bitmap known at compile time is to use the bitmap converter to convert it into a "C" file and add it to the project / makefile. For details about the bitmap converter please refer to chapter 10, "Bitmap converter".
If the application needs to display images not known at compile time, the image needs to be available in a graphic file format supported by emWin. In this case, the image file can reside in memory or on an other storage device; it can be displayed even if the amount of availalble memory is less than the size of the image file.
emWin currently supports BMP, JPEG, GIF and PNG file formats.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

8.1 BMP file support

Although bitmaps which can be used with emWin are normally compiled and linked as "C" files with the application, there may be situations when using these types of structures is not desirable. A typical example would be an application that continuously references new images, such as bitmaps downloaded by the user. The following functions support .bmp files which have been loaded into memory.
For images that you plan to re-use (i.e. a company logo) it is much more efficient to compile and link it as "C" file which can be used directly by emWin. This may be easily done with the Bitmap Converter.

8.1.1 Supported formats

The BMP file format has been defined by Microsoft. There are a number of different formats as shown in the table below:

Bits per pixel	Indexed	Compression	Supported
1	yes	no	yes
4	yes	no	yes
4	yes	yes	yes
8	yes	no	yes
8	yes	yes	yes
16	no	no	yes
24	no	no	yes
32	no	no	yes

8.1.2 BMP file API

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

Routine	Explanation
GUI_BMP_Draw()	Draws a BMP file which has been loaded into memory.
GUI_BMP_DrawEx()	Draws a BMP file which needs not to be loaded into memory.
GUI_BMP_DrawScaled()	Draws a BMP file with scaling which has been loaded into memory.
GUI_BMP_DrawScaledEx()	Draws a BMP file with scaling which needs not to be loaded into memory.
GUI_BMP_GetXSize()	Returns the X-size of a BMP file loaded into memory.
GUI_BMP_GetXSizeEx()	Returns the X-size of a BMP file which needs not to be loaded into memory.
GUI_BMP_GetYSize()	Returns the Y-size of a bitmap loaded into memory.
GUI_BMP_GetYSizeEx()	Returns the Y-size of a BMP file which needs not to be loaded into memory.
GUI_BMP_Serialize()	Creates a BMP-file.
GUI_BMP_SerializeEx()	Creates a BMP-file from the given rectangle.

GUI_BMP_Draw()

Description

Draws a Windows .bmp file, which has been loaded into memory, at a specified position in the current window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_BMP_Draw(const void * pFileData, int x0, int y0);
```

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

Add. information

The table at the beginning of the chapter shows the supported BMP file formats. The sample 2DGL_DrawBMP.c shows how to use the function.

GUI_BMP_DrawEx()

Description

Draws a .bmp file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_BMP_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Return value

Zero on success, nonzero if the function fails.

Add. information

This function is used for drawing .bmp files if not enough RAM is available to load the whole file into memory. The GUI library then calls the function pointed by the parameter pfGetData to read the data. The 'GetData' function needs to return the number of requested bytes. The maximum number of bytes requested by the GUI is the number of bytes needed for drawing one line of the image.

GUI_BMP_DrawScaled()

Description

Draws a .bmp file, which has been loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int GUI_BMP_DrawScaled(const void * pFileData,
                       int x0, int y0, int Num, int Denom);
```

Parameter	Meaning
pFileData	Pointer to the start of the memory area in which the .bmp file resides.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Zero on success, nonzero if the function fails.

Add. information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrinked to 2/3 of size the parameter Num should be 2 and Denom should be 3.

GUI_BMP_DrawScaledEx()

Description

Draws a .bmp file, which does not have to be loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int  GUI_BMP_DrawScaledEx(GUI_BMP_GET_DATA_FUNC * pfGetData,
                          void * p, int x0, int y0,
                        int Num, int Denom);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Return value

Zero on success, nonzero if the function fails.

Add. information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrinked to 2/3 of size the parameter Num should be 2 and Denom should be 3.
For more details please refer to the function GUI_BMP_DrawEx() explained earlier in this chapter.

GUI_BMP_GetXSize()

Description

Returns the X-size of a specified bitmap which has been loaded into memory.

Prototype

```
int GUI_BMP_GetXSize(const void * pFileData);
```

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

Return value

X-size of the bitmap.

GUI_BMP_GetXSizeEx()

Description

Returns the X-size of a specified .bmp file which does not have to be loaded into memory.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_BMP_GetXSizeEx(GUI_BMP_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called by the GUI for getting data.
p	Void pointer passed to the function pointed by pfGetData.

Return value

X-size of the bitmap.

GUI_BMP_GetYSize()

Description

Returns the Y-size of a specified bitmap which has been loaded into memory.

Prototype

```
int GUI_BMP_GetYSize(const void * pFileData);;
```

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

Return value

Y-size of the bitmap.

GUI_BMP_GetYSizeEx()

Description

Returns the Y-size of a specified .bmp file which does not have to be loaded into memory.

Prototype

```
int GUI_BMP_GetYSizeEx(GUI_BMP_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called by the GUI for getting data.
p	Void pointer passed to the function pointed by pfGetData.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

8.2 JPEG file support

JPEG (pronounced "jay-peg") is a standardized compression method for full-color and gray-scale images. JPEG is intended for compressing "real-world" scenes; line drawings, cartoons and other non-realistic images are not its strong suit. JPEG is lossy, meaning that the output image is not exactly identical to the input image. Hence you must not use JPEG if you have to have identical output bits. However, on typical photographic images, very good compression levels can be obtained with no visible change, and remarkably high compression levels are possible if you can tolerate a low-quality image.

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. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.)
The JPEG file support does not contain provision for the hierarchical or lossless processes defined in the standard.

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` subfolder. It converts the given binary file (in this case the JPEG file) to a 'C' file. The filename of the 'C' file is the same as the binary file name with the file extension '.c'. The following steps will show how to embed a JPEG file using `Bin2C`:

- Start `Bin2C.exe` and select the JPEG file to be converted to a 'C' file, for example 'Image.jpeg' and convert it to a 'C' file.
- Add the 'C' file to the project.

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 refer to the memory device chapter.

8.2.4 Memory usage

The JPEG decompression uses app. 33Kb RAM for decompression independent of the image size and a size dependent amount of bytes. The RAM requirement can be calculated as follows:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

App. RAM requirement = X-Size of image * 80 bytes + 33 Kbytes

The X-size dependent amount depends on the compression type of the JPEG file. The following table shows some samples:

Compression	Size of image in pixels	RAM usage [Kbyte]	RAM usage, size dependent [Kbyte]
H1V1	160x120	45	12
H2V2	160x120	46	13
GRAY	160x120	38	4

The memory required for the decompression is allocated dynamically by the emWin memory management system. After drawing the JPEG image the complete RAM will be released.

8.2.5 Progressive JPEG files

Contrary to baseline and extended-sequential JPEG files progressive JPEGs consist of multiple scans. Each of these scans is based on the previous scan(s) and refines the appearance of the JPEG image. This requires scanning the whole file even if only one line needs to be decompressed.
If enough RAM is configured for the whole image data, the decompression needs only be done one time. If less RAM is configured, the JPEG decoder uses 'banding' for drawing the image. The more bands required the more times the image needs to be decompressed and the slower the performance. With other words: The more RAM the better the performance.

8.2.6 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_DrawEx()	Draws a JPEG file which needs not to be loaded into memory.
GUI_JPEG_DrawScaled()	Draws a JPEG file with scaling which has been loaded into memory.
GUI_JPEG_DrawScaledEx()	Draws a JPEG file with scaling which needs not to be loaded into memory.
GUI_JPEG_GetInfo()	Fills a GUI_JPEG_INFO structure from a JPEG file which has been loaded into memory.
GUI_JPEG_GetInfoEx()	Fills a GUI_JPEG_INFO structure from a JPEG file which needs not to be loaded into memory.

GUI_JPEG_Draw()

Description

Draws a .jpeg file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_JPEG_Draw(const void * pFileData, int DataSize, int x0, int y0);
```

Parameter	Meaning
pFileData	Pointer to the start of the memory area in which the .jpeg file resides.
DataSize	Number of bytes of the .jpeg file.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

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

Add. information

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

GUI_JPEG_DrawEx()

Description

Draws a `.jpeg` file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_JPEG_DrawEx(GUI_JPEG_GET_DATA_FUNC * pfGetData,
                    void * p, int x0, int y0);
```

Parameter	Meaning
`pfGetData`	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
`p`	Void pointer passed to the function pointed by `pfGetData`.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.

Return value

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

Add. information

This function is used for drawing `.jpegs` if not enough RAM is available to load the whole file into memory. The JPEG library then calls the function pointed by the parameter `pfGetData` to read the data.
The 'GetData' function should return the number of available bytes. This could be less or equal the number of requested bytes. The function needs at least to return 1 new byte. The sample folder contains the sample `2DGL_DrawJPGScaled.c` which shows how to use a 'GetData' function.

GUI_JPEG_DrawScaled()

Description

Draws a `.jpeg` file, which has been loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int  GUI_JPEG_DrawScaled(const void * pFileData, int DataSize,
                         int x0, int y0, int Num, int Denom);
```

Parameter	Meaning
`pFileData`	Pointer to the start of the memory area in which the .jpeg file resides.
`DataSize`	Number of bytes of the .jpeg file.
`x0`	X-position of the upper left corner of the bitmap in the display.
`y0`	Y-position of the upper left corner of the bitmap in the display.
`Num`	Numerator to be used for scaling.
`Denom`	Denominator used for scaling.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

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

Add. information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrinked to 2/3 of size the parameter Num should be 2 and Denom should be 3.
The sample folder contains the sample 2DGL_DrawJPGScaled.c which shows how to draw scaled JPEGs.

GUI_JPEG_DrawScaledEx()

Description

Draws a .jpeg file, which does not have to be loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int  GUI_JPEG_DrawScaledEx(GUI_JPEG_GET_DATA_FUNC * pfGetData, void * p,
                           int x0, int y0, int Num, int Denom);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Return value

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

Add. information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrinked to 2/3 of size the parameter Num should be 2 and Denom should be 3.
For more details please refer to the function GUI_JPEG_DrawEx() explained earlier in this chapter.
The sample folder contains the sample 2DGL_DrawJPGScaled.c which shows how to use the function.

GUI_JPEG_GetInfo()

Description

Fills a GUI_JPEG_INFO structure with information about a .jpeg file, which has been loaded into memory.

Prototype

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

Parameter	Meaning
pFileData	Pointer to the start of the memory area in which the .jpeg file resides.
DataSize	Number of bytes of the .jpeg file.
pInfo	Pointer to a GUI_JPEG_INFO structure to be filled by the function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Zero on success, nonzero if the function fails.

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.

GUI_JPEG_GetInfoEx()

Description

Fills a GUI_JPEG_INFO structure with information about a .jpeg file, which does not have to be loaded into memory.

Prototype

```
int GUI_JPEG_GetInfoEx(GUI_JPEG_GET_DATA_FUNC * pfGetData, void * p,
                       GUI_JPEG_INFO * pInfo);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a GUI_JPEG_INFO structure to be filled by the function.

Return value

Zero on success, nonzero if the function fails.

Add. information

For more details about the function and the parameters pfGetData and p please refer to the function GUI_JPEG_GetInfo() and GUI_JPEG_DrawEx() explained earlier in this chapter.
The sample folder contains the sample 2DGL_DrawJPGScaled.c which shows how to use the function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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. Unsupported data like raw text or application specific data will be ignored by emWin.
GIF files uses the LZW (Lempel-Zif-Welch) file compression method for compressing the image data. This compression method works without loosing data. The output image is exactly identical to the input image.

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 16Kbytes of dynamically allocated RAM for decompression. After drawing an image the RAM used for decompressing will be released.

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 the first image of a GIF file which has been loaded into memory.
GUI_GIF_DrawEx()	Draws the first image of a GIF file which needs not to be loaded into memory.
GUI_GIF_DrawSub()	Draws the given sub image of a GIF file which has been loaded into memory.
GUI_GIF_DrawSubEx()	Draws the given sub image of a GIF file which needs not to be loaded into memory.
GUI_GIF_DrawSubScaled()	Draws the given sub image of a GIF file with scaling which has been loaded into memory.
GUI_GIF_DrawSubScaledEx()	Draws the given sub image of a GIF file with scaling which needs not to be loaded into memory.
GUI_GIF_GetComment()	Returns the given comment of a GIF file which has been loaded into memory.
GUI_GIF_GetCommentEx()	Returns the given comment of a GIF file which needs not to be loaded into memory.
GUI_GIF_GetImageInfo()	Returns information about the given sub image of a GIF file which has been loaded into memory.
GUI_GIF_GetImageInfoEx()	Returns information about the given sub image of a GIF file which needs not to be loaded into memory.
GUI_GIF_GetInfo()	Returns information about a GIF file which has been loaded into memory.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
GUI_GIF_GetInfoEx()	Returns information about a GIF file which needs not to be loaded into memory.
GUI_GIF_GetXSize()	Returns the X-size of a bitmap loaded into memory.
GUI_GIF_GetXSizeEx()	Returns the X-size of a bitmap which needs not to be loaded into memory.
GUI_GIF_GetYSize()	Returns the Y-size of a bitmap loaded into memory.
GUI_GIF_GetYSizeEx()	Returns the Y-size of a bitmap which needs not to be loaded into memory.

GUI_GIF_Draw()

Description

Draws the first image of a .gif file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_Draw(const void * pGIF, U32 NumBytes, int x0, int y0);
```

Parameter	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

0 on success, != 0 on error.

Add. information

If the file contains more than one image, the function shows only the first image of the file. Transparency and interlaced images are supported.

GUI_GIF_DrawEx()

Description

Draws a .gif file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_DrawEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p,
                   int x0, int y0);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Return value

Zero on success, nonzero if the function fails.

Add. information

This function is used for drawing .gif files if not enough RAM is available to load the whole file into memory. The library calls the function pointed by the parameter pfGetData to read the data.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

The 'GetData' function should return the number of available bytes. This could be less or equal the number of requested bytes. The function needs at least to return 1 new byte.

GUI_GIF_DrawSub()

Description

Draws the given sub image of a .gif file, which has been loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_DrawSub(const void * pGIF, U32 NumBytes,
                    int x0, int y0, int Index);
```

Parameter	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.
Index	Zerobased index of sub image to be shown.

Return value

0 on success, != 0 on error.

Add. information

The function manages the background pixels between the current and the previous image. If for example sub image #3 should be drawn at offset x20/y20 with a size of w10/h10 and the previous sub image was shown at x15/y15 with a size of w20/h20 and the background needs to be redrawn, the function fills the pixels between the images with the background color.
The file 2DGL_DrawGIF.c of the sample folder shows how to use the function.

GUI_GIF_DrawSubEx()

Description

Draws the given sub image of a .gif file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_GIF_DrawSubEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p,
                      int x0, int y0, int Index);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Index	Zerobased index of sub image to be shown.

Return value

Zero on success, nonzero if the function fails.

Add. information

This function is used for drawing .gif images if not enough RAM is available to load the whole file into memory. The GUI library then calls the function pointed by the parameter pfGetData to read the data.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

For more details please refer to the function GUI_GIF_DrawEx() explained earlier in this chapter.

GUI_GIF_DrawSubScaled()

Description

Draws the given sub image of a .gif file, which has been loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int GUI_GIF_DrawSubScaled(const void * pGIF, U32 NumBytes, int x0, int y0,
                          int Index, int Num, int Denom);
```

Parameter	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.
Index	Zerobased index of sub image to be shown.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Return value

Zero on success, nonzero if the function fails.

Add. information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrinked to 2/3 of size the parameter Num should be 2 and Denom should be 3.

GUI_GIF_DrawSubScaledEx()

Description

Draws the given sub image of a .gif file, which does not have to be loaded into memory, at a specified position in the current window using scaling.

Prototype

```
int GUI_GIF_DrawSubScaledEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p,
                            int x0, int y0, int Index, int Num, int Denom);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.
Index	Zerobased index of sub image to be shown.
Num	Numerator to be used for scaling.
Denom	Denominator used for scaling.

Return value

Zero on success, nonzero if the function fails.

Add. information

The function scales the image by building a fraction with the given numerator and denominator. If for example an image should be shrinked to 2/3 of size the parameter Num should be 2 and Denom should be 3.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_GIF_GetComment()

Description

Returns the given comment from a GIF image, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetComment(const void * pGIF, U32 NumBytes,
                       U8 * pBuffer, int MaxSize, int Index);
```

Parameter	Meaning
pGIF	Pointer to the start of the memory area in which the .gif file resides.
NumBytes	Number of bytes of the .gif file.
pBuffer	Pointer to a buffer to be filled with the comment.
MaxSize	Size of the buffer.
Index	Zero based index of comment to be returned.

Return value

0 on success, != 0 on error.

Add. information

A GIF file can contain 1 or more comments. The function copies the comment into the given buffer. If the comment is larger than the given buffer only the bytes which fit into the buffer will be copied.
The file 2DGL_DrawGIF.c of the sample folder shows how to use the function.

GUI_GIF_GetCommentEx()

Description

Returns the given comment from a GIF image, which does not have to be loaded into memory.

Prototype

```
int GUI_GIF_GetCommentEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p,
                         U8 * pBuffer, int MaxSize, int Index);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
pBuffer	Pointer to a buffer to be filled with the comment.
MaxSize	Size of the buffer.
Index	Zero based index of comment to be returned.

Return value

0 on success, != 0 on error.

Add. information

For details please refer to the function GUI_GIF_GetComment().

GUI_GIF_GetImageInfo()

Description

Returns information about the given sub image of a GIF file, which has been loaded into memory.

Prototype

```
int GUI_GIF_GetImageInfo(const void * pGIF, U32 NumBytes,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                GUI_GIF_IMAGE_INFO * pInfo, int Index);
```

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.
pInfo	Pointer to a GUI_GIF_IMAGE_INFO structure which will be filled by the function.
Index	Zero based index of sub image.

Return value

0 on success, != 0 on error.

Elements of GUI_GIF_IMAGE_INFO

Data type	Element	Meaning
int	xPos	X position of the last drawn image.
int	yPos	Y position of the last drawn image.
int	xSize	X size of the last drawn image.
int	ySize	Y size of the last drawn image.
int	Delay	Time in 1/100 seconds the image should be shown in a movie.

Add. information

If an image needs be shown as a movie this function should be used to get the time the sub image should be visible and the next sub image should be shown.
If the Delay member is 0 the image should be visible for 1/10 second.

GUI_GIF_GetImageInfoEx()

Description

Returns information about the given sub image of a GIF file, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetImageInfoEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p,
                           GUI_GIF_IMAGE_INFO * pInfo, int Index);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a GUI_GIF_IMAGE_INFO structure which will be filled by the function.
Index	Zero based index of sub image.

Return value

0 on success, != 0 on error.

Add. information

For more details please refer to the function GUI_GIF_GetImageInfo().

GUI_GIF_GetInfo()

Description

Returns an information structure with information about the size and the number of sub images within the given GIF file, which has been loaded into memory.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_GIF_GetInfo(const void * pGIF, U32 NumBytes, GUI_GIF_INFO * pInfo);
```

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.
pInfo	Pointer to a GUI_GIF_INFO structure which will be filled by this function.

Return value

0 on success, != 0 on error.

Elements of GUI_GIF_INFO

Data type	Element	Meaning
int	XSize	Pixel size in X of the image.
int	YSize	Pixel size in Y of the image.
int	NumImages	Number of sub images in the file.

GUI_GIF_GetInfoEx()

Description

Returns an information structure with information about the size and the number of sub images within the given GIF file, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetInfoEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p,
                      GUI_GIF_INFO * pInfo);;
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
pInfo	Pointer to a GUI_GIF_INFO structure which will be filled by this function.

Return value

0 on success, != 0 on error.

Elements of GUI_GIF_INFO

Data type	Element	Meaning
int	XSize	Pixel size in X of the image.
int	YSize	Pixel size in Y of the image.
int	NumImages	Number of sub images in the file.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

X-size of the GIF image.

GUI_GIF_GetXSizeEx()

Description

Returns the X-size of a specified GIF image, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetXSizeEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.

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.

GUI_GIF_GetYSizeEx()

Description

Returns the Y-size of a specified GIF image, which needs not to be loaded into memory.

Prototype

```
int GUI_GIF_GetYSizeEx(GUI_GIF_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.

Return value

Y-size of the GIF image.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

8.4 PNG file support

The PNG (Portable Network Graphics) format is an image format which offers lossless data compression and alpha blending by using a non-patented data compression method. Version 1.0 of the PNG specification has been released in 1996. Since the end of 2003 PNG is an international standard (ISO/IEC 15948).
The emWin implementation of PNG support is based on the 'libpng' library from Glenn Randers-Pehrson, Guy Eric Schalnat and Andreas Dilger which is freely available under www.libpng.org. It is used in emWin under the copyright notice in GUI\PNG\png.h which allows using the library without any limitation.
The PNG library of emWin is available under www.segger.com/link/emwin_png.zip.

8.4.1 Converting a PNG file to 'C' source

Under some circumstances it can be useful to add a PNG file as 'C' file to the project. This can be done by exactly the same way as described before under 'JPEG file support'. Further the bitmap converter is able to load PNG files and can convert them into 'C' bitmap files.

8.4.2 Displaying PNG files

The graphic library first decodes the graphic information. If the image has to be drawn the decoding process takes considerable time. If a PNG file is used in a frequently called callback routine of the window manager, the decoding process can take a considerable amount of time. The calculation time can be reduced by the use of memory devices. The best way would be to draw the image first into a memory device. In this case the decompression would be executed only one time. For more information about memory devices please take a look to the memory device chapter.

8.4.3 Memory usage

The PNG decompression uses app. 21Kbytes of RAM for decompression independent of the image size and a size dependent amount of bytes. The RAM requirement can be calculated as follows:

App. RAM requirement = (X-Size + 1) * Y-Size * 4 + 21Kbytes

8.4.4 PNG file API

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

Routine	Explanation
GUI_PNG_Draw()	Draws the PNG file which has been loaded into memory.
GUI_PNG_DrawEx()	Draws the PNG file which needs not to be loaded into memory.
GUI_PNG_GetXSize()	Returns the X-size of a bitmap loaded into memory.
GUI_PNG_GetXSizeEx()	Returns the X-size of a bitmap which needs not to be loaded into memory.
GUI_PNG_GetYSize()	Returns the Y-size of a bitmap loaded into memory.
GUI_PNG_GetYSizeEx()	Returns the Y-size of a bitmap which needs not to be loaded into memory.

GUI_PNG_Draw()

Description

Draws a .PNG file, which has been loaded into memory, at a specified position in the current window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_PNG_Draw(const void * pFileData, int FileSize, int x0, int y0);
```

Parameter	Meaning
pFileData	Pointer to the start of the memory area in which the .PNG file resides.
FileSize	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 on success, nonzero if the function fails. (The current implementation always returns 0)

Add. information

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

GUI_PNG_DrawEx()

Description

Draws a .PNG file, which does not have to be loaded into memory, at a specified position in the current window.

Prototype

```
int GUI_PNG_DrawEx(GUI_GET_DATA_FUNC * pfGetData,
                   void * p, int x0, int y0);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.
x0	X-position of the upper left corner of the bitmap in the display.
y0	Y-position of the upper left corner of the bitmap in the display.

Return value

Zero on success, nonzero if the function fails.

Add. information

This function is used for drawing .PNG if not enough RAM is available to load the whole file into memory. The PNG library then calls the function pointed by the parameter pfGetData to read the data.
The 'GetData' function should return the number of available bytes. This could be less or equal the number of requested bytes. The function needs at least to return 1 new byte. Please note that the PNG library internally allocates a buffer for the complete image. This can not be avoided by using this function.

GUI_PNG_GetXSize()

Description

Returns the X-size of a specified PNG image, which has been loaded into memory.

Prototype

```
int GUI_PNG_GetXSize(const void * pFileData, int FileSize);
```

Parameter	Meaning
pFileData	Pointer to the start of the memory area in which the .PNG file resides.
FileSize	Size of the file in bytes.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

X-size of the PNG image.

GUI_PNG_GetXSizeEx()

Description

Returns the X-size of a specified PNG image, which needs not to be loaded into memory.

Prototype

```
int GUI_PNG_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.

Return value

X-size of the PNG image.

GUI_PNG_GetYSize()

Description

Returns the Y-size of a specified PNG image, which has been loaded into memory.

Prototype

```
int GUI_PNG_GetYSize(const void * pFileData, int FileSize);
```

Parameter	Meaning
pFileData	Pointer to the start of the memory area in which the .PNG file resides.
FileSize	Size of the file in bytes.

Return value

Y-size of the PNG image.

GUI_PNG_GetYSizeEx()

Description

Returns the X-size of a specified PNG image, which needs not to be loaded into memory.

Prototype

```
int GUI_PNG_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

Parameter	Meaning
pfGetData	Pointer to a function which is called for getting data. For details about the 'GetData' function please refer to the end of this chapter.
p	Void pointer passed to the function pointed by pfGetData.

Return value

Y-size of the PNG image.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

8.5 Getting data with the ...Ex() functions

As explained before drawing BMP, GIF and JPEG files also work without loading the whole image into RAM. For this case the ...Ex() functions can be used. Common for all of these functions is the use of a GetData-function.

Prototype of 'GetData' function

```
int APP_GetData(void * p,
                const U8 ** ppData,
                unsigned NumBytesReq,
                U32 Off);
```

Parameter	Meaning
p	Application defined void pointer.
NumBytesReq	Number of requested bytes.
ppData	Pointer to data pointer. This pointer should be set to a valid location. The reason why the application provided callback routine sets this pointer is that it allows efficient implementations of the callback routine: In cases where the data is already somewhere in memory, the data does not need to be copied.
StartOfFile	If this flag is 1, the data pointer should be set to the beginning of the data stream.

Example

The following code excerpt shows how to use a 'GetData' function:

```
static char acBuffer[0x200];

int APP_GetData(void * p, const U8 ** ppData, unsigned NumBytesReq, U32 Off) {
  DWORD NumBytesRead;
  HANDLE * phFile;

  phFile = (HANDLE *)p;
  /*
  * Check buffer size
  */
  if (NumBytesReq > sizeof(_acBuffer)) {
    NumBytesReq = sizeof(_acBuffer);
  }
  /*
  * Set file pointer to the required position
  */
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  /*
  * Read data into buffer
  */
  ReadFile(*phFile, _acBuffer, NumBytesReq, &NumBytesRead, NULL);
  /*
  * Set data pointer to the beginning of the buffer
  */
  *ppData = _acBuffer;
  /*
  * Return number of available bytes
  */
  return NumBytesRead;
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 9

Fonts

This chapter describes the various methods of font support in emWin. The most common fonts are shipped with emWin as 'C' font files. All of them contain the ASCII character set and most of them also the characters of ISO 8859-1. 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 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.
For accessing the full Unicode area of 65536 possible characters emWin supports UTF8 decoding which is described in chapter 23 "Foreign Language Support".

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.1 Introduction

The first way of font support was the possibility to use 'C' files with font definitions containing bitmaps with 1bpp pixel information for each character. This kind of font support was limited to use only the fonts which are compiled with the application.
Over time, the font support has been improved regarding font quality, ROM requirement, performance, scalability and the ability to add further fonts at run time.
In the meantime emWin fonts cover antialiasing, drawing of compound characters like required in Thai language, fonts located on external non addressable media and TrueType support. Except the TrueType font format, which is a vector font, all other kinds of fonts are bitmap fonts.

9.2 Font types

emWin supports different internal types of fonts defined by emWin and the common used TrueType fonts.

Monospaced bitmap fonts

Each character of a monospaced bitmap font has the same size. In a proportional font each character has its own width, whereas in a monospaced font the width is defined only one time. The pixel information is saved with 1bpp.

Proportional bitmap fonts

Each character of a proportional bitmap font has the same height and its own width. The pixel information is saved with 1bpp.

Antialiased fonts with 2 bpp antialiasing information

Each character has the same height and its own width. The pixel information is saved with 2bpp antialiasing information.

Antialiased fonts with 4 bpp antialiasing information

Each character has the same height and its own width. The pixel information is saved with 4bpp antialiasing information.

Extended proportional bitmap fonts

Each character of an extended proportional bitmap font has its own height and its own width. The pixel information stored in the previous listed font types cover the whole character areas, whereas the pixel information stored in extended proportional bitmap fonts only covers the area of the glyph bitmap.

Extended proportional bitmap fonts, framed

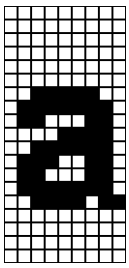
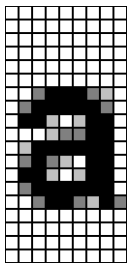
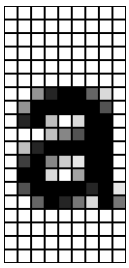
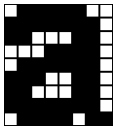
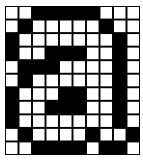
In some cases, for example in situations, where the background color is unknown at compile time, it can make sense to use a framed font. A framed font is always drawn in transparent mode regardless of the current settings. The character pixels are drawn in the currently selected foreground color and the frame is drawn in background color. A good contrast between foreground and background color makes sure, that the text can be read regardless of the background.
Please note that this type of font is not suitable for compound characters like in Thai language. It is also not suitable for Arabic fonts. The picture below shows some framed text in front of a photo:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Table of font types

The following table shows the difference between the font types. The pictures only show the pixel information saved in the font file:

Prop. bitmap font	Prop. bitmap font, AA2	Prop. bitmap font, AA2	Ext. prop. bitmap font *	Ext. prop. framed bitmap font *
				

TrueType vector fonts

The TrueType font support of emWin means support for the TrueType font file format described later in this chapter.

9.3 Font formats

The following explains the differences between the supported font formats, when to use them and what is required to be able to use them.

9.3.1 'C' file format

This is the most common way of using fonts. When using fonts in form of 'C' files, we recommend compiling all available fonts and linking them as library modules or putting all of the font object files in a library which you can link with your application. This way you can be sure that only the fonts which are needed by your application are actually linked. The font converter (described in a separate manual) may be used to create additional fonts.

When to use

This format should be used if the fonts are known at compile time and if there is enough addressable memory available for the font data.

Requirements

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

- The font file is in a form compatible with emWin as "C" file, object file or library.
- The font file is linked with your application.
- The font declaration is contained in the application.

Format description

A font 'C' file contains at first the pixel information of all characters included by the font. It is followed by a character information table with size information about each character. This table is followed by range information structures for each contiguous area of characters contained in the font file, whereas each structure points to the next one. Please note that this method can enlarge a font file a lot if using many separate characters. After the range information structures a `GUI_FONT` structure follows with the main information like type, pixel size and so on of the font.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.3.2 System Independent Font (SIF) format

System independent fonts are binary data blocks containing the font information. The font converter can be used to create system independent fonts. This tool is not part of the basic package. A short description follows later in this chapter. For details about how to create system independent fonts please refer to the font converter documentation.

When to use

This format should be used if the fonts are not known at compile time and if there is enough addressable memory available for the font data.

Requirements

In order to be able to use a SIF font file in your application, it is required that the whole file reside in addressable memory (ROM or RAM).

Format description

The structure of a SIF file is nearly the same as of a 'C' file. It contains the same information in binary format. The sequence of the file components is vice versa: General font information followed by range information structures, character information table and at least pixel information of all characters.

9.3.3 External Bitmap Font (XBF) format

As well as SIF fonts XBF fonts are binary data blocks containing the font information and the font converter can be used to create XBF files.
Contrary to other fonts, XBF fonts do not have to reside in memory when they are used, whereas all other kinds of emWin fonts need to reside completely in memory. The XBF font file can remain on any external media when it is used. Data access is done by a 'GetData' callback function whereas all other fonts need to reside in addressable memory (RAM or ROM). The advantage of XBF fonts is that it is possible to use very large fonts on system with little memory.

When to use

This format should be used if there is not enough addressable memory available for the font data and if there is any kind of external media available for storing the fonts.

Requirements

In order to be able to use a XBF font in your application, a 'GetData' callback function is required which is responsible for getting font data.

Format description

This format differs in general from SIF and 'C' file format. At first it contains a small block of general font information including the lowest character code and the highest character code. It is followed by an access table containing offset and data size information for each character between lowest and highest character code. If a character does not exist this information is zero for the according character. The access table is followed by the character information of all characters containing pixel data and character size information.

9.3.4 TrueType Font (TTF) format

TrueType is an outline font standard developed by Apple Computer. It offers font developers a high degree of control over how their fonts are displayed at various font heights. Contrary to bitmap fonts which are based on bitmaps for each character, TrueType fonts are based on vector graphics. The advantage of the vector representation is the loss-free scalability.
This implies that each character first needs to be rasterized into a bitmap before it is drawn. To avoid rasterization each time a character is drawn the bitmap data normally is cached by the font engine. This requires a fast CPU and enough RAM.
The emWin TTF package is not part of the shipment. It is freely available under www.segger.com/link/emwin_freetype.zip.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Licensing

The emWin implementation of the TTF support is based on the FreeType font library from David Turner, Robert Wilhelm and Werner Lemberg which is freely available under www.freetype.org. It is used in emWin under the FreeType license which can be found under GUI\TrueType\FTL.txt. It has been slightly adapted and a 'glue' layer with GUI-functions has been added.

When to use

This format should be used if fonts need to be scaleable at run-time.

Requirements

- CPU: TTF support works only on 32 bit CPUs. Our definition of a 32bit CPU: `sizeof(int) = 4`.
- ROM: The ROM requirement of the TTF engine is app. 250K. The exact size depends on the CPU, the compiler and the optimization level of the compiler.
- RAM: The RAM requirement of the library depends a lot on the used fonts. The basic RAM requirement of the TTF engine is app. 50K. When creating a GUI font with `GUI_TTF_CreateFont()` the font engine loads all font tables defined in the TTF file required to generate the characters. The table sizes varies a lot between the fonts. The additional required amount of RAM for creating a font can be between a few KB up to more than 1MB. For typical fonts 80-300kB are required. It depends on the used font file how much RAM is required. At least the TTF engine requires a bitmap cache. Per default the engine uses 200K for the cache. This should be enough for most applications.
  The TTF engine allocates its memory via the non emWin functions `malloc()` and `free()`. It must be made sure that these functions work before using the TTF engine.

Format description

For details about the TTF format please refer to the information available under www.apple.com.

9.4 Converting a TTF file to 'C' source

Under some circumstances it can be useful to add a TTF file as 'C' file to the project, for example if no file system is available. This can be done by using the tool `Bin2C.exe` shipped with emWin. It can be found in the Tools subfolder. It converts the given binary file (in this case the TTF file) to a 'C' file.

9.5 Declaring custom fonts

The most recommended way of declaring the prototypes of custom fonts is to put them into an application defined header file. This should be included from each application source file which uses these fonts. It could look like the following sample:

```
#include "GUI.h"

extern GUI_CONST_STORAGE GUI_FONT GUI_FontApp1;
extern GUI_CONST_STORAGE GUI_FONT GUI_FontApp2;
```

Please note that this kind of declaring prototypes does not work if the fonts should be used with emWin configuration macros like `BUTTON_FONT_DEFAULT` or similar. In this case the fonts need to be declared in the configuration file `GUIConf.h`. The declaration in this case can look like the following sample:

```
typedef struct GUI_FONT GUI_FONT;

extern const GUI_FONT GUI_FontApp1;

#define BUTTON_FONT_DEFAULT &GUI_FontApp1
#define EDIT_FONT_DEFAULT   &GUI_FontApp1
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

The typedef is required because the structure GUI_FONT has not been defined at the early point where GUIConf.h is included by emWin.

9.6 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() or one of the GUI_XXX_CreateFont() functions, which select the font to use for all text output to follow for the current task.
If no font has been selected by your application, the default font is used. This default is configured in GUIConf.h and can be changed. You should make sure that the default font is one that you are actually using in your application because the default font will be linked with your application and will therefore use up ROM memory.

9.7 Font API

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

Routine	Explanation
'C' file related font functions	
GUI_SetFont()	Sets the current font.
'SIF' file related font functions	
GUI_SIF_CreateFont()	Creates and selects a font by passing a pointer to system independent font data.
GUI_SIF_DeleteFont()	Deletes a font created by GUI_SIF_CreateFont()
'TTF' file related font functions	
GUI_TTF_CreateFont()	Creates a GUI font from a TTF font file.
GUI_TTF_DestroyCache()	Destroys the cache of the TTF engine.
GUI_TTF_Done()	Frees all dynamically allocated memory of the TTF engine.
GUI_TTF_GetFamilyName()	Returns the family name of the font.
GUI_TTF_GetStyleName()	Returns the stype nabe of the font.
GUI_TTF_SetCacheSize()	Can be used to set the default size of the TTF cache.
'XBF' file related font functions	
GUI_XBF_CreateFont()	Creates and selects a font by passing a pointer to a callback function, which is responsible for getting data from the XBF font file.
GUI_XBF_DeleteFont()	Deletes a font created by GUI_XBF_CreateFont()
Common font-related functions	
GUI_GetCharDistX()	Returns the width in pixels (X-size) of a specified character in the current font.
GUI_GetFont()	Returns a pointer to the currently selected font.
GUI_GetFontDistY()	Returns the Y-spacing of the current font.
GUI_GetFontInfo()	Returns a structure containing font information.
GUI_GetFontSizeY()	Returns the height in pixels (Y-size) of the current font.
GUI_GetLeadingBlankCols()	Returns the number of leading blank pixel columns of the given character.
GUI_GetStringDistX()	Returns the X-size of a text using the current font.
GUI_GetTextExtend()	Evaluates the size of a text using the current font
GUI_GetTrailingBlankCols()	Returns the number of trailing blank pixel columns of the given character.
GUI_GetYDistOfFont()	Returns the Y-spacing of a particular font.
GUI_GetYSizeOfFont()	Returns the Y-size of a particular font.
GUI_IsInFont()	Evaluates whether a specified character is in a particular font.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.8 'C' file related font functions

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

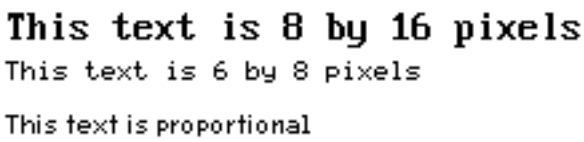
Returns a pointer to the previously selected font so that it may be restored at a later point.

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
}
```

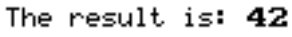
Screen shot of above example:



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:



9.9 'SIF' file related font functions

GUI_SIF_CreateFont()

Description

Sets the font to be used by passing a pointer to system independent font data.

Prototype

```
void GUI_SIF_CreateFont(void * pFontData, GUI_FONT * pFont,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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.
GUI_SIF_TYPE_PROP_EXT	Should be used if the parameter pFont points to a non antialiased extended proportional font.
GUI_SIF_TYPE_PROP_FRM	Should be used if the parameter pFont points to a non antialiased extended framed proportional font.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

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

9.10 ’TTF’ file related font functions

The emWin implementation of TTF file support is based on the FreeType font library from David Turner, Robert Wilhelm and Werner Lemberg. For details please refer to the TTF format description at the beginning of this chapter.

GUI_TTF_CreateFont()

Description

Creates and selects an emWin font by using a TTF font file.

Prototype

```
int GUI_TTF_CreateFont(GUI_FONT * pFont, GUI_TTF_CS * pCS);
```

Parameter	Meaning
pFont	Pointer to a GUI_FONT structure in RAM filled by the function.
pCS	Pointer to a GUI_TTF_CS structure containing the creation parameters.

Return value

0 on success, 1 on error.

Elements of GUI_TTF_CS

Data type	Element	Meaning
GUI_TTF_DATA *	pTTF	Pointer to GUI_TTF_DATA structure which contains location and size of the font file to be used.
PixelHeight	PixelHeight	Pixel height of new font. It means the height of the surrounding rectangle between the glyphs 'g' anf 'f'. Please notice that it is not the distance between two lines of text. With other words the value returned by GUI_GetFontSizeY() is not identically with this value.
FaceIndex	FaceIndex	Some font files can contain more than one font face. In case of more than one face this index specifies the zero based face index to be used to create the font. Usually 0.

Elements of GUI_TTF_DATA

Data type	Element	Meaning
const void *	pData	Pointer to TTF font file in addressable memory area.
NumBytes	NumBytes	Size of file in bytes.

Add. Information

When using the function the first time it initializes the TTF engine and the internal cache system. If the cache should use other values as defined per default it needs to be configured before the first call of this function. For details how to configure the cache please refer to the function GUI_TTF_SetCacheSize().
The internal data cache manages the complete mechanism of creating fonts and caching bitmap data. Font faces are uniquely identified from the cache by the address given in parameter pTTF and the parameter FaceIndex, which normally is 0. If the same font file for example should be used for creating fonts of different sizes the parameter pTTF should point to the same location of a GUI_TTF_DATA structure.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

The parameter PixelHeight specifies the height of the surrounding rectangle between the glyphs ‘g’ anf ‘f’. Please note that the value of PixelHeight this is not the same as the offset from one line of text to the next line.

Example

```
GUI_TTF_CS   Cs0, Cs1;
GUI_TTF_DATA Data;
GUI_FONT     Font0, Font1;
/* Set parameters for accessing the font file */
Data.pData      = aTTF;         /* Address */
Data.NumBytes   = sizeof(aTTF); /* Size */
/* Set creation parameters of first font */
Cs0.pTTF        = &Data; /* Use address of GUI_TTF_DATA */
Cs0.PixelHeight = 24;    /* Pixel height */
Cs0.FaceIndex   = 0;     /* Initialize to 0 */
/* Set creation parameters of second font */
Cs1.pTTF        = &Data; /* Use address of GUI_TTF_DATA */
Cs1.PixelHeight = 48;    /* Pixel height */
Cs1.FaceIndex   = 0;     /* Initialize to 0 */
/* Create 2 fonts */
GUI_TTF_CreateFont(&Font0, &Cs0);
GUI_TTF_CreateFont(&Font1, &Cs1);
/* Draw something using the fonts */
GUI_SetFont(&Font0);
GUI_DispString("Hello world\n");
GUI_SetFont(&Font1);
GUI_DispString("Hello world");
```

GUI_TTF_DestroyCache()

Description

This function frees all memory allocated by the TTF cache system and destroys the cache.

Prototype

```
void GUI_TTF_DestroyCache(void);
```

Add. Information

The next time GUI_TTF_CreateFont() is used emWin automatically creates and initializes a new cache.

GUI_TTF_Done()

Description

This function frees all memory allocated by the TTF engine and its internal cache system.

Prototype

```
void GUI_TTF_Done(void);
```

Add. Information

The next time GUI_TTF_CreateFont() is used emWin automatically initializes the TTF engine and creates and initializes a new cache.

GUI_TTF_GetFamilyName()

Description

The function returns the font family name defined in the font file.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_TTF_GetFamilyName(GUI_FONT * pFont, char * pBuffer, int NumBytes);
```

Parameter	Meaning
pFont	Pointer to a GUI_FONT structure which has been created using GUI_TTF_CreateFont().
pBuffer	Buffer to be filled with the family name.
NumBytes	Size of buffer in bytes.

Return value

0 on success, 1 on error.

GUI_TTF_GetStyleName()

Description

The function returns the style name (bold, regular, ...) defined in the font file.

Prototype

```
int GUI_TTF_GetStyleName(GUI_FONT * pFont, char * pBuffer, int NumBytes);
```

Parameter	Meaning
pFont	Pointer to a GUI_FONT structure which has been created using GUI_TTF_CreateFont().
pBuffer	Buffer to be filled with the style name.
NumBytes	Size of buffer in bytes.

Return value

0 on success, 1 on error.

GUI_TTF_SetCacheSize()

Description

Sets the size parameters used to create the cache on the first call of GUI_TTF_CreateFont().

Prototype

```
void GUI_TTF_SetCacheSize(unsigned MaxFaces,
                          unsigned MaxSizes, U32 MaxBytes);
```

Parameter	Meaning
MaxFaces	Maximum number of font faces the cache should be able to handle simultaneously. 0 selects default value.
MaxSizes	Maximum number of size objects the cache should be able to handle simultaneously. 0 selects default value.
MaxBytes	Maximum number of bytes used for the bitmap cache. 0 selects default value.

Add. Information

If for example 3 font faces should be used, each with 2 sizes, the cache should be able to manage 6 size objects.
The default values used by the TTF engine are: 2 faces, 4 size objects and 200K of bitmap data cache.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.11 'XBF' file related font functions

GUI_XBF_CreateFont()

Description

Creates and selects a font by passing a pointer to a callback function, which is responsible for getting data from the XBF font file.

Prototype

```
int GUI_XBF_CreateFont(GUI_FONT * pFont,
                       GUI_XBF_DATA * pXBF_Data,
                       const GUI_XBF_TYPE * pFontType,
                       GUI_XBF_GET_DATA_FUNC * pfGetData,
                       void * pVoid);
```

Parameter	Meaning
pFont	Pointer to a GUI_FONT structure in RAM filled by the function.
pXBF_Data	Pointer to a GUI_XBF_DATA structure in RAM filled by the function.
pFontType	See table below.
pfGetData	Pointer to a callback function which is responsible for getting data from the font file.
pVoid	Application defined pointer passed to the 'GetData' callback function.

Permitted values for element pFontType	
GUI_XBF_TYPE_PROP	Should be used if the parameter pFont points to a non antialiased proportional font.
GUI_XBF_TYPE_PROP_EXT	Should be used if the parameter pFont points to a non antialiased extended proportional font.
GUI_XBF_TYPE_PROP_FRM	Should be used if the parameter pFont points to a non antialiased extended framed proportional font.

Add. Information

Contrary to the emWin standard fonts, which must be compiled and linked with the application program, external binary fonts (XBF) are binary data blocks containing the font information. The font converter can be used to create XBF files. This tool is not part of the basic package. A short description of the font converter follows later in this chapter. For details about how to create external binary fonts please refer to the font converter documentation.
Contrary to other fonts, XBF fonts do not have to reside in memory when they are used, whereas all other kinds of emWin fonts need to reside completely in memory. The XBF font file can remain on any external media during it is used. Data access is done by a 'GetData' callback function whereas all other fonts need to reside in addressable memory (RAM or ROM). The advantage of XBF fonts is that it is possible to use very large fonts on system with little memory.
The parameter pfGetData should point to an application defined callback routine, which is responsible for getting data from the font. Parameter pVoid is passed to the callback function when requesting font data. It can be used for example to pass a file handle to the callback function.
The function requires pointers to a GUI_FONT structure and a GUI_XBF_DATA structure. These structures will be filled by the function with font information. It is required, that the contents of this structures remain valid during the use of the font. Because the function does not know what kind of XBF font should be created, the parameter pFontType is used to tell the function the type of the font to be created. This has been done to avoid linkage of code which is not required.
The maximum number of data bytes per character is limited to 200 per default. This should cover the most requirements. If loading a character with more bytes a warning will be generated in the debug version. The default value can be increased by adding the following define to the file GUIConf.h:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
#define GUI_MAX_XBF_BYTES 500 /* Sets the maximum number of bytes/character to 500*/
```

Example

```
static GUI_FONT     Font;     /* GUI_FONT structure in RAM */
static GUI_XBF_DATA XBF_Data; /* GUI_XBF_DATA structure in RAM */

static int _cbGetData(U32 Off, U16 NumBytes, void * pVoid, void * pBuffer) {
  /* May use the pVoid pointer to get a file handle */
  .../* TBD */
  /* Set file pointer to the given position */
  .../* TBD */
  /* Read the required number of bytes into the given buffer */
  .../* TBD */
}

void CreateXBF_Font(void * pVoid) {
  GUI_XBF_CreateFont(&Font,             /* Pointer to GUI_FONT structure */
                     &XBF_Data,         /* Pointer to GUI_XBF_DATA structure */
                     GUI_XBF_TYPE_PROP, /* Font type to be created */
                     _cbGetData,        /* Pointer to callback function */
                     pVoid);            /* Pointer to be passed to callback */

}
```

GUI_XBF_DeleteFont()

Description

Deletes a XBF font pointed by the parameter pFont.

Prototype

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

Parameter	Meaning
pFont	Pointer to the font to be deleted.

Add. Information

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

9.12 Common font-related functions

GUI_GetFont()

Description

Returns a pointer to the currently selected font.

Prototype

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

GUI_GetCharDistX()

Description

Returns the width in pixels (X-size) used to display a specified character in the currently selected font.

Prototype

```
int  GUI_GetCharDistX(U16 c);
```

Parameter	Meaning
c	Character to calculate width from.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_GetLeadingBlankCols()

Description

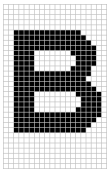
Returns the number of leading blank pixel columns in the currently selected font for the given character.

Prototype

```
int GUI_GetLeadingBlankCols(U16 c);
```

Parameter	Meaning
c	Character to be used.

Sample



The result for the character 'B' shown in the screenshot above should be 2.

GUI_GetStringDistX()

Description

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

Prototype

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

Parameter	Meaning
s	Pointer to the string.

GUI_GetTextExtend()

Description

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

Prototype

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

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

GUI_GetTrailingBlankCols()

Description

Returns the number of trailing blank pixel columns in the currently selected font for the given character.

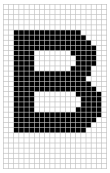
Prototype

```
int GUI_GetTrailingBlankCols(U16 c);
```

Parameter	Meaning
c	Character to be used.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Sample



The result for the character 'B' shown in the screenshot above should be 1.

GUI_GetYDistOfFont()

Description

Returns the Y-spacing of a particular font.

Prototype

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

Parameter	Meaning
pFont	Pointer to the font.

Add. information

(see GUI_GetFontDistY())

GUI_GetYSizeOfFont()

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_DispString("GUI_FontD32 does not contains 'X'");
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.13 Character sets

9.13.1 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.

9.13.2 ISO 8859-1 Western Latin character set

emWin supports the ISO 8859-1, which defines characters as listed below:

Code	Description	Char
160	non-breaking space	
161	inverted exclamation	¡
162	cent sign	¢
163	pound sterling	£
164	general currency sign	¤
165	yen sign	¥
166	broken vertical bar	¦
167	section sign	§
168	umlaut (dieresis)	¨
169	copyright	©
170	feminine ordinal	ª
171	left angle quote, guillemotleft	«
172	not sign	¬
173	soft hyphen	
174	registered trademark	®
175	macron accent	¯
176	degree sign	°
177	plus or minus	±
178	superscript two	²
179	superscript three	³
180	acute accent	´
181	micro sign	µ
182	paragraph sign	¶
183	middle dot	·
184	cedilla	¸
185	superscript one	¹
186	masculine ordinal	º
187	right angle quote, guillemot right	»
188	fraction one-fourth	¼
189	fraction one-half	½

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

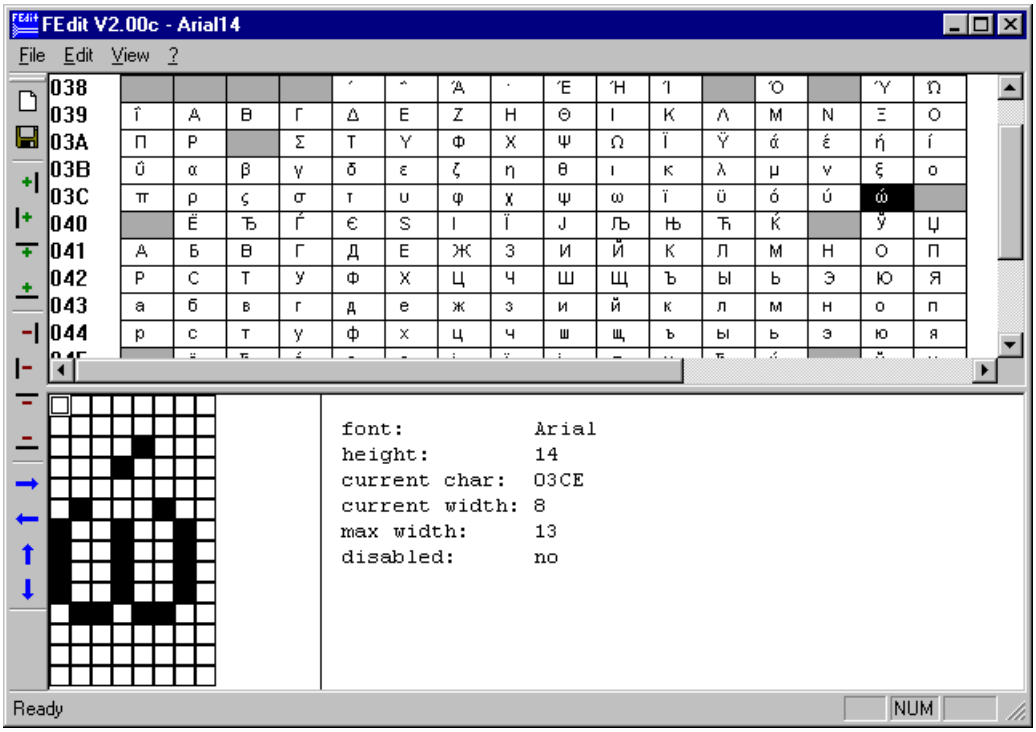
9.13.3 Unicode

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.14 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).

9.14.1 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;
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.15 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.

9.15.1 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
GUI_Font	Standard prefix for all fonts shipped with emWin.
<style>	Specifies a non-standard font style. Example: Comic style in GUI_FontComic18B_ASCII.
<width>	Width of characters, contained only in monspaced fonts.
<height>	Height of the font in pixels.
<MagX>	Factor of magnification in X, contained only in magnified fonts.
<MagY>	Factor of magnification in Y, contained only in magnified fonts.
H	Abbreviation for "high". Only used if there is more than one font with the same height. It means that the font appears "higher" than other fonts.
B	Abbreviation for "bold". Used in bold fonts.
<characterset>	Specifies the contents of characters: ASCII: Only ASCII characters 0x20-0x7E (0x7F). 1: ASCII characters and European extensions 0xA0 - 0xFF. HK: Hiragana and Katakana. 1HK: ASCII, European extensions, Hiragana and Katakana. D: Digit fonts, character set: +-.0123456789.

Example 1

`GUI_Font16_ASCII`

Element	Meaning
GUI_Font	Standard font prefix.
16	Heigth in pixels.
ASCII	Font contains ASCII characters only.

Example 2

`GUI_Font8x15B_ASCII`

Element	Meaning
GUI_Font	Standard font prefix.
8	Width of characters.
x15	Heigth in pixels.
B	Bold font.
ASCII	Font contains ASCII characters only.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example 3

```
GUI_Font8x16x1x2
```

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

9.15.2 Font file naming convention

The names for the font files are similar to the names of the fonts themselves. The files are named as follows:

```
F[<width>]<height>[H][B][<characterset>]
```

Element	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.
<characterset>	Specifies the contents of characters: ASCII: Only ASCII characters 0x20-0x7E (0x7F). 1: ASCII characters and European extensions 0xA0 - 0xFF. HK: Hiragana and Katakana. 1HK: ASCII, European extensions, Hiragana and Katakana. D: Digit fonts.

9.15.3 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:

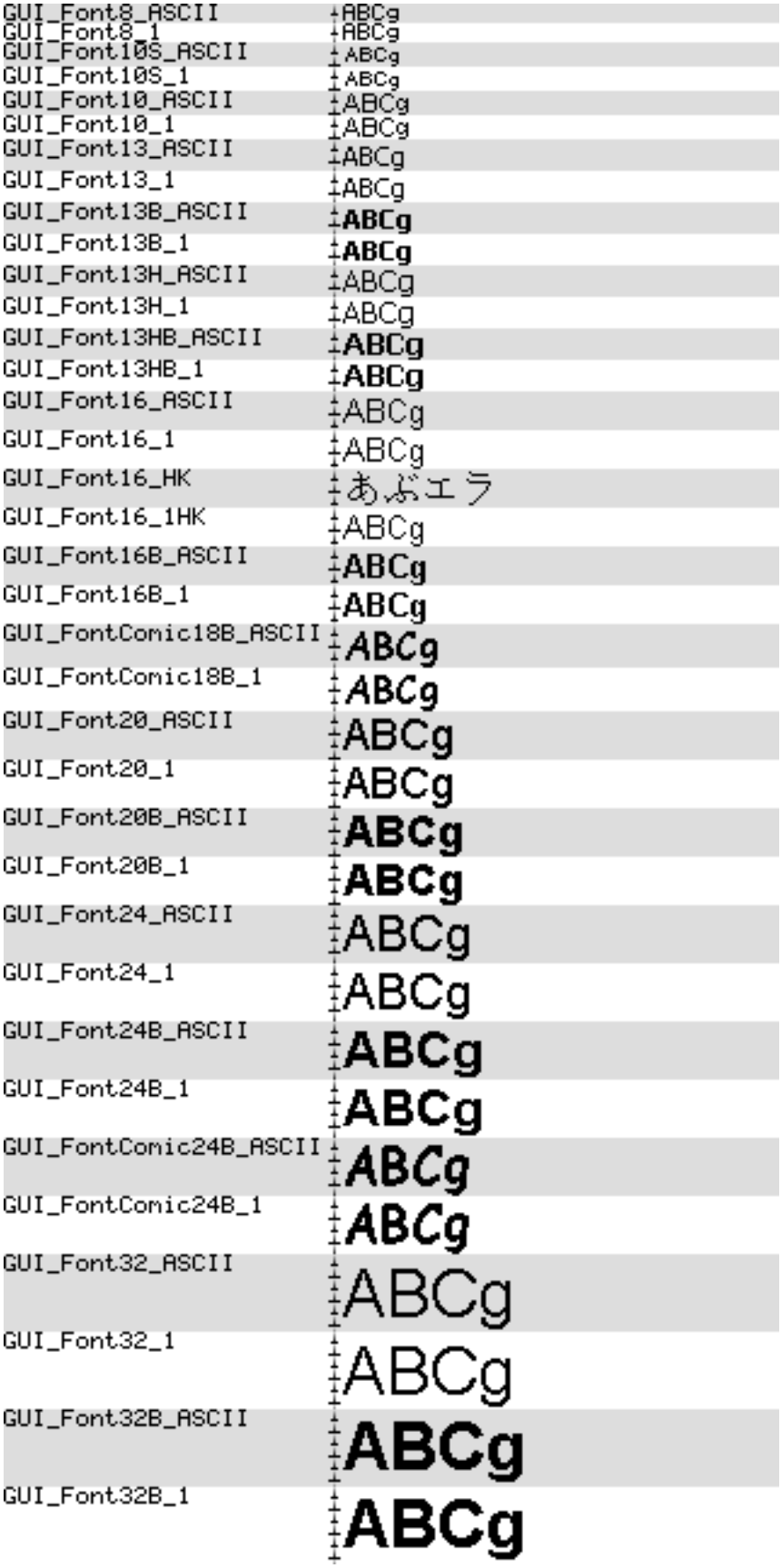
Element	Meaning
F	Size of font in Y.
B	Distance of base line from the top of the font.
C	Height of capital characters.
L	Height of lowercase characters.
U	Size of underlength used by letters such as "g", "j" or "y".

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.15.4 Proportional fonts

9.15.4.1 Overview

The following screenshot gives an overview of all available proportional fonts:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.15.4.2 Measurement, ROM size and used files

The following table shows the measurement, ROMsize and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_Font8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1562	F08_ASCII.c
GUI_Font8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1562+ 1586	F08_ASCII.c F08_1.c
GUI_Font10S_ASCII	F: 10, B: 8, C: 6, L: 4, U: 2	1760	F10S_ASCII.c
GUI_Font10S_1	F: 10, B: 8, C: 6, L: 4, U: 2	1760+ 1770	F10_ASCII.c F10_1.c
GUI_Font10_ASCII	F: 10, B: 9, C: 8, L: 6, U: 1	1800	F10_ASCII
GUI_Font10_1	F: 10, B: 9, C: 8, L: 6, U: 1	1800+ 2456	F10_ASCII.c F10_1.c
GUI_Font13_ASCII	F: 13, B: 11, C: 8, L: 6, U: 2	2076	F13_ASCII.c
GUI_Font13_1	F: 13, B: 11, C: 8, L: 6, U: 2	2076+ 2149	F13_ASCII.c F13_1.c
GUI_Font13B_ASCII	F: 13, B: 11, C: 8, L: 6, U: 2	2222	F13B_ASCII.c
GUI_Font13B_1	F: 13, B: 11, C: 8, L: 6, U: 2	2222+ 2216	F13B_ASCII.c F13B_1.c
GUI_Font13H_ASCII	F: 13, B: 11, C: 9, L: 7, U: 2	2232	F13H_ASCII.c
GUI_Font13H_1	F: 13, B: 11, C: 9, L: 7, U: 2	2232+ 2291	F13H_ASCII.c F13H_1.c
GUI_Font13HB_ASCII	F: 13, B: 11, C: 9, L: 7, U: 2	2690	F13HB_ASCII.c
GUI_Font13HB_1	F: 13, B: 11, C: 9, L: 7, U: 2	2690+ 2806	F13HB_ASCII.c F13HB_1.c
GUI_Font16_ASCII	F: 16, B: 13, C: 10, L: 7, U: 3	2714	F16_ASCII.c
GUI_Font16_1	F: 16, B: 13, C: 10, L: 7, U: 3	2714+ 3850	F16_ASCII.c F16_1.c
GUI_Font16_HK	-	6950	F16_HK.c
GUI_Font16_1HK	F: 16, B: 13, C: 10, L: 7, U: 3	120+ 6950+ 2714+ 3850	F16_1HK.c F16_HK.c F16_ASCII.c F16_1.c
GUI_Font16B_ASCII	F: 16, B: 13, C: 10, L: 7, U: 3	2690	F16B_ASCII.c
GUI_Font16B_1	F: 16, B: 13, C: 10, L: 7, U: 3	2690+ 2790	F16B_ASCII.c F16B_1.c
GUI_FontComic18B_ASCII	F: 18, B: 15, C: 12, L: 9, U: 3	3572	FComic18B_ASCII.c
GUI_FontComic18B_1	F: 18, B: 15, C: 12, L: 9, U: 3	3572+ 4334	FComic18B_ASCII.c FComic18B_1.c
GUI_Font20_ASCII	F: 20, B: 16, C: 13, L: 10, U: 4	4044	F20_ASCII.c
GUI_Font20_1	F: 20, B: 16, C: 13, L: 10, U: 4	4044+ 4244	F20_ASCII.c F20_1.c
GUI_Font20B_ASCII	F: 20, B: 16, C: 13, L: 10, U: 4	4164	F20B_ASCII.c
GUI_Font20B_1	F: 20, B: 16, C: 13, L: 10, U: 4	4164+ 4244	F20B_ASCII.c F20B_1.c
GUI_Font24_ASCII	F: 24, B: 20, C: 17, L: 13, U: 4	4786	F24_ASCII.c

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

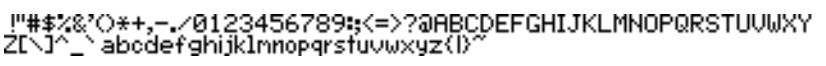
Font name	Measurement	ROM size in bytes	Used files
GUI_Font24_1	F: 24, B: 20, C: 17, L: 13, U: 4	4786+ 5022	F24_ASCII.c F24_1.c
GUI_Font24B_ASCII	F: 24, B: 19, C: 15, L: 11, U: 5	4858	F24B_ASCII.c
GUI_Font24B_1	F: 24, B: 19, C: 15, L: 11, U: 5	4858+ 5022	F24B_ASCII.c F24B_1.c
GUI_FontComic24B_ASCII	F: 24, B: 20, C: 17, L: 13, U: 4	6146	FComic24B_ASCII
GUI_FontComic24B_1	F: 24, B: 20, C: 17, L: 13, U: 4	6146+ 5598	FComic24B_ASCII FComic24B_1
GUI_Font32_ASCII	F: 32, B: 26, C: 20, L: 15, U: 6	7234	F32_ASCII.c
GUI_Font32_1	F: 32, B: 26, C: 20, L: 15, U: 6	7234+ 7734	F32_ASCII.c F32_1.c
GUI_Font32B_ASCII	F: 32, B: 25, C: 20, L: 15, U: 7	7842	F32B_ASCII.c
GUI_Font32B_1	F: 32, B: 25, C: 20, L: 15, U: 7	7842+ 8118	F32B_ASCII.c F32B_1.c

9.15.4.3 Characters

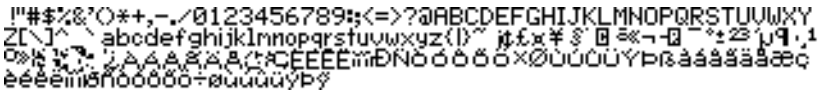
The following shows all characters of all proportional standard fonts:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

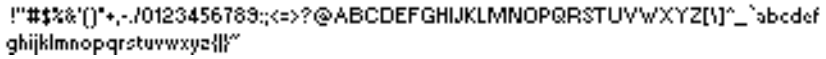
GUI_Font8_ASCII



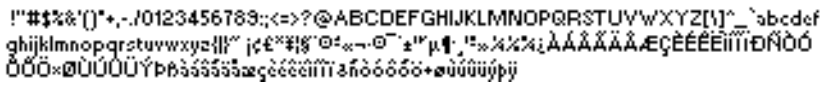
GUI_Font8_1



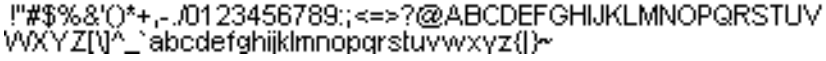
GUI_Font10S_ASCII



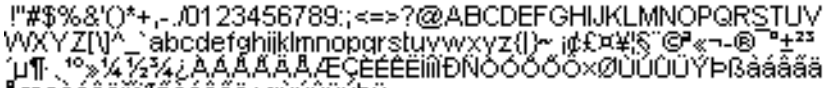
GUI_Font10S_1



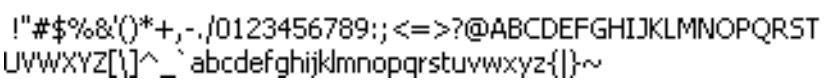
GUI_Font10_ASCII



GUI_Font10_1

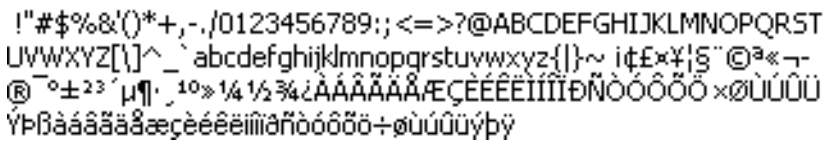


GUI_Font13_ASCII



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

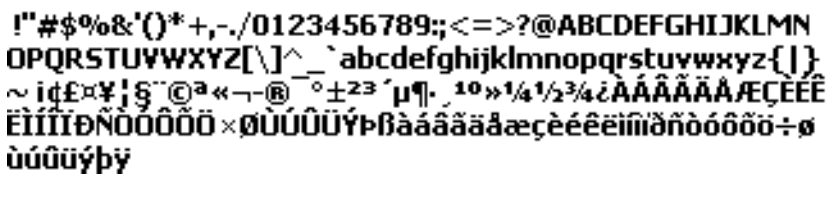
GUI_Font13_1



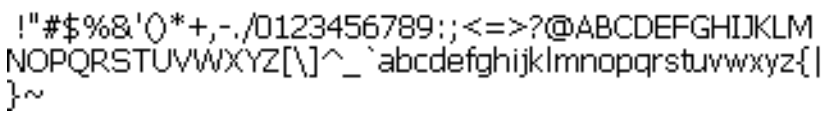
GUI_Font13B_ASCII



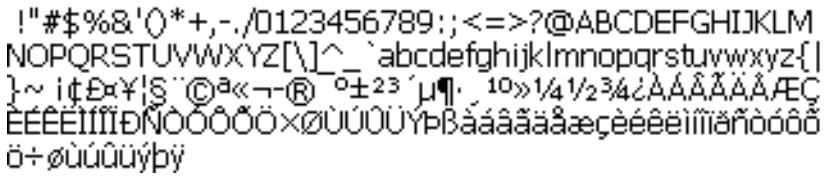
GUI_Font13B_1



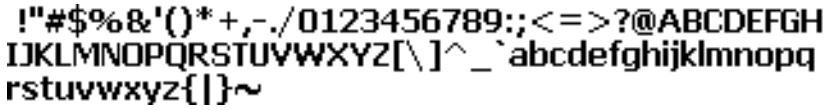
GUI_Font13H_ASCII



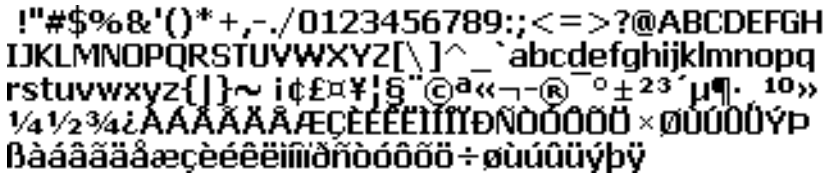
GUI_Font13H_1



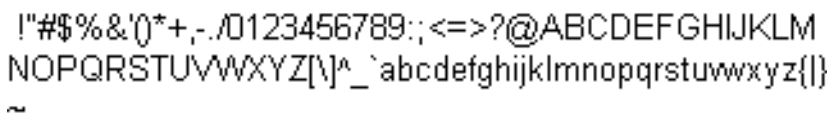
GUI_Font13HB_ASCII



GUI_Font13HB_1

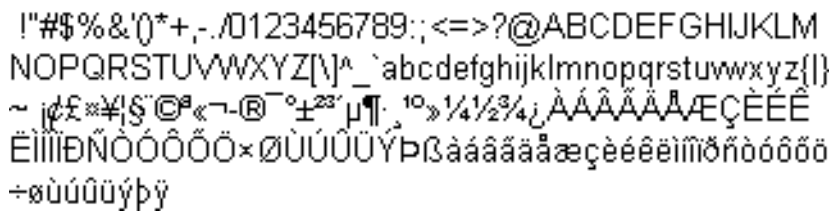


GUI_Font16_ASCII

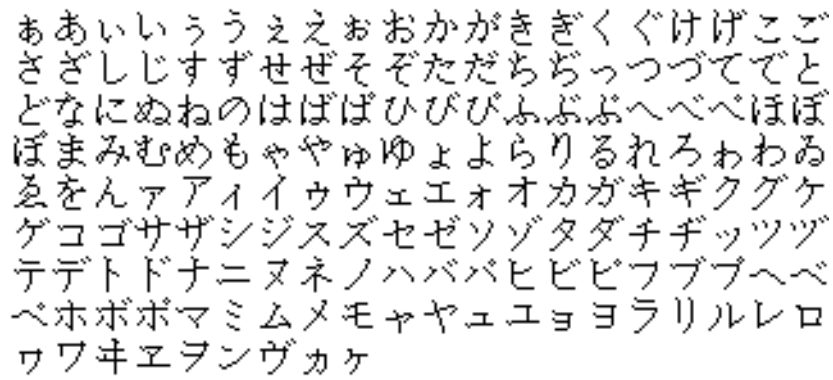


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

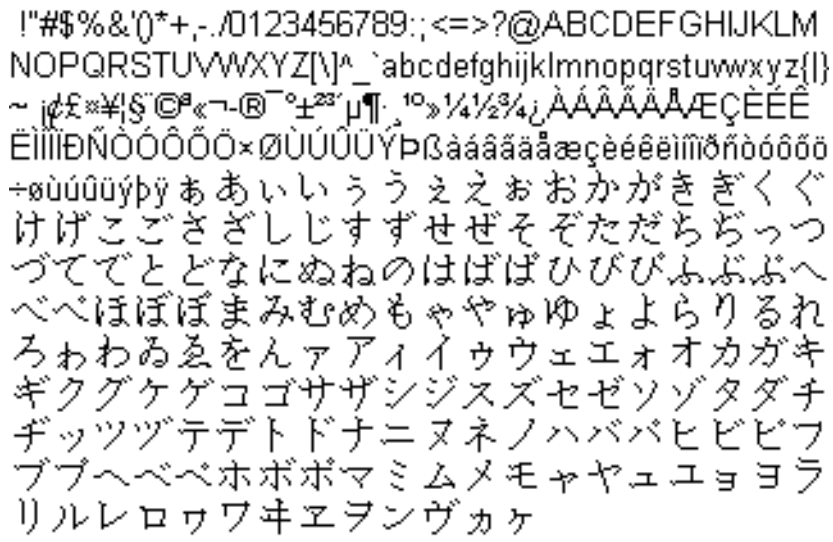
GUI_Font16_1



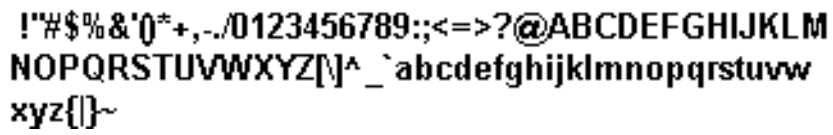
GUI_Font16_HK



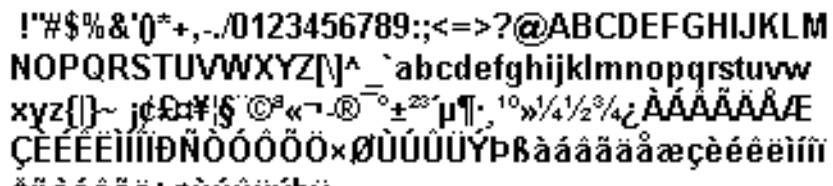
GUI_Font16_1HK



GUI_Font16B_ASCII

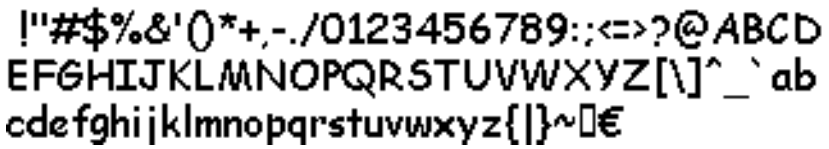


GUI_Font16B_1

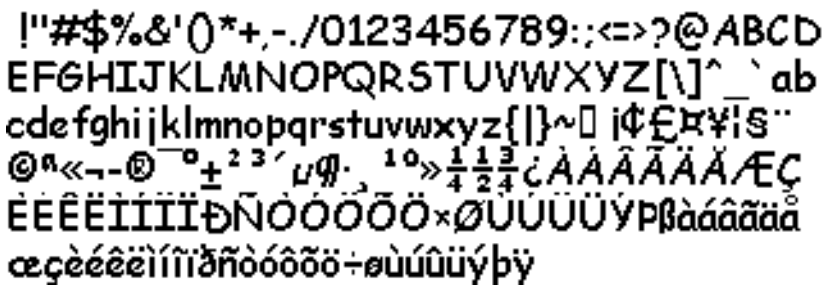


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

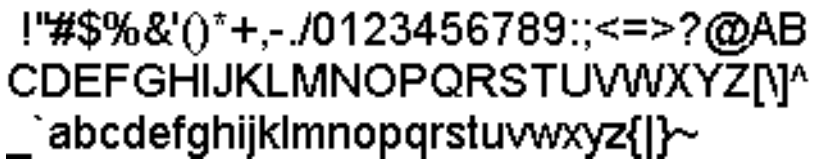
GUI_FontComic18B_ASCII



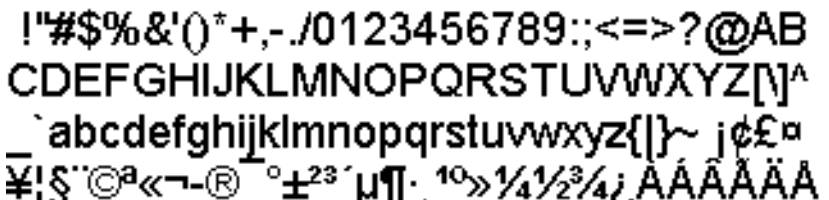
GUI_FontComic18B_1



GUI_Font20_ASCII



GUI_Font20_1



GUI_Font20B_ASCII

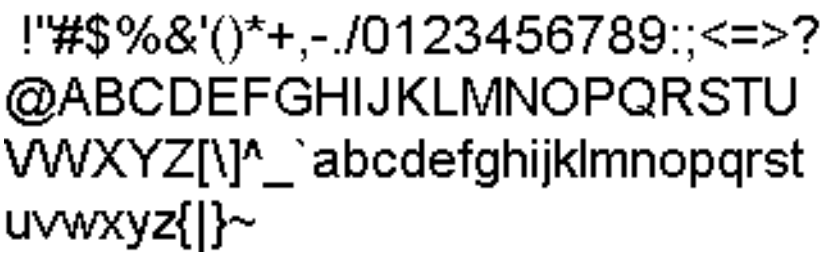


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

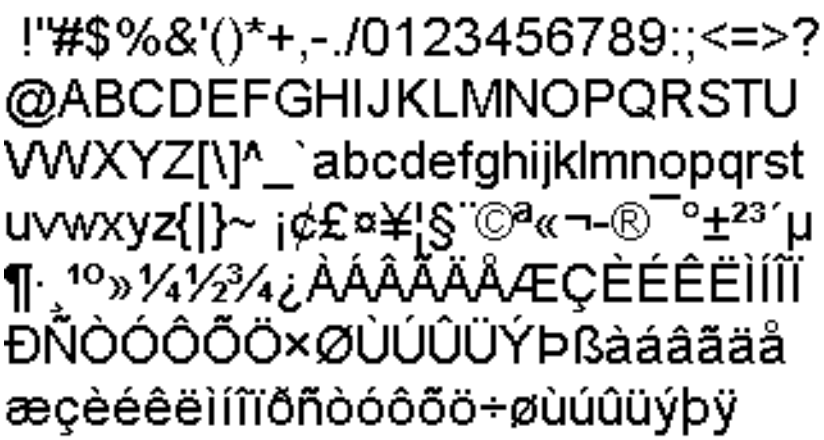
GUI_Font20B_1



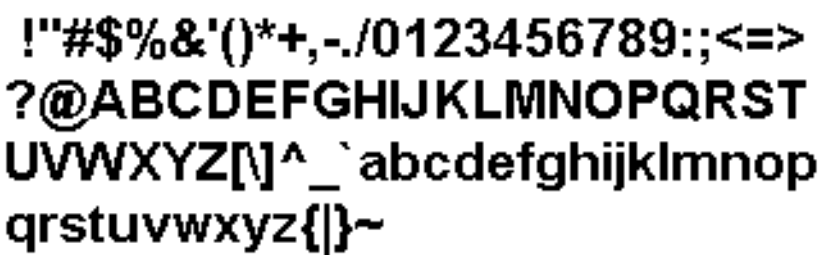
GUI_Font24_ASCII



GUI_Font24_1

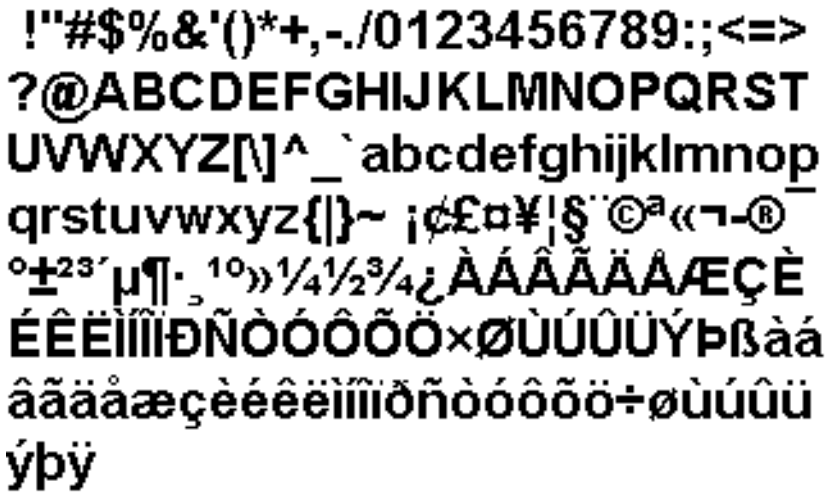


GUI_Font24B_ASCII

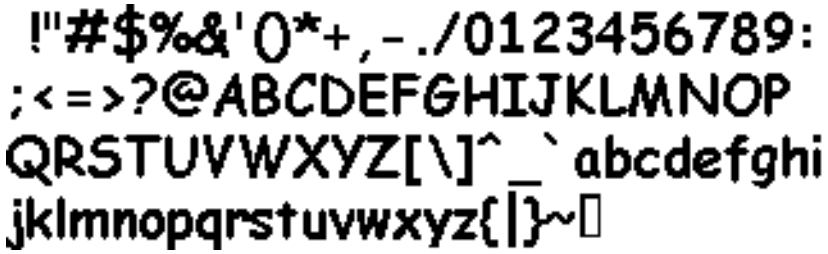


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

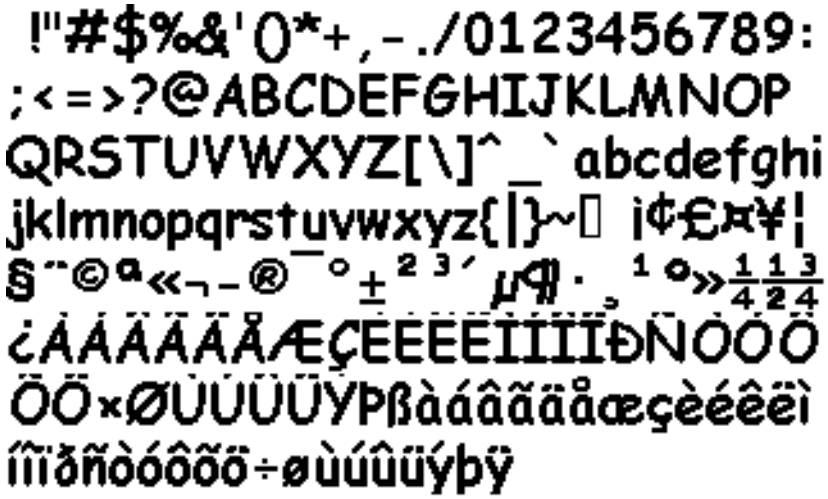
GUI_Font24B_1



GUI_FontComic24B_ASCII

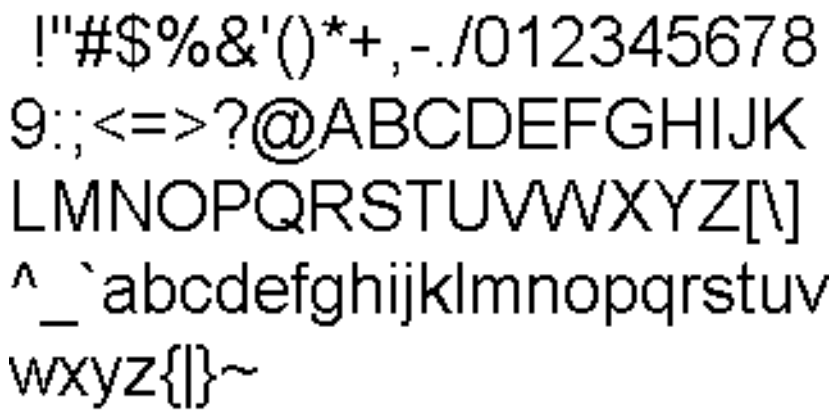


GUI_FontComic24B_1



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

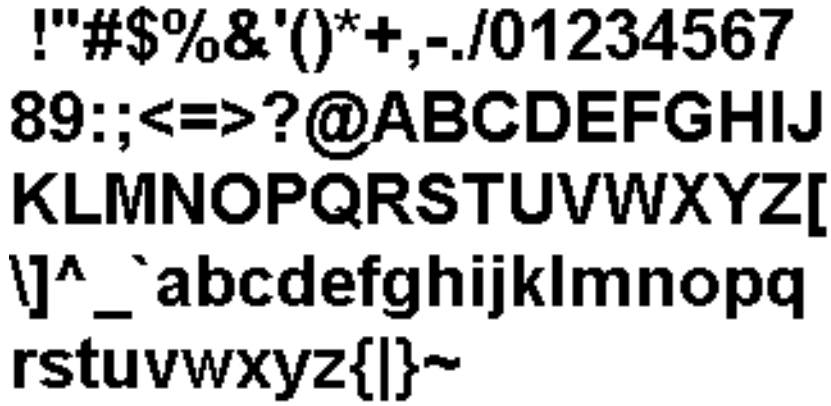
GUI_Font32_ASCII



GUI_Font32_1



GUI_Font32B_ASCII



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_Font32B_1

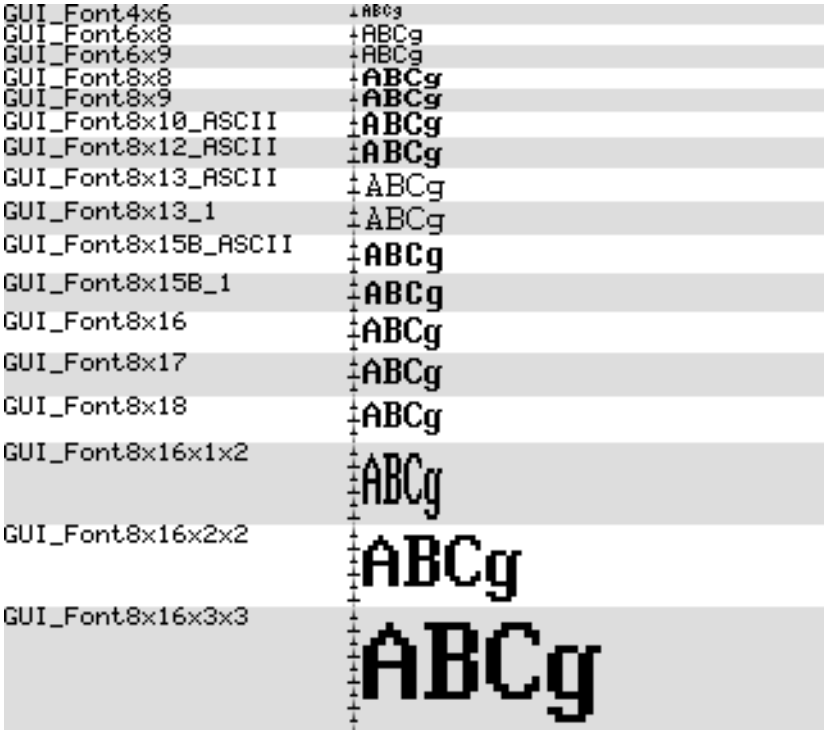


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.15.5 Monospaced fonts

9.15.5.1 Overview

The following screenshot gives an overview of all available monospaced fonts:



9.15.5.2 Measurement, ROM size and used files

The following table shows the measurement, ROMsize and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_Font4x6	F: 6, B: 5, C: 5, L: 4, U: 1	620	F4x6.c
GUI_Font6x8	F: 8, B: 7, C: 7, L: 5, U: 1	1840	F6x8.c
GUI_Font6x9	F: 9, B: 7, C: 7, L: 5, U: 2	1840 (same ROM location as GUI_Font6x8)	F6x8.c
GUI_Font8x8	F: 8, B: 7, C: 7, L: 5, U: 1	1840	F8x8.c
GUI_Font8x9	F: 9, B: 7, C: 7, L: 5, U: 2	1840 (same ROM location as GUI_Font8x8)	F8x8.c
GUI_Font8x10_ASCII	F: 10, B: 9, C: 9, L: 7, U: 1	1770	F8x10_ASCII.c
GUI_Font8x12_ASCII	F: 12, B: 10, C: 9, L: 6, U: 2	1962	F8x12_ASCII.c
GUI_Font8x13_ASCII	F: 13, B: 11, C: 9, L: 6, U: 2	2058	F8x13_ASCII.c
GUI_Font8x13_1	F: 13, B: 11, C: 9, L: 6, U: 2	2058+ 2070	F8x13_ASCII.c F8x13_1.c
GUI_Font8x15B_ASCII	F: 15, B: 12, C: 9, L: 7, U: 3	2250	F8x15_ASCII.c
GUI_Font8x15B_1	F: 15, B: 12, C: 9, L: 7, U: 3	2250+ 2262	F8x15B_ASCII.c F8x15B_1.c
GUI_Font8x16	F: 16, B: 12, C: 10, L: 7, U: 4	3304	F8x16.c

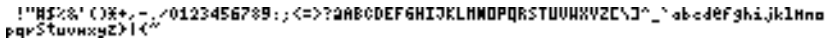
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Font name	Measurement	ROM size in bytes	Used files
GUI_Font8x17	F: 17, B: 12, C: 10, L: 7, U: 5	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x18	F: 18, B: 12, C: 10, L: 7, U: 6	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16x1x2	F: 32, B: 24, C: 20, L: 14, U: 8	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16x2x2	F: 32, B: 24, C: 20, L: 14, U: 8	3304 (same ROM location as GUI_Font8x16)	F8x16.c
GUI_Font8x16x3x3	F: 48, B: 36, C: 30, L: 21, U: 12	3304 (same ROM location as GUI_Font8x16)	F8x16.c

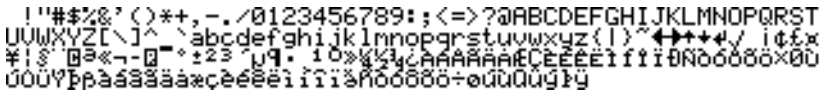
9.15.5.3 Characters

The following shows all characters of all monospaced standard fonts:

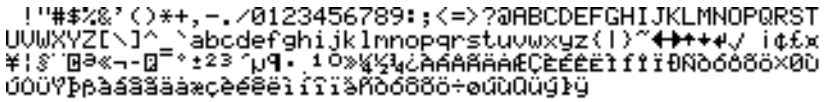
GUI_Font4x6



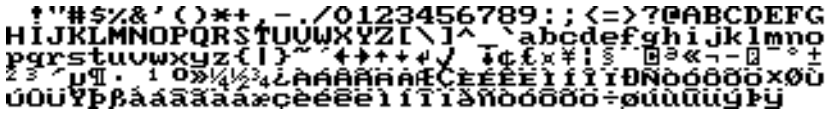
GUI_Font6x8



GUI_Font6x9



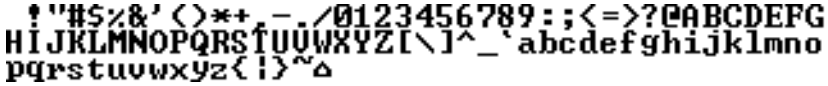
GUI_Font8x8



GUI_Font8x9

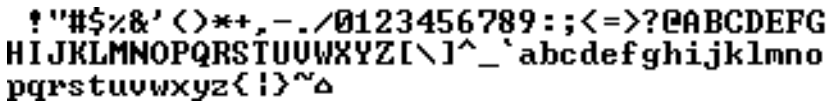


GUI_Font8x10_ASCII

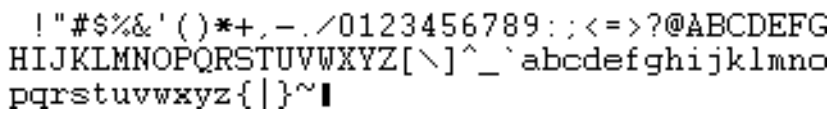


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

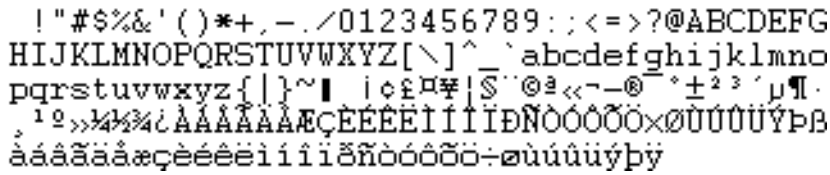
GUI_Font8x12_ASCII



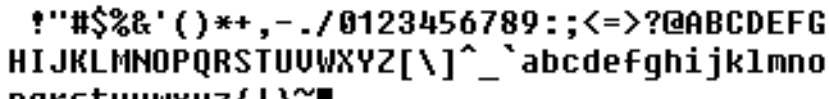
GUI_Font8x13_ASCII



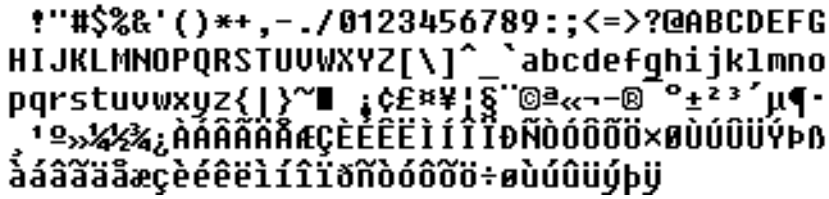
GUI_Font8x13_1



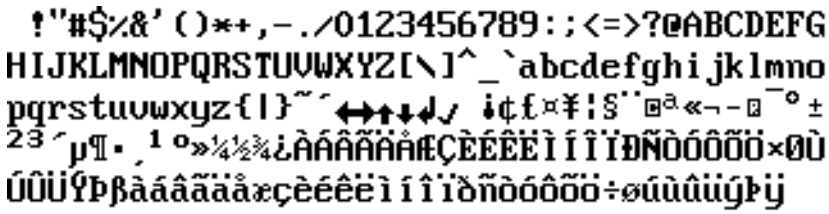
GUI_Font8x15B_ASCII



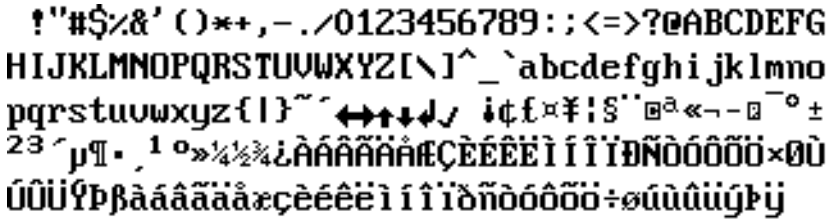
GUI_Font8x15B_1



GUI_Font8x16

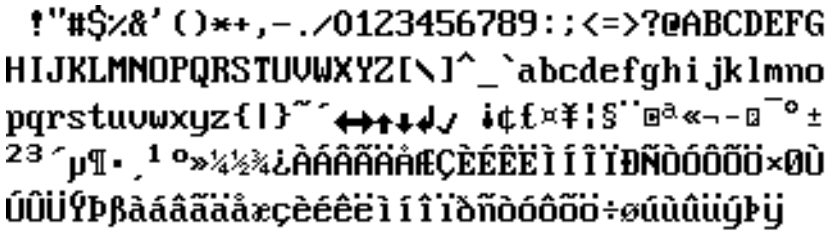


GUI_Font8x17



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

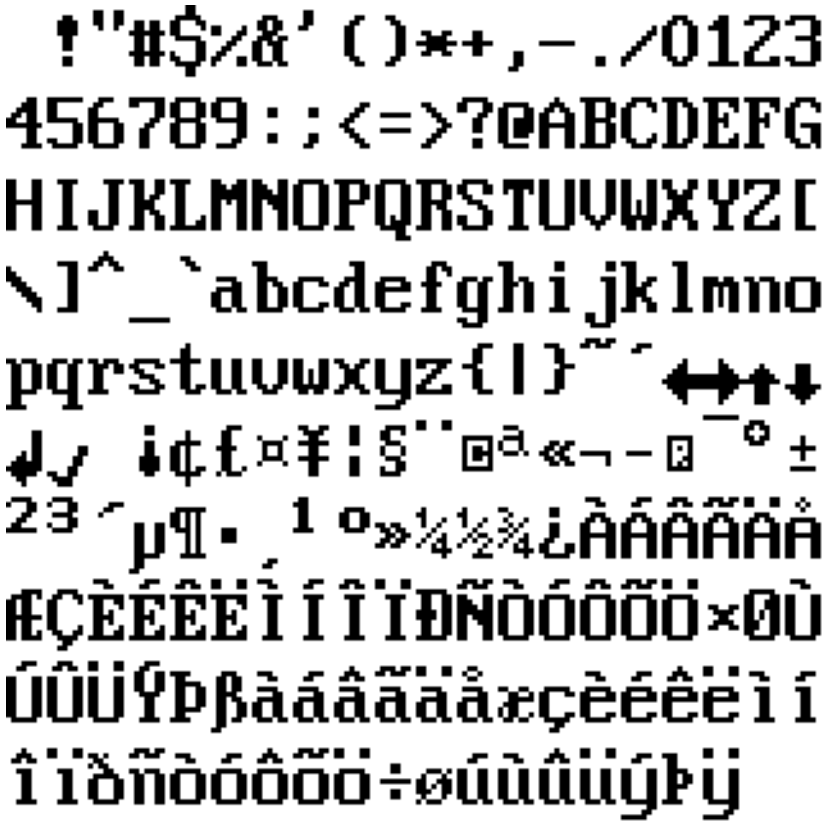
GUI_Font8x18



GUI_Font8x16x1x2

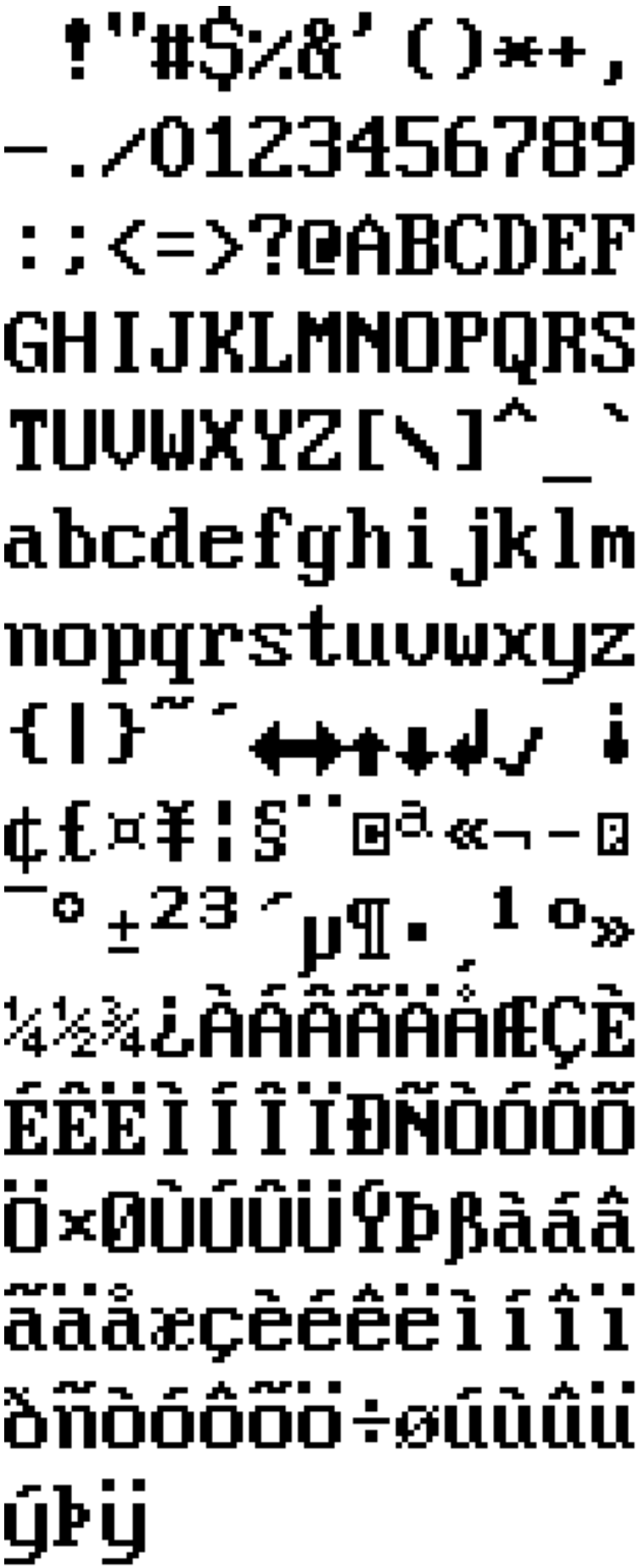


GUI_Font8x16x2x2



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_Font8x16x3x3

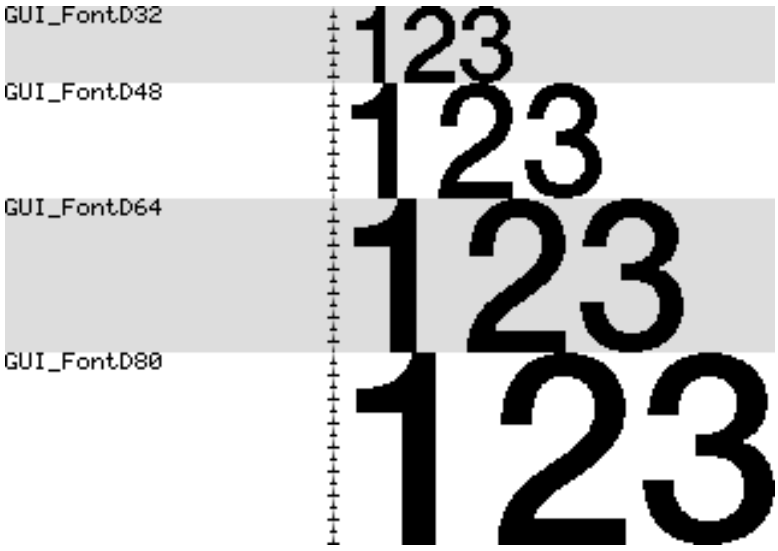


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.15.6 Digit fonts (proportional)

9.15.6.1 Overview

The following screenshot gives an overview of all available proportional digit fonts:



9.15.6.2 Measurement, ROM size and used files

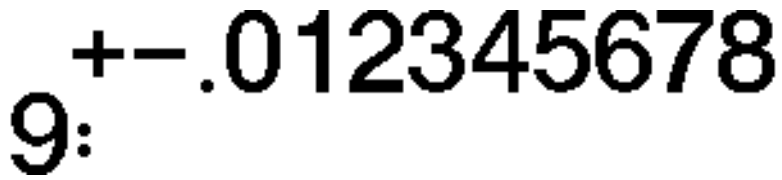
The following table shows the measurement, ROMsize and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_FontD32	F: 32, C: 31	1574	FD32.c
GUI_FontD48	F: 48, C: 47	3512	FD48.c
GUI_FontD64	F: 64, C: 63	5384	FD64.c
GUI_FontD80	F: 80, C: 79	8840	FD80.c

9.15.6.3 Characters

The following shows all characters of all proportional digit fonts:

GUI_FontD32

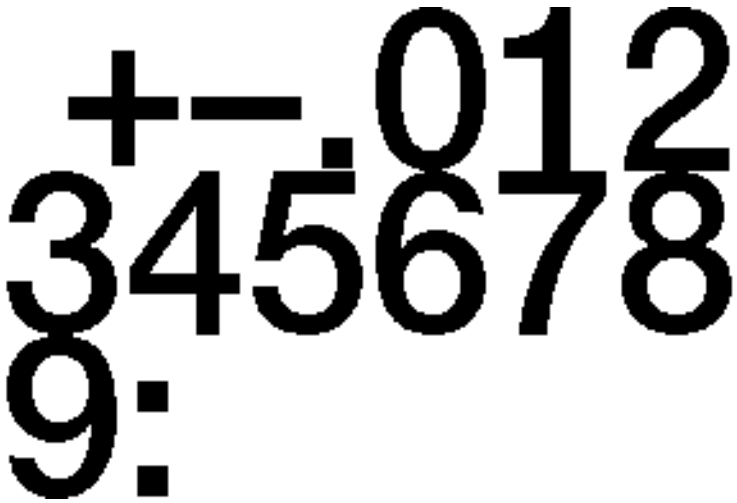


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

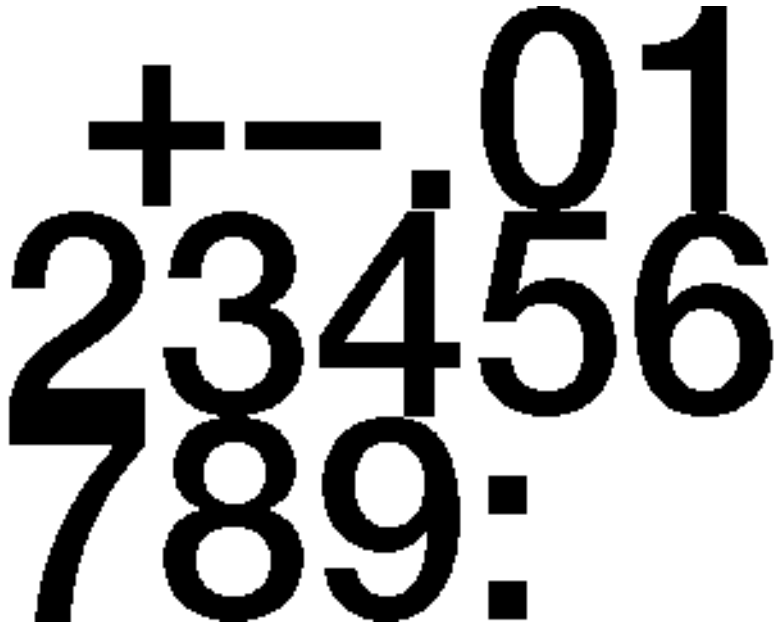
GUI_FontD48



GUI_FontD64



GUI_FontD80

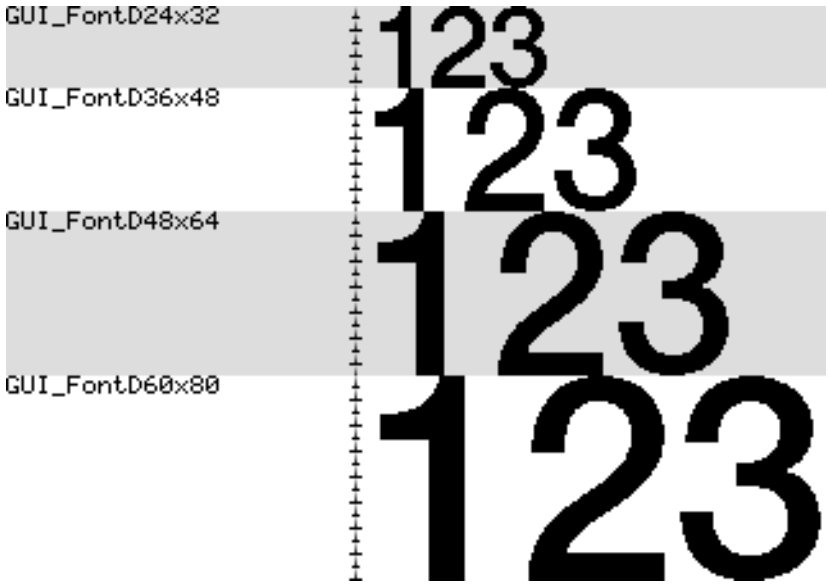


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

9.15.7 Digit fonts (monospaced)

9.15.7.1 Overview

The following screenshot gives an overview of all available monospaced digit fonts:



9.15.7.2 Measurement, ROM size and used files

The following table shows the measurement, ROMsize and used files of the fonts:

Font name	Measurement	ROM size in bytes	Used files
GUI_FontD24x32	F: 32, C: 31	1606	FD24x32.c
GUI_FontD36x48	F: 48, C: 47	3800	FD36x48.c
GUI_FontD48x64	F: 64, C: 63	5960	FD48x60.c
GUI_FontD60x80	F: 80, C: 79	9800	FD60x80.c

9.15.7.3 Characters

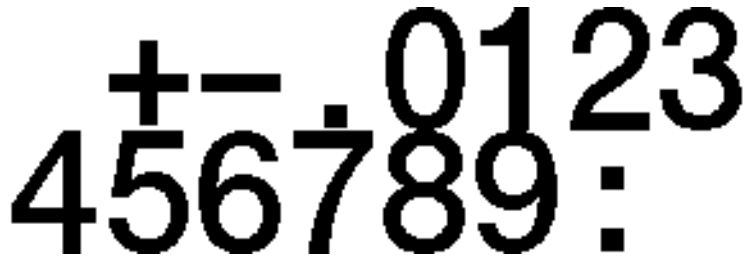
The following shows all characters of all monospaced digit fonts:

GUI_FontD24x32

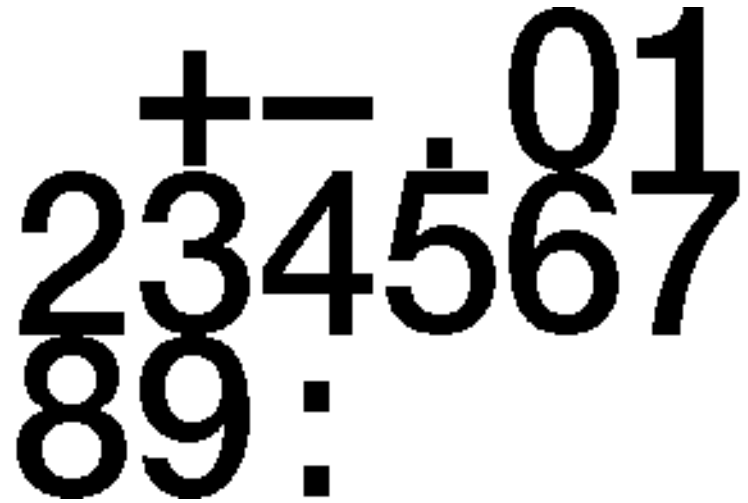


 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

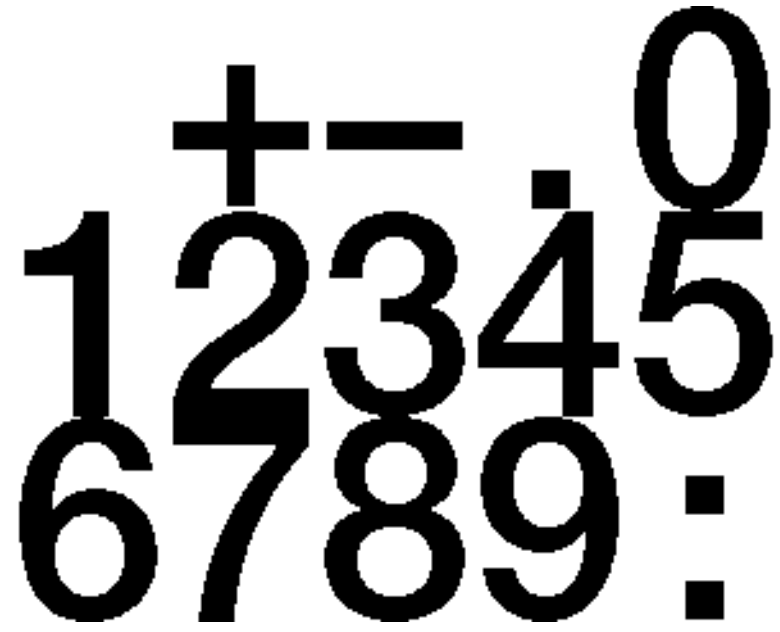
GUI_FontD36x48



GUI_FontD48x64



GUI_FontD60x80



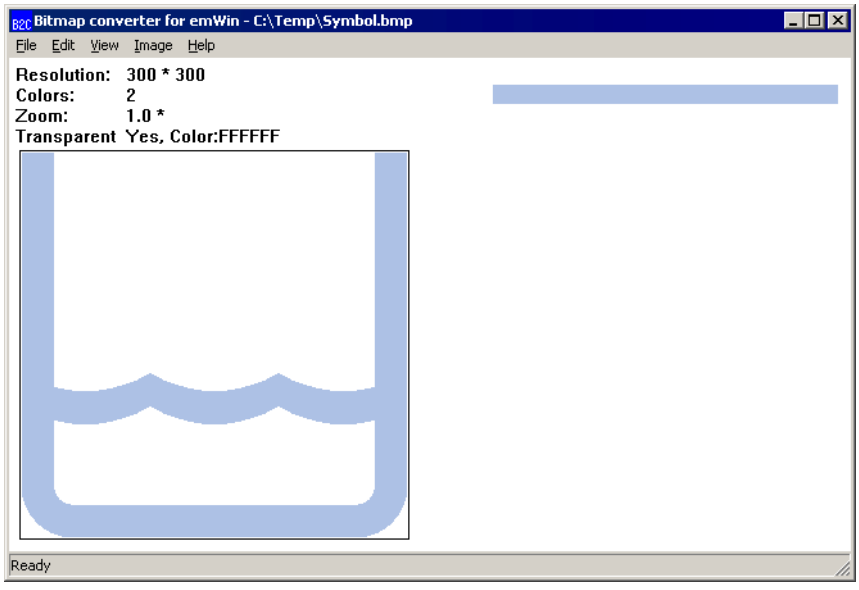
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

10.1 What it does

The bitmap converter is primarily intended as a tool to convert bitmaps from a PC format to a "C" file. Bitmaps which can be used with emWin are normally defined as `GUI_BITMAP` structures in "C". The structures -- or rather the picture data which is referenced by these structures -- can be quite large. It is time-consuming and inefficient to generate these bitmaps manually. We therefore recommend using the bitmap converter, which automatically generates "C" files from bitmaps.
It also features color conversion, so that the resulting "C" code is not unnecessarily large. You would typically reduce the number of bits per pixel in order to reduce memory consumption. The bitmap converter displays the converted image.
A number of simple functions can be performed with the bitmap converter, including scaling the size, flipping the bitmap horizontally or vertically, rotating it, and inverting the bitmap indices or colors (these features can be found under the `Image` menu). Any further modifications to an image must be made in a bitmap manipulation program such as Adobe Photoshop or Corel Photopaint. It usually makes the most sense to perform any image modifications in such a program, using the bitmap converter for converting purposes only.

10.2 Loading a bitmap

10.2.1 Supported 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.
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 a display which uses the same palette as the bitmap.
"C" files may be generated as "C with palette", "C without palette", "C with palette, compressed" or "C without palette, compressed". For more information on compressed files, see the section "Compressed bitmaps" as well as the example at the end of the chapter.

10.3.1 Supported bitmap formats

The following table shows the currently available output formats for "C" files:

Format	Color depth	Compression	Transparency	Palette
1 bit per pixel	1bpp	no	yes	yes
2 bits per pixel	2bpp	no	yes	yes
4 bits per pixel	4bpp	no	yes	yes
8 bits per pixel	8bpp	no	yes	yes
Compressed, RLE4	4bpp	yes	yes	yes
Compressed, RLE8	8bpp	yes	yes	yes
High color 555	15bpp	no	no	no
High color 555, red and blue swapped	15bpp	no	no	no
High color 565	16bpp	no	no	no
High color 565, red and blue swapped	16bpp	no	no	no
High color 565, compressed	16bpp	yes	no	no
High color 565, red and blue swapped, compressed	16bpp	yes	no	no
True color 888	24bpp	no	no	no
True color 8888 with alpha blending	32bpp	no	yes	no

10.3.2 Palette information

A bitmap palette is an array of 24 bit RGB color entries. Bitmaps with a color depth from 1 - 8 bpp can be saved with (device independent bitmap, DIB) or without palette information (device dependent bitmap DDB).

Device independent bitmaps (DIB)

The color information is stored in the form of an index into the color array. Before emWin draws a DIB, it converts the 24 bit RGB colors of the bitmap palette into color indices of the hardware palette. The advantage of using DIBs is that they are hardware independent and can be drawn correctly on systems with different color configurations. The disadvantages are the additional ROM requirement for the palette and the slower performance because of the color conversion.

Device dependent bitmaps (DDB)

The pixel information of a DDB is the index of the displays hardware palette. No conversion needs to be done before drawing a DDB. The advantages are less ROM requirement and a better performance. The disadvantage is that these bitmaps can not be displayed correctly on systems with other color configurations.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

10.3.3 Transparency

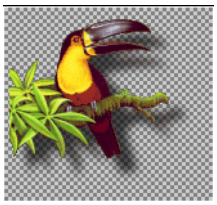
A palette based bitmap can be converted to a transparent bitmap. Transparency means each pixel with index 0 will not produce any output. The command `Image/Transparency` can be used to select the color which should be used for transparency. After selecting the transparent color, the pixel indices of the image will be recalculated, so that the selected color is on position 0 of the bitmap palette. When saving the bitmap file as 'C' file, it will be saved with the transparency attribute.

10.3.4 Alpha blending

Alpha blending is a method of combining an image with the background to create the effect of semi transparency. The alpha value of a pixel determines its transparency. The color of a pixel after drawing the bitmap is a blend of the former color and the color value in the bitmap. In emWin logical colors are handled as 32 bit values. The lower 24 bits are used for the color information and the upper 8 bits are used to manage the alpha value. An alpha value of 0 means the image is opaque and a value of 0xFF means completely transparent. Whereas BMP and GIF files do not support alpha blending PNG files support alpha blending. So the easiest way to create bitmap files with alpha blending is to load a PNG file. When working with BMP and/or GIF files the bitmap converter initially has no information about the alpha values.

Loading a PNG file

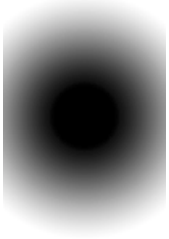
This is the most recommended way for creating bitmaps with an alpha mask:

After loading


The PNG file contains all required information.

Loading the alpha values from an alpha mask bitmap

This method loads the alpha values from a separate file. Black pixels of the alpha mask file means opaque and white means transparent. The following table shows a sample:

Starting point	Alpha mask	Result
		

The command `File/Load Alpha Mask` can be used for loading an alpha mask.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Creating the alpha values from two bitmaps

This method uses the difference between the pixels of two pictures to calculate the alpha values. The first image should show the item on a black background. The second image should show the same on a white background. The following table shows a sample of how to create the alpha values using the command `File/Create Alpha`:

Starting point	Black background	White background	Result
			

The command `File/Create Alpha` can be used tor creating the alpha values.

10.3.5 Selecting the best format

emWin supports various formats for the generated "C" file. It depends on several conditions which will be the 'best' format and there is no general rule to be used. Color depth, compression, palette and transparency affect the drawing performance and/or ROM requirement of the bitmap.

Color depth

In general the lower the color depth the smaller the ROM requirement of the bitmap. Each display driver has been optimized for drawing 1bpp bitmaps (text) and bitmaps with the same color depth as the display.

Compression

The supported RLE compression method has the best effect on bitmaps with many horizontal sequences of equal-colored pixels. Details later in this chapter. The performance is typically slightly slower than drawing uncompressed bitmaps.

Palette

The ROM requirement of a palette is 4 bytes for each color. So a palette of 256 colors uses 1kB. Furthermore emWin needs to convert the colors of the palette before drawing the bitmap. Advantage: Bitmaps are device independent meaning they can be displayed on any display, independent of its color depth and format.

Transparency

The ROM requirement of transparent bitmaps is the same as without transparency. The performance is with transparency slightly slower than without.

High color and true color bitmaps

Special consideration is required for bitmaps in these formats. Generally the use of these formats only make sense on displays with a color depth of 15 bits and above. Further it is strongly recommended to save the 'C' files in the exact same format used by the hardware. Please note that using the right format will have a positive effect on the drawing performance. If a high color bitmap for example should be shown on a system with a color depth of 16bpp which has the red and blue components swapped, the best format is 'High color 565, red and blue swapped'. Already a slightly other format has the effect, that each pixel needs color conversion, whereas a bitmap in the right format can be rendered very fast without color conversion. The difference of drawing performance in this case can be factor 10 and more.

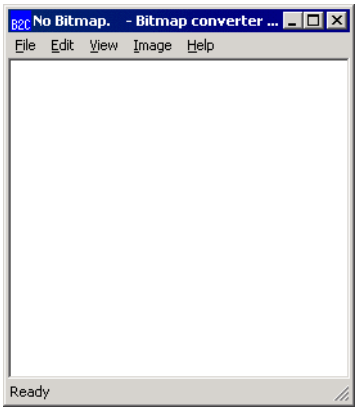
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

10.3.6 Saving the file

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

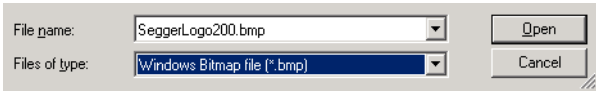
Step 1: Start the application.

The bitmap converter is opened showing an empty window.

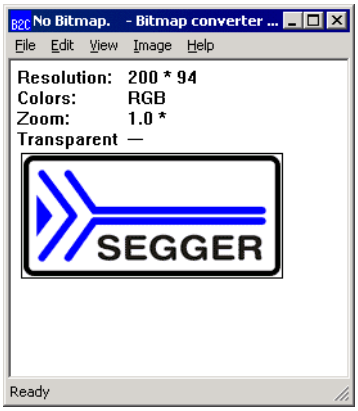


Step 2: Load a bitmap into the bitmap converter.

Choose `File/Open`.
Locate the document you want to open and click `Open` (must be a `.bmp` file).
In this example, the file `SeggerLogo200.bmp` is chosen.



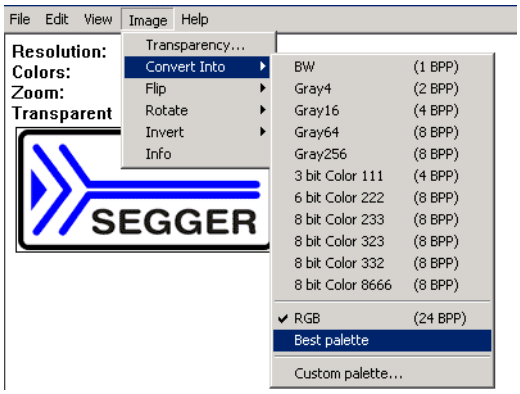
The bitmap converter displays the loaded bitmap.



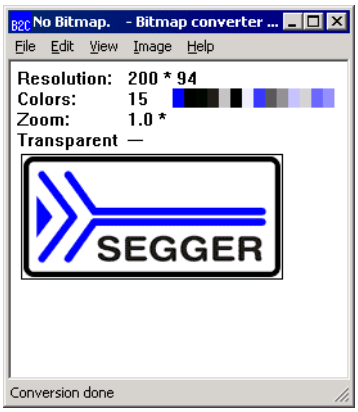
In this example, the loaded bitmap is in full-color mode. It must be converted to a palette format before a "C" file can be generated.

Step 3: Convert the image if necessary.

Choose `Image/Convert Into`.
Select the desired palette.
In this example, the option `Best palette` is chosen.

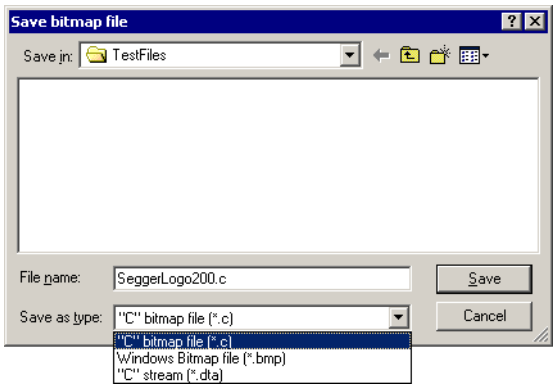
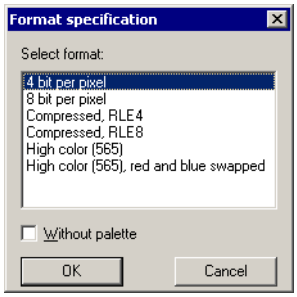


The bitmap converter displays the converted bitmap.



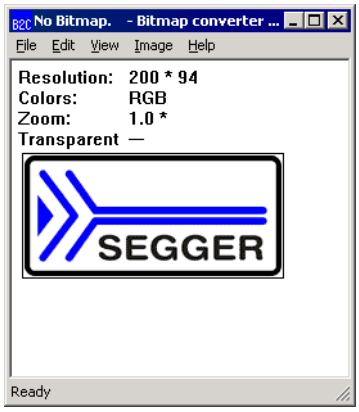
The image is unchanged in terms of appearance, but uses less memory since a palette of only 15 colors is used instead of the full-color mode. These 15 colors are the only ones actually required to display this particular image.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.	

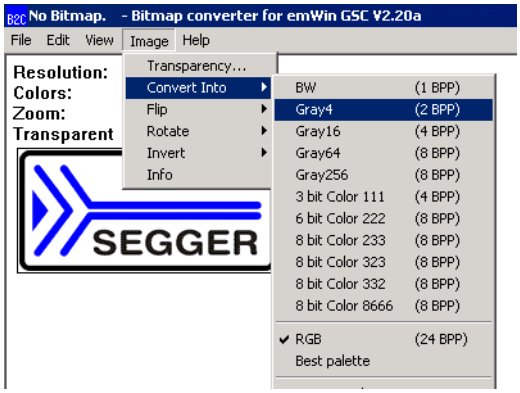
10.4 Color conversion

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.	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

Converting bitmaps to a custom palette and saving them without palette information can save memory and can increase the performance of bitmap drawing operations.

More efficien memory utilisation

Per default each bitmap contains its own palette. Even the smallest bitmaps can contain a large palette with up to 256 colors. In many cases only a small fraction of the palette is used by the bitmap. If using many of these bitmaps the amount of memory used by the palettes can grow rapidly.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

So it can save much ROM if converting the bitmaps used by emWin to the available hardware palette and saving them as (D)evice (D)ependent (B)itmaps without palette information.

Better bitmap drawing performance

Before emWin draws a bitmap, it needs to convert each device independent bitmap palette to the available hardware palette. This is required because the pixel indices of the bitmap file are indices into the device independent bitmap palette and not to the available hardware palette.
Converting the bitmap to a DDB means that color conversion at run time is not required and speeds up the drawing.

10.6.1 Saving a palette file

The bitmap converter can save the palette of the currently loaded bitmap into a palette file which can be used for converting other bitmaps with the command `Image/Convert Into/Custom palette`. This requires that the current file is a palette based file and not a RGB file. To save the palette the command `File/Save palette...` can be used.

10.6.2 Palette file format

Custom palette files are simple files defining the available colors for conversion. They contain the following:

- Header (8 bytes).
- NumColors (U32, 4 bytes).
- 0 (4 bytes).
- U32 Colors[NumColors] (NumColors*4 bytes, type GUI_COLOR).

Total file size is therefore: 16+(NumColors*4) bytes. A custom palette file with 8 colors would be 16+(8*4) = 48 bytes. At this point, a binary editor must be used in order to create such a file.
The maximum number of colors supported is 256; the minimum is 2.

Sample

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.

10.6.3 Palette files for fixed palette modes

Using the custom palette feature can even make sense with the most common used fixed palette modes, not only with custom hardware palettes. For the most palette based fixed palette modes a palette file can be found in the folder `Sample\Palette`.

10.6.4 Converting a bitmap

The command `Image/Convert Into/Custom palette` should be used for converting the currently loaded bitmap to a custom palette. The bitmap converter tries to find the narest color of the palette file for each pixel of the currently loaded bitmap.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

10.7 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 -convertintobestpalette -saveaslogo,1 -exit
```

Note that while the file to be loaded into the bitmap converter always includes its `.bmp` extension, no file extension is written in the `-saveas` command. An integer is used instead to specify the desired file type. The number 1 in the `-saveas` command above designates "C with palette". The `-exit` command automatically closes the program upon completion. See the table below for more information.

10.7.2 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
`-convertintobw`	Convert to BW.
`-convertintogray4`	Convert to Gray4.
`-convertintogray16`	Convert to Gray16.
`-convertintogray64`	Convert to Gray64.
`-convertintogray256`	Convert to Gray256.
`-convertinto111`	Convert to 111.
`-convertinto222`	Convert to 222.
`-convertinto233`	Convert to 233.
`-convertinto323`	Convert to 323.
`-convertinto332`	Convert to 332.
`-convertinto8666`	Convert to 8666.
`-convertintorgb`	Convert to RGB.
`-convertintobestpalette`	Convert to best palette.
`-convertintocustompalette<filename>`	Convert to a custom palette.
`<filename>`	User-specified filename of desired custom palette.
`-exit`	Terminate PC program automatically.
`-fliph`	Flip image horizontally.
`-flipv`	Flip image vertically.
`-help`	Display this box.
`-invertindices`	Invert indices.
`-rotate90cw`	Rotate image by 90 degrees clockwise.
`-rotate90cc`	Rotate image by 90 degrees counter-clockwise.
`-rotate180`	Rotate image by 180 degrees.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Command	Explanation
-saveas<filename>,<type>[,<fmt>[,<noplt>]]	Save file as filename.
<filename>	User-specified file name including the file extension.
<type>	Must be an integer from 1 to 6 as follows: 1: "C" with palette (.c file) 2: Windows Bitmap file (.bmp file) 3: "C" stream (.dta file) 4: GIF format (.gif file)
<fmt>	Specifies the bitmap format (only if type == 1): 1: 1 bit per pixel 2: 2 bits per pixel 4: 4 bits per pixel 5: 8 bits per pixel 6: RLE4 compression 7: RLE8 compression 8: High color 565 9: High color 565, red and blue swapped 10: High color 555 11: High color 555, red and blue swapped 12: RLE16 compression 13: RLE16 compression, red and blue swapped 17: True color 24bpp If this parameter is not given, the bitmap converter uses the following default formats in dependence of the number of colors of the bitmap: Number of colors <= 2: 1 bit per pixel Number of colors <= 4: 2 bits per pixel Number of colors <= 16: 4 bits per pixel Number of colors <= 256: 8 bits per pixel RGB: High color 565
<noplt>	Saves the bitmap with or without palette (only if type == 1) 0: Save bitmap with palette (default) 1: Save bitmap without palette
-transparency<RGB-Color>	Sets the transparent color.
<RGB-Color>	RGB color which should be used as transparent color.
-?	Display this box.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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:   0xBBGGRR
*/

static GUI_CONST_STORAGE GUI_COLOR ColorsLogo[] = {
     0xFFFFFF,0xFF0000,0x000000,0x0F0F0F
    ,0x1C1F23,0xC3C3C3,0x020202,0xFBEFEF
    ,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,
0x0A, 0x22, 0x22, 0x22, ... , 0xA0,
0xC6, 0x22, 0x23, 0x95, ... , 0x6C,
.
.
.
0xC6, 0x22, 0x23, 0xA5, ... , 0x6C,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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:   0xBBGGRR
*/

static GUI_CONST_STORAGE GUI_COLOR ColorsLogoCompressed[] = {
     0xFFFFFF,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]
};
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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,


  0};  /* 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 *** */
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 0xFFFFFF, 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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		0xFFFFFF
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

11.3 Fixed palette modes

The following table lists the available fixed palette color modes and the necessary identifiers which need to be used when creating a driver- or a memory device. Detailed descriptions follow.

Identifier	No. available colors	Mask
GUICC_1	2 (black and white)	0x01
GUICC_2	4 (grayscales)	0x03
GUICC_4	16 (grayscales)	0x0F
GUICC_5	32 (grayscales)	0x1F
GUICC_111	8	BGR
GUICC_M111	8	RGB
GUICC_222	64	BBGGRR
GUICC_M222	64	RRGGBB
GUICC_233	256	BBGGGRRR
GUICC_M233	256	RRGGGBBB
GUICC_323	256	BBBGGRRR
GUICC_M323	256	RRRGGBBB
GUICC_332	256	BBBGGGRR
GUICC_M332	256	RRRGGGBB
GUICC_44412	4096	0000BBBBGGGGRRRR
GUICC_M44412	4096	0000RRRRGGGGBBBB
GUICC_444121	4096	BBBBGGGGRRRR0000
GUICC_44416	4096	0BBBB0GGGG0RRRR0
GUICC_M44416	4096	0RRRR0GGGG0BBBB0
GUICC_555	32768	0BBBBBGGGGGRRRRR
GUICC_M555	32768	0RRRRRGGGGGBBBBB
GUICC_556	65536	BBBBBGGGGGRRRRRR
GUICC_M556	65536	RRRRRGGGGGBBBBBB
GUICC_565	65536	BBBBBGGGGGGRRRRR
GUICC_M565	65536	RRRRRGGGGGGBBBBB
GUICC_655	65536	BBBBBBGGGGGRRRRR
GUICC_M655	65536	RRRRRRGGGGGBBBBB
GUICC_666	262144	BBBBBBGGGGGGRRRRRR
GUICC_822216	256	0xFF
GUICC_84444	240	0xFF
GUICC_8666	232	0xFF
GUICC_86661	233 (232 + transparency)	0xFF
GUICC_888	16777216	BBBBBBBBGGGGGGGGRRRRRRRR
GUICC_M888	16777216	RRRRRRRRGGGGGGGGBBBBBBBB
GUICC_8888	16777216 + 8 bit alpha blending	AAAAAAAABBBBBBBBGGGGGGGGRRRRRRRR

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Identifier	No. available colors	Mask
GUICC_M8888	16777216 + 8 bit alpha blending	AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB
GUICC_0	x	x

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

11.4 Detailed fixed palette mode description

The following gives a detailed description of the available colors in each predefined fixed palette mode.

GUICC_1: 1 bpp (black and white)

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

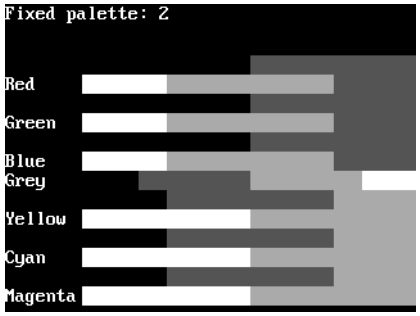


Available colors: 2:



GUICC_2: 2 bpp (4 grayscales)

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

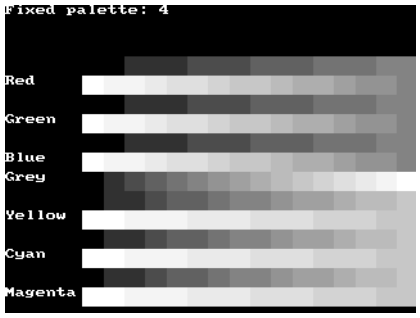


Available colors: 2 x 2 = 4:



GUICC_4: 4 bpp (16 grayscales)

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

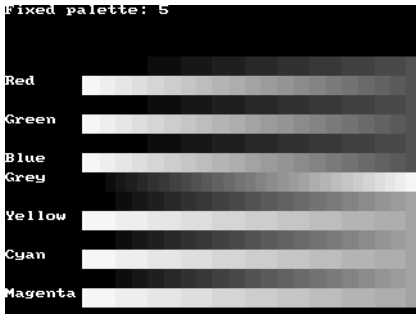


Available colors: 2 x 2 x 2 x 2 = 16:



GUICC_5: 5 bpp (32 grayscales)

Use of this mode is necessary for monochrome displays with 5 bits per pixel.



Available colors: 2 x 2 x 2 x 2 x 2 = 32:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUICC_111: 3 bpp (2 levels per color)



Use this mode if the basic 8 colors are enough, if your hardware supports only one bit per pixel and color or if you do not have sufficient video memory for a higher color depth.
Color mask: BGR

Available colors: 2 x 2 x 2 = 8:



GUICC_M111: 3 bpp (2 levels per color), red and blue swapped

Use this mode if the basic 8 colors are enough, if your hardware supports only one bit per pixel and color or if you do not have sufficient video memory for a higher color depth. The available colors are the same as those in 111 mode.
Color mask: RGB
Available colors: 2 x 2 x 2 = 8:



GUICC_222: 6 bpp (4 levels per color)



This mode is a good choice if your hardware does not have a palette for every individual color. 2 bits per pixel and color are reserved; usually 1 byte is used to store one pixel.
Color mask: BBGGRR

Available colors: 4 x 4 x 4 = 64:

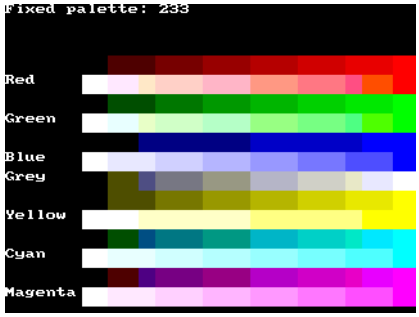


GUICC_M222: 6 bpp (4 levels per color), red and blue swapped

This mode is a good choice if your hardware does not have a palette for every individual color. 2 bits per pixel and color are reserved; usually 1 byte is used to store one pixel. The available colors are the same as those in 222 mode.
Color mask: RRGGBB
Available colors: 4 x 4 x 4 = 64:



GUICC_233: 8 bpp



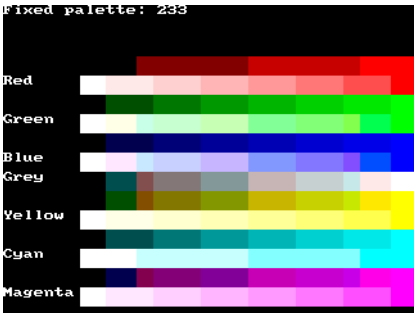
This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. As shown in the picture, the result is 8 grades for green and red and 4 grades for blue. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: BBGGGRRR

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Available colors: 4 x 8 x 8 = 256:



GUICC_M233: 8 bpp, red and blue swapped

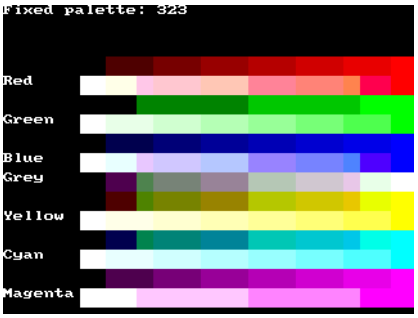


This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. The result is 8 grades for green and blue and 4 grades for red. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: RRGGGBBB

Available colors: 4 x 8 x 8 = 256:



GUICC_323: 8 bpp



This mode supports 256 colors. 3 bits are used for the red and blue components of the color and 2 bits for the green component. As shown in the picture, the result is 8 grades for blue and red and 4 grades for green. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: BBBGGRRR

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Available colors: 8 x 4 x 8 = 256:



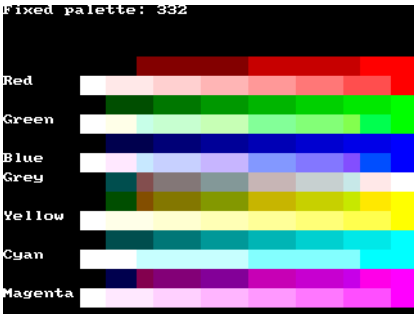
GUICC_M323: 8 bpp, red and blue swapped

This mode supports 256 colors. 3 bits are used for the red and blue components of the color and 2 bits for the green component. The available colors are the same as those in 323 mode. The result is 8 grades for red and blue and 4 grades for green. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: RRRGGBBB
Available colors: 8 x 4 x 8 = 256:



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

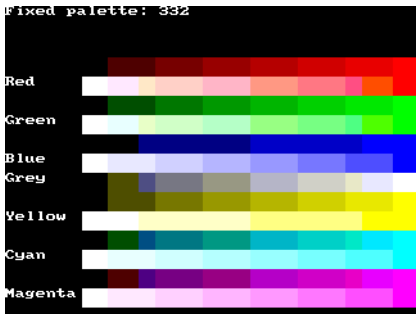


Available colors: 8 x 8 x 4 = 256:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUICC_332: 8 bpp, red and blue swapped

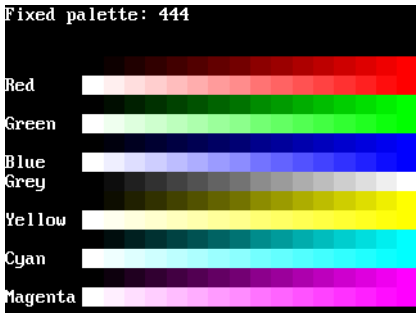


This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. The result is 8 grades for red and green and only 4 grades for blue. We discourage the use of this mode because it do not contain real shades of gray.
Color mask: RRRGGGBB

Available colors: 8 x 8 x 4 = 256:



GUICC_44412:



The red, green and blue components are each 4 bits.
Color mask: 0000BBBBGGGGRRRR
Available colors: 16 x 16 x 16 = 4096.

GUICC_44416:

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: 0BBBB0GGGG0RRRR0
Available colors: 16 x 16 x 16 = 4096.

GUICC_44412: 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 x 16 x 16 = 4096.
Color mask: RRRRGGGGBBBB

GUICC_44416: red and blue swapped

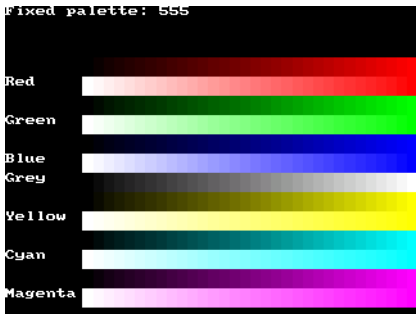
The red, green and blue components are each 4 bits. One bit between the color components is not used. The available colors are the same as those in 44412 mode.
Color mask: 0RRRR0GGGG0BBBB0
Available colors: 16 x 16 x 16 = 4096.

GUICC_444121:

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 x 16 x 16 = 4096.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUICC_555: 15 bpp

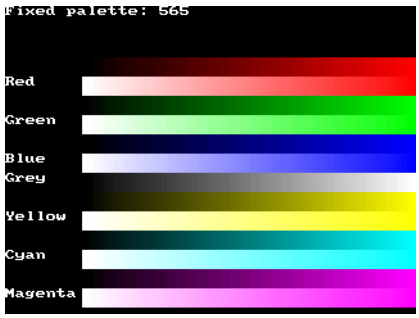


Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 15 bpp (such as SED1356 or SED13806). The red, green and blue components are each 5 bits.
Color mask: BBBBBGGGGGRRRRR
Available colors: 32 x 32 x 32 = 32768.

GUICC_M555: 15 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 15 bpp. The red, green and blue components are each 5 bits. The available colors are the same as those in 555 mode.
Color mask: RRRRRGGGGGBBBBB
Available colors: 32 x 32 x 32 = 32768.

GUICC_565: 16 bpp



Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The red and the blue component is 5 bits and the green component is 6 bit.
Color mask: BBBBBGGGGGGRRRRR
Available colors: 32 x 64 x 32 = 65536.

GUICC_M565: 16 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The available colors are the same as those in 565 mode.
Color sequence: RRRRRGGGGGGBBBBB
Available colors: 32 x 64 x 32 = 65536.

GUICC_556: 16 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The blue and the green component is 5 bit and the red component is 6 bit.
Color mask: BBBBBGGGGGRRRRRR
Available colors: 32 x 32 x 64 = 65536.

GUICC_M556: 16 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The red and the green component is 5 bit and the blue component is 6 bit.
Color mask: RRRRRGGGGGBBBBBB
Available colors: 32 x 32 x 64 = 65536.

GUICC_655: 16 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The red and the green component is 5 bit and the blue component is 6 bit.
Color mask: BBBBBBGGGGGRRRRR
Available colors: 64 x 32 x 32 = 65536.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUICC_M655: 16 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 16 bpp. The blue and the green component is 5 bit and the red component is 6 bit.
Color mask: RRRRRRGGGGGBBBBB
Available colors: 64 x 32 x 32 = 65536.

GUICC_666: 18 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 18 bpp. The red, green and the blue component is 6 bit.
Color mask: BBBBBBGGGGGGRRRRRR
Available colors: 64 x 64 x 64 = 262144.

GUICC_M666: 18 bpp, red and blue swapped

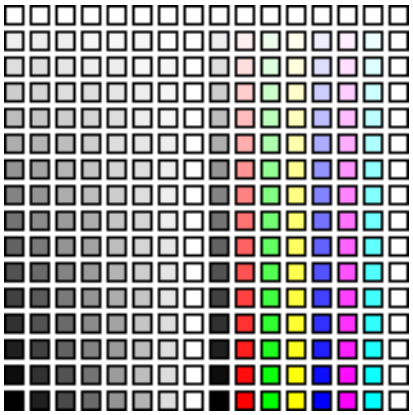
Use of this mode is necessary for a display controller that supports RGB colors with a color-depth of 18 bpp. The red, green and the blue component is 6 bit.
Color mask: RRRRRRGGGGGGBBBBBB
Available colors: 64 x 64 x 64 = 262144.

GUICC_822216: 8 bpp, 2 levels per color + 8 grayscales + 16 levels of alpha blending

This mode can be used with a programmable color lookup table (LUT), supporting a total of 256 possible colors and alpha blending support. It supports the 8 basic colors, 8 grayscales and 16 levels of alpha blending for each color / grayscale. With other words it can be used if only a few colors are required but more levels of alpha blending.

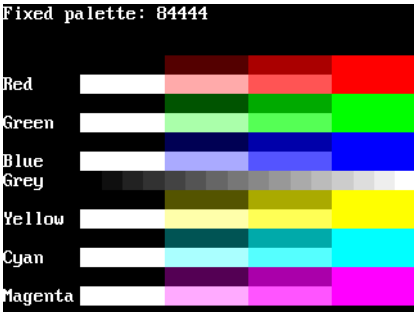
Available colors: (2 x 2 x 2 + 8) * 16 = 256





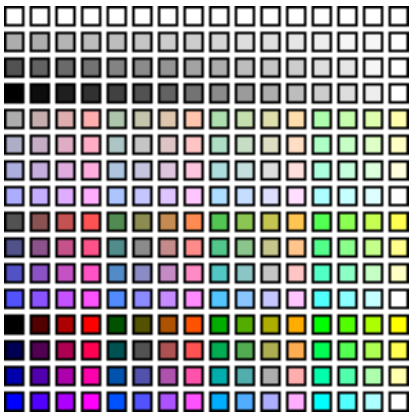
GUICC_84444: 8 bpp, 4 levels per color + 16 grayscales + 4(3) levels of alpha blending

This mode can be used with a programmable color lookup table (LUT), supporting a total of 256 possible colors and alpha blending support. 4 levels of intensity are available for each color, in addition to 16 grayscales and 4 levels of alpha blending for each color / grayscale. With other words it can be used if only a few levels of alpha blending are required and different shades of colors.



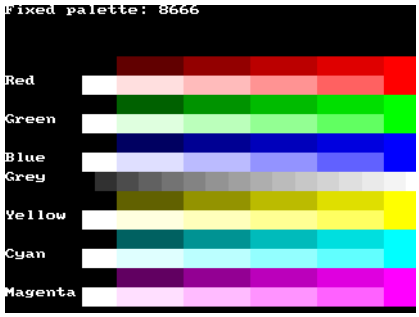
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Available colors: (4 x 4 x 4 + 16) * 3 = 240



GUICC_8666: 8bpp, 6 levels per color + 16 grayscales

This mode is most frequently used with a programmable color lookup table (LUT), supporting a total of 256 possible colors using a palette. The 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 x 6 x 6 + 16 = 232:



GUICC_86661: 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 x 6 x 6 + 16 = 232.



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUICC_888: 24 bpp



Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 24 bpp. The red, green and blue components are each 8 bits.
Color mask: BBBBBBBBGGGGGGGGRRRRRRRR
Available colors: 256 x 256 x 256 = 16777216.

GUICC_M888: 24 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 24 bpp. The red, green and blue components are each 8 bits.
Color mask: RRRRRRRRGGGGGGGGBBBBBBBB
Available colors: 256 x 256 x 256 = 16777216.

GUICC_8888: 32 bpp

Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 32 bpp, where the lower 3 bytes are used for the color components and the upper byte is used for alpha blending. The red, green, blue and alpha blending components are each 8 bits.
Color mask: AAAAAAAABBBBBBBBGGGGGGGGRRRRRRRR
Available colors: 256 x 256 x 256 = 16777216.

GUICC_M8888: 32 bpp, red and blue swapped

Use of this mode is necessary for a display controller that supports RGB colors with a color depth of 32 bpp, where the lower 3 bytes are used for the color components and the upper byte is used for alpha blending. The red, green, blue and alpha blending components are each 8 bits.
Color mask: AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB
Available colors: 256 x 256 x 256 = 16777216.

GUICC_0: Custom palette mode

(explained later in this chapter)

11.5 Application defined color conversion

If none of the fixed palette modes matches the need of color conversion this mode makes it possible to use application defined color conversion routines. The purpose of these routines is converting an RGB value into an index value for the hardware and vice versa.

Sample of defining custom color conversion routines

The following sample should explain how it works:

```
static unsigned _Color2Index_User(LCD_COLOR Color) {
  unsigned Index;
  /* Add code for converting the RGB value to an index value for the hardware */
  return Index;
}

static LCD_COLOR _Index2Color_User(unsigned Index) {
  LCD_COLOR Color;
  /* Add code for converting the index value into an RGB value */
  return Color;
}

static unsigned _GetIndexMask_User(void) {
  return 0xffff; /* Example for using 16 bits */
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
const LCD_API_COLOR_CONV LCD_API_ColorConv_User = {
  _Color2Index_User,
  _Index2Color_User,
  _GetIndexMask_User
};
```

The function LCD_Color2Index_User() is called by emWin if a RGB value should be converted into an index value for the display controller whereas the function LCD_Index2Color_User() is called if an index value should be converted into a RGB value.
LCD_GetIndexMask_User() should return a bit mask value, which has each bit set to 1 which is used by the display controller and unused bits should be set to 0. For example the index mask of GUICC_44416 mode is 0BBBB0GGGG0RRRR0, where 0 stands for unused bits. The bit mask for this mode is 0x7BDE.

Sample of using custom color conversion routines

As described in the chapter 'Configuration' a pointer to an API table is required for creating the display driver device. As shown in the sample above the API table consists of function pointers to the color conversion routines.
A good location for the API table and the color conversion routines is the configuration file LCDConf.c located in the Config folder. The routines can be used as follow in the function LCD_X_Config() which is responsible to dreate the display driver device:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, &LCD_API_ColorConv_User, 0, 0);
  .
  .
  .
}
```

11.6 Custom palette mode

If none of the fixed palette modes fulfils the requirements of the application emWin is able to use a custom palette. A custom palette simply lists all the available colors in the same order as they are used by the hardware. This means that no matter what colors 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.
A custom palette is typically used during the initialization in the function LCD_X_Config() which is responsible for creating and configuring the display driver device.

Sample

The following sample should show how a custom palette can be used. It passes the palette to the function:

```
static const LCD_COLOR _aColors_16[] = {
  0x000000,  0x0000FF,  0x00FF00,  0x00FFFF,
  0xFF0000,  0xFF00FF,  0xFFFF00,  0xFFFFFF,
  0x000000,  0x000080,  0x008000,  0x008080,
  0x800000,  0x800080,  0x808000,  0x808080,
};

static const LCD_PHYSPALETTE _aPalette_16 = {
  COUNTOF(_aColors_16), _aColors_16
};

void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  .
  .
  .
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  //
  // Set user palette data (only required if no fixed palette is used)
  //
  LCD_SetLUTEx(0, _aPalette_16);
}
```

11.7 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	
GUI_GetBkColor()	Return the current background color.
GUI_GetBkColorIndex()	Return the index of the current background color.
GUI_GetColor()	Return the current foreground color.
GUI_GetColorIndex()	Return the index of the current foreground color.
GUI_SetBkColor()	Set the current background color.
GUI_SetBkColorIndex()	Set the index of the current background color.
GUI_SetColor()	Set the current foreground color.
GUI_SetColorIndex()	Set the index of the current foreground color.
Index & color conversion	
GUI_CalcColorDist()	Returns the difference between 2 colors
GUI_CalcVisColorError()	Returns the difference to the next available color
GUI_Color2Index()	Convert color into color index.
GUI_Color2VisColor()	Returns the nearest available color
GUI_ColorIsAvailable()	Checks if given color is available
GUI_Index2Color()	Convert color index into color.

11.8 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

11.9 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: Difference = $(Red1 - Red0)^2 + (Green1 - Green0)^2 + (Blue1 - Blue0)^2$

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 12

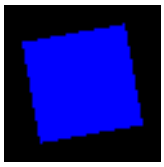
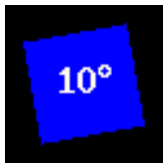
Memory Devices

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`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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:
1 bpp, 8 bpp, 16 bpp and 32 bpp.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Creating memory devices "compatible" to the display

There are two ways to create memory devices. If they are use to avoid flickering, a memory device compatible to the display is created. This "compatible" memory device needs to have the same or a higher color depth as the display. emWin automatically selects the "right" type of memory device for the display if the functions `GUI_MEMDEV_Create()`, `GUI_MEMDEV_CreateEx()` are used.
The Window manager, which also has the ability to use memory devices for some or all windows in the system, also uses these functions.
This way, the memory device with the lowest color depth (using the least memory) is automatically used.

Creating memory devices for other purposes

Memory devices of any type can be created using `GUI_MEMDEV_CreateFixed()`. A typical application would be the use of a memory device for printing as described later in this chapter.

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
32 bpp	18, 24 and 32 bpp	4 bytes / pixel: XSIZE * YSIZE * 4

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:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Number of required bytes = (111 * 2 + (111 + 7) / 8) * 33 = 7788 bytes

Memory usage with transparency support

If a memory device should support transparency it needs one additional byte / 8 pixels for internal management.

Color depth of memory device	System color depth (LCD_BITSPERPIXEL)	Memory usage
1 bpp	1 bpp	2 byte / 8 pixels: (XSIZE + 7) / 8 * YSIZE * 2
8 bpp	2, 4 and 8 bpp	1 bytes / pixel + 1 byte / 8 pixels: (XSIZE + (XSIZE + 7) / 8) * YSIZE
16 bpp	12 and 16 bpp	2 bytes / pixel + 1 byte / 8 pixels: (XSIZE * 2 + (XSIZE + 7) / 8) * YSIZE
32 bpp	18, 24 and 32 bpp	4 bytes / pixel + 1 byte / 8 pixels: (XSIZE * 4 + (XSIZE + 7) / 8) * YSIZE

Example:

A memory device of 200 pixels in X and 50 pixels in Y should be created. It should be compatible to a display with a color depth of 4bpp and should support transparency. The required number of bytes can be calculated as follows:

Number of required bytes = (200 + (200 + 7) / 8) * 50 = 11250 bytes

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 <= 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_DrawPerspectiveX()	Draws the given memory device perspectively distorted into the current selected device.
GUI_MEMDEV_GetDataPtr()	Returns a pointer to the data area for direct manipulation.
GUI_MEMDEV_GetXSize()	Returns the X-size (width) of memory device.
GUI_MEMDEV_GetYSize()	Returns the Y-size (height) of memory device.
GUI_MEMDEV_MarkDirty()	Marks the given area as containing pixels to be drawn.
GUI_MEMDEV_ReduceYSize()	Reduces Y-size of memory device.
GUI_MEMDEV_Rotate()	Rotates and scales a memory device and writes the result into a memory device using the 'nearest neighbour' method.
GUI_MEMDEV_RotateHQ()	Rotates and scales a memory device and writes the result into a memory device using the 'high quality' method.
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

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
hMem	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

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

Parameter	Meaning
hMem	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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	
GUI_MEMDEV_APILIST_1	Create memory device with 1bpp color depth (1 byte per 8 pixels) Use if the specified color conversion requires 1bpp.
GUI_MEMDEV_APILIST_8	Create memory device with 8bpp color depth (1 byte per pixel) Use if the specified color conversion requires 8bpp or less.
GUI_MEMDEV_APILIST_16	Create memory device with 16bpp color depth (1 U16 per pixel) Use if the specified color conversion requires more than 8 bpp. (High color modes)
GUI_MEMDEV_APILIST_32	Create memory device with 32bpp color depth (1 U32 per pixel) Use if the specified color conversion requires more than 16 bpp. (True color modes)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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	
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_565	Same color conversion like the fixed palette mode 565.
GUI_COLOR_CONV_M565	Same color conversion like the fixed palette mode M565.
GUI_COLOR_CONV_8666	Same color conversion like the fixed palette mode 8666.
GUI_COLOR_CONV_888	Same color conversion like the fixed palette mode 888.
GUI_COLOR_CONV_8888	Same color conversion like the fixed palette mode 8888.

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_DrawPerspectiveX()

Description

Draws the given memory device perspectively distorted into the currently selected device.

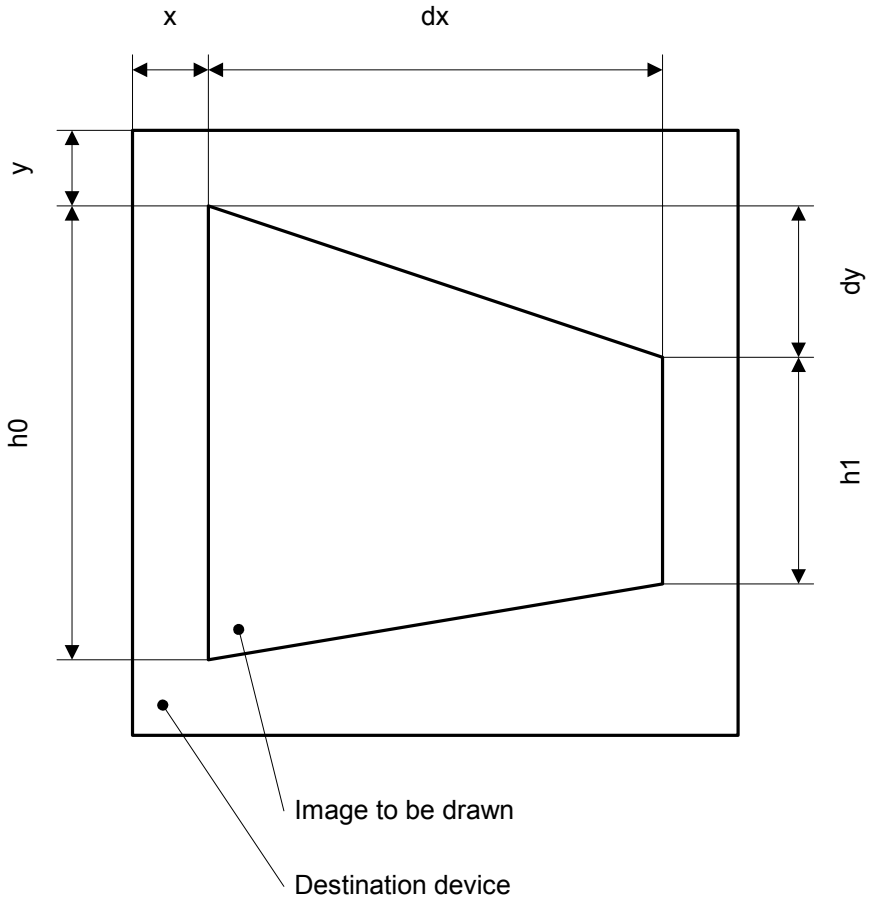
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void GUI_MEMDEV_DrawPerspectiveX(GUI_MEMDEV_Handle hMem, int x, int y,
                                 int h0, int h1, int dx, int dy);
```

Parameter	Meaning
hMem	Handle to source memory device with the image to be drawn.
x	Horizontal start position in pixels.
y	Vertical start position in pixels.
h0	Height of the leftmost edge of the image to be drawn.
h1	Height of the rightmost edge of the image to be drawn.
dx	Width of the image to be drawn.
dy	Position in y from the topmost pixel at the right relative to the topmost pixel at the left.

The picture below explains the parameters more detailed:



Add. information

The function draws the contents of the given memory device into the currently selected device. The origin of the source device should be (0, 0). Size and distortion of the new image is defined by the parameters dx, dy, h0 and h1.
Please note that the function currently does only work with memory devices with 32bpp color depth and a system color depth of 32bpp.

Example

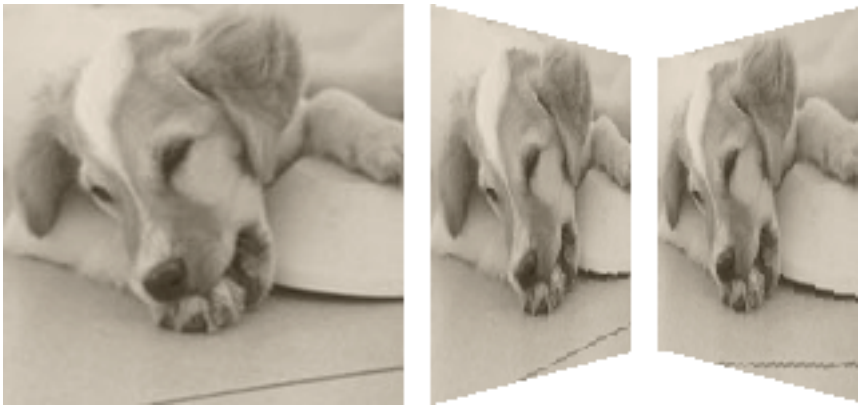
The following sample shows how to use the function:

```
GUI_MEMDEV_Handle hMem0, hMem1, hMem2;
hMem0 = GUI_MEMDEV_CreateFixed(0, 0, 150, 150, GUI_MEMDEV_NOTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
hMem1 = GUI_MEMDEV_CreateFixed(0, 0,  75, 150, GUI_MEMDEV_HASTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
hMem2 = GUI_MEMDEV_CreateFixed(0, 0,  75, 150, GUI_MEMDEV_HASTRANS,
                               GUI_MEMDEV_APILIST_32,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                  GUI_COLOR_CONV_888);
GUI_MEMDEV_Select(hMem0);
GUI_JPEG_Draw(_aJPEG, sizeof(_aJPEG), 0, 0);
GUI_MEMDEV_Select(hMem1);
GUI_MEMDEV_DrawPerspectiveX(hMem0, 0,  0, 150, 110, 75,  20);
GUI_MEMDEV_Select(hMem2);
GUI_MEMDEV_DrawPerspectiveX(hMem0, 0, 20, 110, 150, 75, -20);
GUI_MEMDEV_CopyToLCDAt(hMem0,   0, 10);
GUI_MEMDEV_CopyToLCDAt(hMem1, 160, 10);
GUI_MEMDEV_CopyToLCDAt(hMem2, 245, 10);
```

Screenshot of the above sample



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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_MarkDirty()

Description

Tells the memory device that the pixels in the given area have been changed.

Prototype

```
void GUI_MEMDEV_MarkDirty(GUI_MEMDEV_Handle hMem,
                          int x0, int y0, int x1, int y1);
```

Parameter	Meaning
hMem	Handle to memory device.
x0	Leftmost pixel to be marked.
y0	Topmost pixel to be marked.
x1	Rightmost pixel to be marked.
y1	Bottommost pixel to be marked.

Add. information

When drawing into a memory device with the emWin drawing API functions like GUI_DrawLine() a memory device with transparency support (default) marks each modified pixel. The drawing functions like GUI_MEMDEV_CopyToLCD() then draw only the modified pixels.
If the contents of a memory device have been changed by the application program using the pointer returned by GUI_MEMDEV_GetDataPtr(), the memory device does not know which pixels are changed. In this case the function GUI_MEMDEV_MarkDirty() should be used to mark these pixels.
Using this function with memory devices without transparency, created with the flag GUI_MEMDEV_NOTRANS, makes no sense.

GUI_MEMDEV_ReduceYSize()

Description

Reduces the Y-size of a memory device.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_Rotate(), GUI_MEMDEV_RotateHQ()

Description

The functions rotate and scale the given source memory device. The source device will be rotated and scaled around its center and then shifted by the given amount of pixels. The result is saved into the given destination memory device.
The difference between both functions is the algorithm for calculating the destination pixel data. GUI_MEMDEV_Rotate() uses the 'nearest neighbor' method which is fast but less accurate. GUI_MEMDEV_RotateHQ() uses a more complex method which is quite accurate but not as fast as the 'nearest neighbor' method.
For a more detailed impression of the difference between the functions there are two screenshots at the end of this function description.

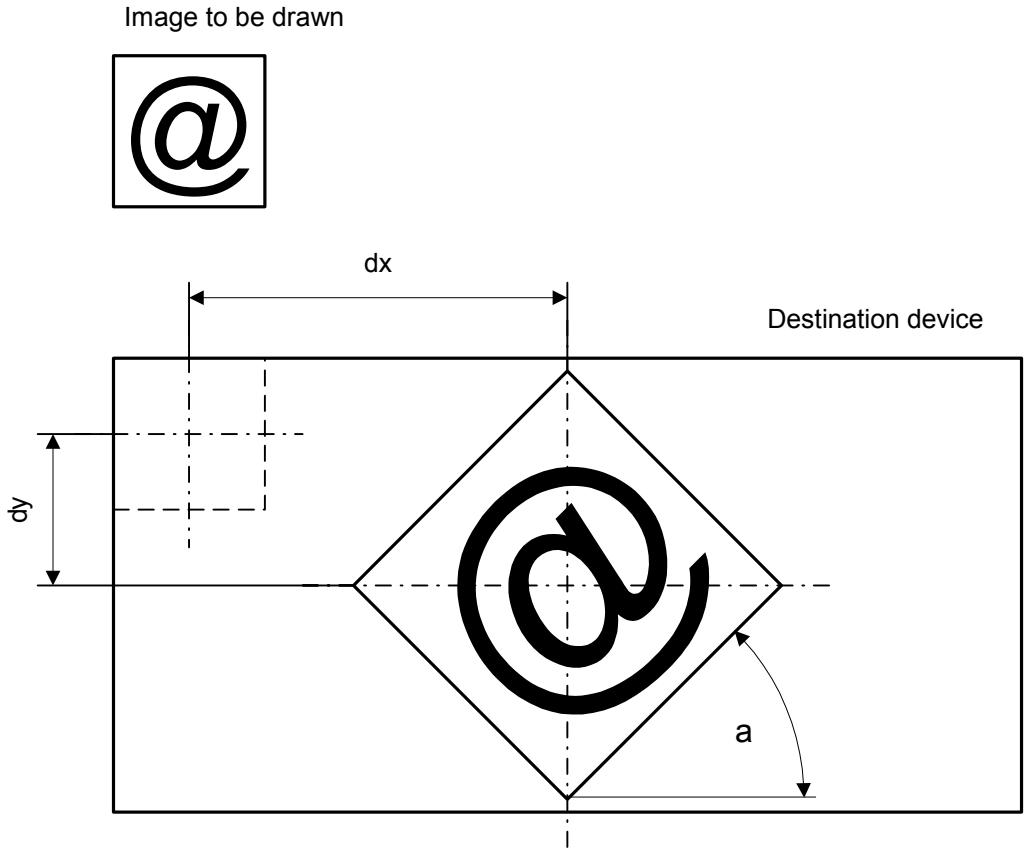
Prototypes

```
void GUI_MEMDEV_RotateHQ(GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                         int dx, int dy, int a, int Mag);

void GUI_MEMDEV_Rotate(GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                       int dx, int dy, int a, int Mag);
```

Parameter	Meaning
hSrc	Handle of memory device to be rotated and scaled.
hDst	Handle of destination device.
dx	Distance in pixels for shifting the image in X.
dy	Distance in pixels for shifting the image in Y.
a	Angle to be used for rotation in degrees * 1000.
Mag	Magnification factor * 1000

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

The following picture gives a more detailed impression of the parameters:



Add. information

Both memory devices, source and destination, need to be 32 bit memory devices. In any other case the function will return immediately.
The sample folder also contains the sample `MEMDEV_ZoomAndRotate.c` which shows in detail how the function can be used.

Sample

```
GUI_MEMDEV_Handle hMemSource;
GUI_MEMDEV_Handle hMemDest;
GUI_RECT RectSource = {0, 0, 69, 39};
GUI_RECT RectDest   = {0, 0, 79, 79};
hMemSource = GUI_MEMDEV_CreateFixed(RectSource.x0, RectSource.y0,
                                    RectSource.x1 - RectSource.x0 + 1,
                                    RectSource.y1 - RectSource.y0 + 1,
                                    GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUI_COLOR_CONV_888);
hMemDest   = GUI_MEMDEV_CreateFixed(RectDest.x0,   RectDest.y0,
                                    RectDest.x1   - RectDest.x0   + 1,
                                    RectDest.y1   - RectDest.y0   + 1,
                                    GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUI_COLOR_CONV_888);
GUI_MEMDEV_Select(hMemSource);
GUI_DrawGradientV(RectSource.x0, RectSource.y0,
                  RectSource.x1, RectSource.y1,
                  GUI_WHITE, GUI_DARKGREEN);
GUI_SetColor(GUI_BLUE);
GUI_SetFont(&GUI_Font20B_ASCII);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringInRect("emWin", &RectSource, GUI_TA_HCENTER | GUI_TA_VCENTER);
GUI_DrawRect(0, 0, RectSource.x1, RectSource.y1);
GUI_MEMDEV_RotateHQ(hMemSource, hMemDest,
                    (RectDest.x1 - RectSource.x1) / 2,
                    (RectDest.y1 - RectSource.y1) / 2,
                    30 * 1000,
                    1000);
GUI_MEMDEV_CopyToLCDAt(hMemSource,  10, (RectDest.y1 - RectSource.y1) / 2);
GUI_MEMDEV_CopyToLCDAt(hMemDest,   100, 0);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screenshot of the above sample using GUI_MEMDEV_RotateHQ()



Screenshot of the above sample using GUI_MEMDEV_Rotate()



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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_WriteExAt(GUI_MEMDEV_Handle hMem,
                          int x, int y, 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:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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))
```

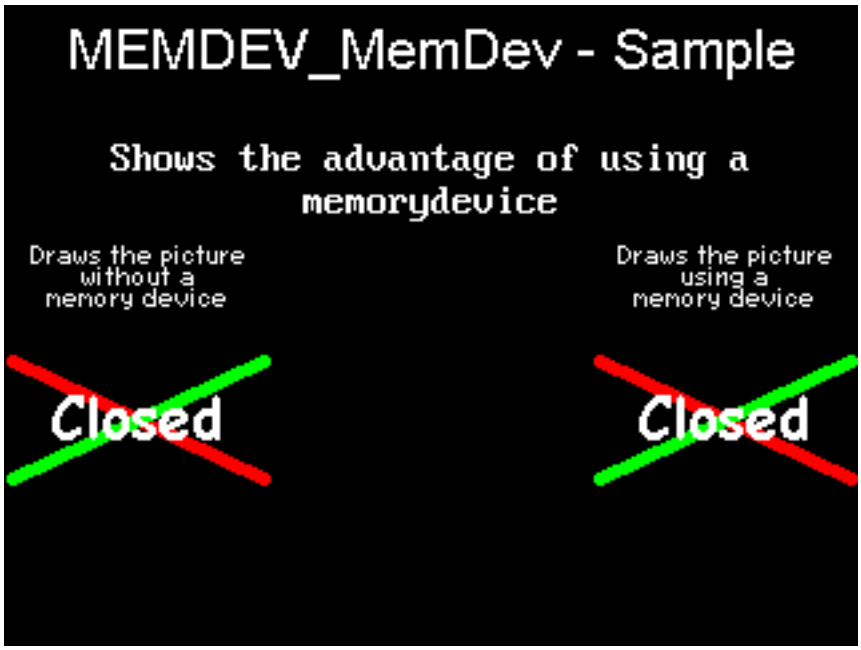
Example for using a memory device

The sample folder contains the following sample which show how memory devices can be used:

- `MEMDEV_MemDev.c`

This example demonstrates the use of a memory device. Some items are written to a memory device and then copied to the display. Please note that several other samples also make use of memory devices and may also be helpful to get familiar with them.

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

(see the GUI_ALLOC_SIZE configuration macro in Chapter 28: "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.

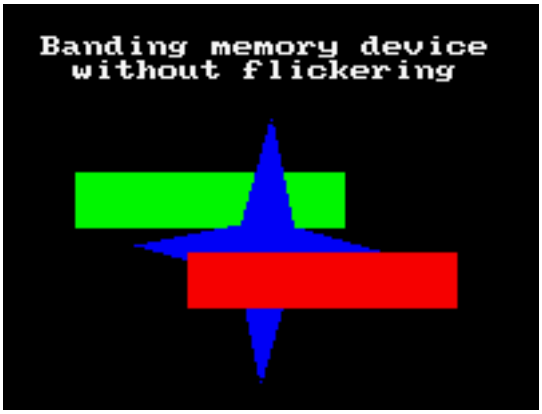
Example for using a banding memory device

The sample folder contains the following sample which shows how the function can be used:

- MEMDEV_Banding.c

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
pAutoDev	Pointer to a GUI_AUTODEV 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
pAutoDev	Pointer to a GUI_AUTODEV object.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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:

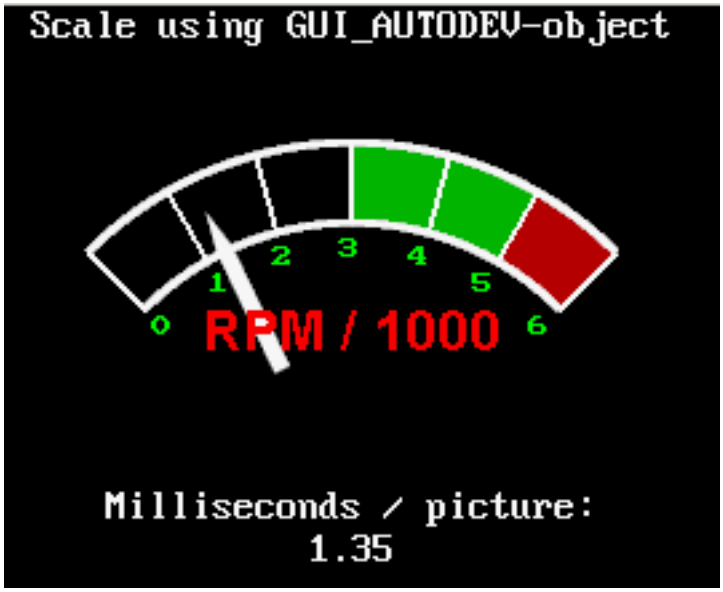
```
typedef struct {
  GUI_AUTODEV_INFO AutoDevInfo; /* Information about what has to be drawn */
  /* Additional data used by the user function */
  ...
} PARAM;

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 */
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example for using an auto device object

The sample MEMDEV_AutoDev.c demonstrates the use of an auto device object. It can be found as MEMDEV_AutoDev.c. A scale with a moving needle is drawn in the background and a small text is written in the foreground. The needle is drawn with the antialiasing feature of emWin. High-resolution antialiasing is used here to improve the appearance of the moving needle. For more information please see Chapter 23: "Antialiasing".

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispDec(Rect.x0, 3);
  GUI_DispString(" Y0:");
  GUI_DispDec(Rect.y0, 3);
  GUI_DispString(" X1:");
  GUI_DispDec(Rect.x1, 3);
  GUI_DispString(" Y1:");
  GUI_DispDec(Rect.y1, 3);
}
```

Screenshot of the above example:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 peridocally 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 peridocally 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 regularily */
  while (1) {
    /* Exec all compontents 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 exectution 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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 regularily */
  while (1) {
    /* Exec all compontents 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 cretain 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 runnig 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.) seperate 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:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
/*********************************************************************
*
*          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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
GUI_X_InitOS()	Initialize the kernel interface module (create a resource semaphore/ mutex).
GUI_X_GetTaskId()	Return a unique, 32-bit identifier for the current task/thread.
GUI_X_Lock()	Lock the GUI (block resource semaphore/mutex).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
GUI_X_SignalEvent()	Signals an event.
GUI_X_Unlock()	Unlock the GUI (unblock resource semaphore/mutex).
GUI_X_WaitEvent()	Waits for an event.

GUI_X_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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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); }
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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);
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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/invalidation

A valid window is a fully updated window which does not need redrawing.
An invalid window does not yet reflect all updates and therefore needs to be redrawn, either completely or partially. When changes are made that affect a particular window, the WM marks that window as invalid. The next time the window is redrawn (either manually or by a callback routine) it will be validated.

Window coordinates

Window coordinates are coordinates of a window. The upper left position (the origin) of the window is (0,0).

Z-position, bottom/top

Although a window is displayed on a two-dimensional screen in terms of X and Y, the WM also manages what is known as a Z-position, or depth coordinate -- a position in a virtual third dimension which determines its placement from background to foreground. Windows can therefore appear on top of or beneath one another.
Setting a window to the bottom will place it "underneath" all of its sibling windows (if any); setting it to the top will place it "on top of" its siblings. When a window is created, it is set to the top by default if no create flag is specified.

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!

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
pMsg	Pointer to a data structure of type WM_MESSAGE.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 occured 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 looses the input focus.
WM_SET_ID	Sent to a window to change the window Id.
WM_SIZE	Sent to a window after its size has changed.
WM_TIMER	Sent to a window after a timer has expired.
Pointer input device (PID) messages	
WM_MOUSEOVER	Sent to a window if a pointer input device touches the outline of a window. Only send if mouse support is enabled.
WM_MOUSEOVER_END	Sent to a window if a pointer input device has been moved out of the outline of a window. Only sent if mouse support is enabled.
WM_PID_STATE_CHANGED	Sent to the window pointed by the pointer input device when the pressed state has been changed.
WM_TOUCH	Sent to a window 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	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 occured in one of its child window. 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 looses 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 looses 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.

WM_TIMER

Description

This message will be send to a window when a timer created by WM_CreateTimer() has expired.

Data

The Data.v value contains the handle of the expired timer.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.
To enable mouse support please add the following line to the file GUIConf.h:

```
#define GUI_SUPPORT_MOUSE 1
```

Data

The Data.p pointer of the message points to a GUI_PID_STATE structure.

Elements of 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_MOUSEOVER_END

Description

A WM_MOUSEOVER_END message is sent to a window if the mouse pointer has been moved out of the window. It is 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. For details about this structure please refer to the message WM_MOUSEOVER.

WM_PID_STATE_CHANGED

Description

Sent to the window affected by the pointer input device when the pressed state has changed. The affected window is the visible window at the input position. With other words: If the user releases for example the touch screen over a window, the pressed state changes from 1 (pressed) to 0 (unpressed). In this case a WM_PID_STATE_CHANGED message is 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 prease 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.

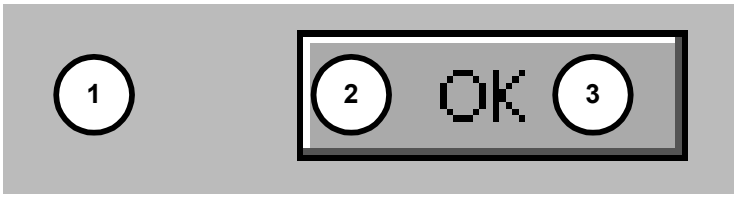
Data

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

The following sample explains what happens if a pointer input device is dragged over a dialog with a button:



Position	Explanation
1	The pointer input device (PID) is pressed at this position. This causes the WM to send the following WM_PID_STATE_CHANGED message to the window at this position: x = Horizontal position in in desktop coordinates. y = Vertical position in in desktop coordinates. State = 1 StatePrev = 0 The WM also sends a WM_TOUCH message with the same x and y coordinates to the window: x = Horizontal position in in desktop coordinates. y = Vertical position in in desktop coordinates. Pressed = 1
2	The PID is dragged to this position. The window below (the button) will receive no WM_PID_STATE_CHANGED message, because the PID remains in pressed state. The WM only sends a WM_TOUCH message to the window: x = Horizontal position in in desktop coordinates. y = Vertical position in in desktop coordinates. Pressed = 1
3	The PID is released at this position. This causes the WM to send the following WM_PID_STATE_CHANGED message to the window at this position: x = Horizontal position in in desktop coordinates. y = Vertical position in in desktop coordinates. State = 0 StatePrev = 1 The WM also sends a WM_TOUCH message with the same x and y coordinates to the window: x = Horizontal position in in desktop coordinates. y = Vertical position in in desktop coordinates. Pressed = 0

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 is changed.
B	WM_SUPPORT_TRANSPARENCY	1	Enable support for transparent windows. If set to 0 the additional code for transparency support is not included.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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	
WM_Activate()	Activates the window manager.
WM_AttachWindow()	Attaches a window to a new parent window.
WM_AttachWindowAt()	Attaches a window to a new parent window at the given position.
WM_BroadcastMessage()	Sends a message to all existing windows.
WM_BringToBottom()	Places a window behind its siblings.
WM_BringToTop()	Places a window in front of its siblings.
WM_ClrHasTrans()	Clears the has transparency flag.
WM_CreateWindow()	Creates a window.
WM_CreateWindowAsChild()	Creates a child window.
WM_Deactivate()	Deactivates the window manager.
WM_DefaultProc()	Default routine to handle messages.
WM_DeleteWindow()	Deletes a window.
WM_DetachWindow()	Detaches a window from its parent window.
WM_DisableWindow()	Sets the widget state to disabled.
WM_EnableWindow()	Sets the window state to enabled (default).
WM_Exec()	Redraws invalid windows by executing callbacks (all jobs).
WM_Exec1()	Redraws one invalid window by executing one callback (one job only).
WM_ForEachDesc()	Iterates over all descendants of a window.
WM_GetActiveWindow()	Returns handle of the active window.
WM_GetCallback()	Returns a pointer to the callback function of a window.
WM_GetClientRect()	Returns the size of the active window.
WM_GetClientRectEx()	Returns the size of a window.
WM_GetDesktopWindow()	Returns the window handle of the desktop window
WM_GetDesktopWindowEx()	Returns the window handle of the specified desktop window
WM_GetDialogItem()	Returns the window handle of a dialog box item (widget).
WM_GetFirstChild()	Returns handle of a window's first child window.
WM_GetFocussedWindow()	Returns the handle of the window with the input focus.
WM_GetHasTrans()	Returns current value of the has transparency flag.
WM_GetInvalidRect()	Returns the invalid rectangle of the given window.
WM_GetNextSibling()	Returns the handle of a window's next sibling.
WM_GetOrgX()	Returns the origin in X of the active window.
WM_GetOrgY()	Returns the origin in Y of the active window.
WM_GetParent()	Returns handle of a window's parent window.
WM_GetPrevSibling()	Returns the handle of a window's previous sibling.
WM_GetStayOnTop()	Returns current value of the stay on top flag.
WM_GetUserData()	Retrieves the user data of a window
WM_GetWindowOrgX()	Returns the origin in X of a window.
WM_GetWindowOrgY()	Returns the origin in Y of a window.
WM_GetWindowRect()	Returns the screen coordinates of the active window.
WM_GetWindowRectEx()	Returns the screen coordinates of a window.
WM_GetWindowSizeX()	Returns the horizontal size (width) of a window.
WM_GetWindowSizeY()	Returns the vertical size (height) of a window.
WM_HasCaptured()	Checks if the given window has captured mouse- and touch-screen-input.
WM_HasFocus()	Checks if the given window has the input focus.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_IsCompletelyCovered()	Checks if a window is completely covered or not.
WM_IsCompletelyVisible()	Checks if a window is completely visible or not.
WM_IsEnabled()	Returns if a window is enabled or not.
WM_IsVisible()	Returns if a window is visible or not.
WM_IsWindow()	Determine whether a specified handle is a valid window handle.
WM_MakeModal()	Changes the window to a 'modal' window.
WM_MoveChildTo()	Sets the position of a window in window coordinates.
WM_MoveTo()	Sets the position of a window in desktop coordinates.
WM_MoveWindow()	Moves a window to another position.
WM_NotifyParent()	Sends a WM_NOTIFY_PARENT message to the parent of the given window.
WM_Paint()	Draws or redraws a window immediately.
WM_PaintWindowAndDescs()	Draws a given window and all descendant windows immediately.
WM_ReleaseCapture()	Stops capturing mouse- and touch screen-input.
WM_ResizeWindow()	Changes window size.
WM_SelectWindow()	Sets the active window to be used for drawing operations.
WM_SendMessage()	Sends a message to a window.
WM_SendMessageNoPara()	Sends a message without parameters to a window.
WM_SendToParent()	Sends the given message to the parent window of the given window.
WM_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.
Timer related	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
WM_CreateTimer()	Creates a timer which sends a WM_TIMER message to a window.
WM_DeleteTimer()	Deletes a timer.
WM_RestartTimer()	Restarts a timer.
Widget related functions	
WM_GetClientWindow()	Returns the handle of the client window.
WM_GetId()	Returns the ID of a widget.
WM_GetInsideRect()	Returns the size of the active window less the border.
WM_GetInsideRectEx()	Returns the size of a window less the border.
WM_GetScrollPosH()	Returns the scroll position of the windows horizontal scroll bar.
WM_GetScrollPosV()	Returns the scroll position of the windows vertical scroll bar.
WM_GetScrollState()	Gets the state of a scroll bar widget.
WM_SetScrollPosH()	Sets the scroll position of the windows horizontal scroll bar.
WM_SetScrollPosV()	Sets the scroll position of the windows vertical scroll bar.
WM_SetScrollState()	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
hWin	Window handle.
hWinParent	Window handle of the new parent.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_BringToTop()

Description

Places a specified window on top of its siblings.

Prototype

```
void WM_BringToTop(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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

Add. information

A window should not delete itself or a parent window in reaction of a broadcasted message.

WM_ClrHasTrans()

Description

Clears the has transparency flag (sets it to 0).

Prototype

```
void WM_ClrHasTrans(WM_HWIN hWin);
```

Parameter	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.
NumExtra-Bytes	Number of extra bytes to be allocated, normally 0.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Style (OR-combinable)	
WM_CF_ANCHOR_BOTTOM	Anchors the bottom edge of the new window relative to the bottom egge 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Style (OR-combinable)	
WM_CF_MEMDEV_ON_REDRAW	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 WM_CF_MEMDEV. It typically accelerates the initial rendering of the window, but maintains the advantage of flicker free updates.
WM_CF_SHOW	Show window after creation.
WM_CF_STAYONTOP	Make sure window stays on top of all siblings created without this flag.

Return value

Handle 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
x0	Upper left X-position in window coordinates of the parent window.
y0	Upper left Y-position in window coordinates of the parent window.
width	X-size of window. If 0, X-size of client area of parent window.
height	Y-size of window. If 0, Y-size of client area of parent window.
hWinParent	Handle of parent window.
Style	Window create flags (see WM_CreateWindow()).
cb	Pointer to callback routine, or NULL if no callback used.
NumExtra-Bytes	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hWin	Window handle.
pcb	Pointer to callback function to be called by WM_ForEachDesc.
pData	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"
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
#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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hWin	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
hWin	Window handle.
pRect	Pointer to a GUI_RECT-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
hWin	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 specfied, rectangular area of a window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_IsCompletelyCovered()

Description

Checks if the given window is completely covered or not.

Prototype

```
char WM_IsCompletelyCovered(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

Return value

1 if the given window is completely covered, otherwise 0.

Add. Information

If the given window handle is invalid or 0 the function returns a wrong result.

WM_IsCompletelyVisible()

Description

Checks if the given window is completely visible or not.

Prototype

```
char WM_IsCompletelyVisible(WM_HWIN hWin);
```

Parameter	Meaning
hWin	Window handle.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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. The macro WM_NOTIFOCATION_USER can be used for defining application defined messages:

```
#define NOTIFICATION_1 (WM_NOTIFOCATION_USER + 0)
#define NOTIFICATION_2 (WM_NOTIFOCATION_USER + 1)
```

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 wnidow 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void WM_SendMessage(WM_HWIN hWin, WM_MESSAGE* pMsg)
```

Parameter	Meaning
hWin	Window handle.
pMsg	Pointer to message.

WM_SendMessageNoPara()

Description

Sends a message without parameters to a specified window.

Prototype

```
void WM_SendMessageNoPara(WM_HWIN hWin, int MsgId)
```

Parameter	Meaning
hWin	Window handle.
MsgId	Id of message to be send.

Add. Information

If only a message Id should be send to a window this should be done with this function, because it does not need a pointer to a WM_MESSAGE structure. Please note that the receiving window gets no further information except the message Id.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 iwhen the user releases the touch.

WM_SetCreateFlags()

Description

Sets the flags to be used as default when creating a new window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 WM_tfPollPID.

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                    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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispDecMin(pThis->v); GUI_DispChar('%');
/* Draw right part of the bar */
    r.x0=r.x1; r.x1=GUI_XMAX;
    WM_SetUserClipRect(&r);
    GUI_SetBkColor(pThis->ColorBar[1]);
    GUI_SetColor(pThis->ColorText[1]);
    GUI_Clear();
    GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 calll `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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 specfied, 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_EnableMemdev (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 Timer related functions

WM_CreateTimer()

Description

Creates a timer which sends a message to the given window after the given time period has expired.

Prototype

```
WM_HTIMER WM_CreateTimer(WM_HWIN hWin, int Id, int Period, int Mode);
```

Parameter	Meaning
hWin	Handle of window to be informed.
Id	(reserved for future use, should be 0)
Period	Time period after which the given window should receive a message.
Mode	(reserved for future use, should be 0)

Return value

Handle of the timer.

Add. information

The function creates a 'one shot timer' which sends a `WM_TIMER` message to the given window. After the timer period has expired the timer object remains valid and can be restarted using the function `WM_RestartTimer()` or deleted with `WM_DeleteTimer()`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

```
static void _cbWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_TIMER:
    /*
      ... do something ...
    */
    WM_RestartTimer(pMsg->Data.v, 1000);
    break;
  }
}

static void _DemoTimer(void) {
  WM_HWIN hWin;
  WM_HTIMER hTimer;
  hWin   = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, _cbWin, 0);
  hTimer = WM_CreateTimer(hWin, 0, 1000, 0);
  while (1) {
    GUI_Exec();
  }
}
```

WM_DeleteTimer()

Description

Deletes the given timer.

Prototype

```
void WM_DeleteTimer(WM_HTIMER hTimer);
```

Parameter	Meaning
hTimer	Handle of the timer to be deleted.

Add. information

After a timer has expired the timer object remains valid and will not be deleted automatically. If it is not used anymore it should be deleted using this function.

WM_RestartTimer()

Description

Restarts the given timer with the given period.

Prototype

```
void WM_RestartTimer(WM_HTIMER hTimer, int Period);
```

Parameter	Meaning
hTimer	Handle of the timer to be restarted.
Period	New period to be used.

Add. information

After the period has expired a WM_TIMER message will be send to the window assigned to the timer. For details please refer to the function WM_CreateTimer().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

14.9 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

14.10 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 */
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
    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();
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 15

Window Objects (Widgets)

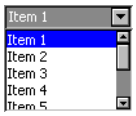
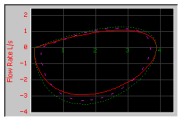
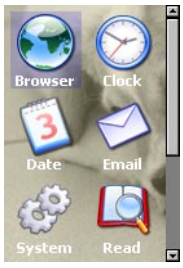
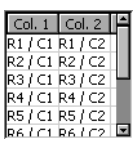
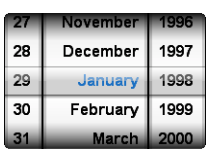
Widgets are windows with object-typeproperties; they are called controls in the windows world and make up the elements of the user interface. They can react automatically to certain events; for example, autton can appear in a different state if it is pressed. Widgets need to be created, have properties 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

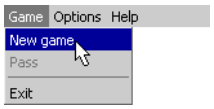
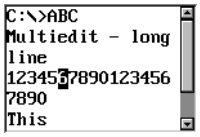
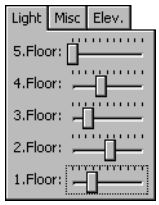
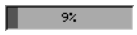
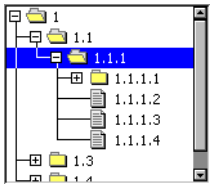
15.1 Some basics

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.
DROPDOWN		Dropdown listbox, opens a listbox when pressed.
EDIT		Single-line edit field which prompts the user to type a number or text.
FRAMEWIN		Frame window. Creates the typical GUI look.
GRAPH		Graph widget, used to show curves or measured values.
HEADER		Header control, used to manage columns.
ICONVIEW		Icon view widget. Useful for a icon based menus like used in many hand held devices.
LISTBOX		Listbox which highlights items as they are selected by the user.
LISTVIEW		Listview widgets are used to creates tables.
LISTWHEEL		Listwheel widget. The data can be moved and accellerated via pointer input device.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Name	Screenshot	Explanation
MENU		Menu widgets are used to create horizontal and vertical menus.
MULTIEDIT		Multiedit widgets are used to edit multiple lines of text.
MULTIPAGE		Multipage widgets are used to create dialogs with multiple pages.
PROGBAR		Progress bar used for visualization.
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.
TREEVIEW		Treeview widget for managing hierarchical lists.

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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_DispStringAt("Progress bar", 100, 20);
hProgBar = PROGBAR_Create(100, 40, 100, 20, WM_CF_SHOW);
```



The first line reserves memory for the handle of the widget. The last line actually creates the widget. The widget will then automatically be drawn by the window manager 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 == DROPDOWN_Callback) {
  /* Widget is a dropdown */
} else if (pCb == LISTBOX_Callback) {
  /* Widget is a listbox */
} else if (...) {
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

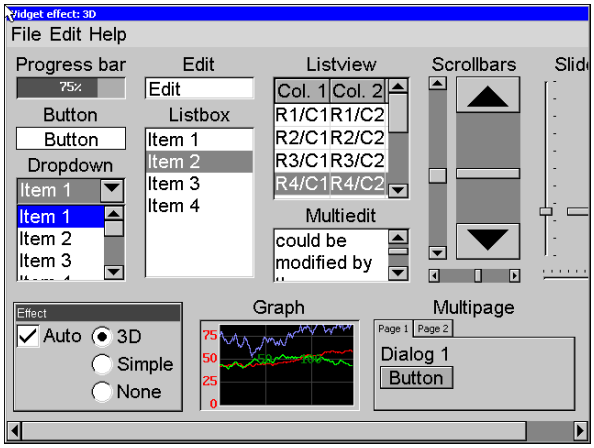
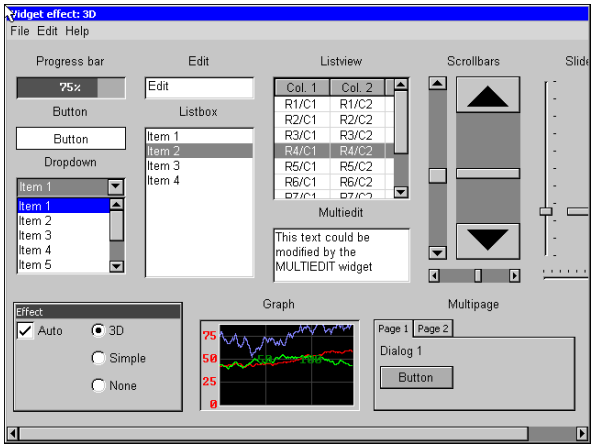
```
  ...
}
```

15.2 Configuration options

Type	Macro	Default	Explanation
B	WIDGET_USE_PARENT_EFFECT	0	If set to 1, each 'child widget' of a widget, has the same effect as the parent widget. If for example a listbox needs to create a scrollbar, the new scrollbar has the same effect as the listbox.
B	WIDGET_USE_SCHEME_LARGE	0	If set to 1, the default appearance of the widgets is large sized. That means that all widgets which show text are configured to use large sized default fonts.
B	WIDGET_USE_SCHEME_MEDIUM	0	If set to 1, the default appearance of the widgets is medium sized. That means that all widgets which show text are configured to use medium sized default fonts.
B	WIDGET_USE_SCHEME_SMALL	1	If set to 1, the default appearance of the widgets is small sized. That means that all widgets which show text are configured to use small sized default fonts.

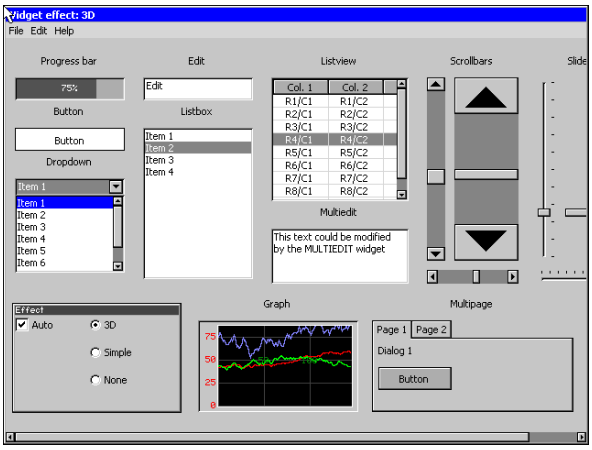
WIDGET_USE_SCHEME...

The table below shows the default appearance of the widget schemes:

WIDGET_USE_SCHEME_LARGE	WIDGET_USE_SCHEME_MEDIUM
	

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

WIDGET_USE_SCHEME_SMALL



15.3 General widget API

15.3.1 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
WM_DeleteWindow()	Delete a window.
WM_DisableMemdev()	Disable usage of memory devices for redrawing.
WM_EnableMemdev()	Enable usage of memory devices for redrawing.
WM_InvalidateWindow()	Invalidate a window.
WM_Paint()	Draw or redraw a window immediately.

For a complete list of WM-related functions, please refer to Chapter 14: "The Window Manager".

15.3.2 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
<WIDGET>_Callback()	Default callback function.
<WIDGET>_CreateIndirect()	Used for automatic creation in dialog boxes.
WIDGET_GetDefaultEffect()	Returns the default effect used for new widgets.
WIDGET_SetDefaultEffect()	Sets the default effect used for new widgets.
WIDGET_SetEffect()	Sets the effect used for a given widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

<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
pMsg	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;
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

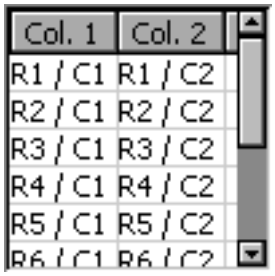
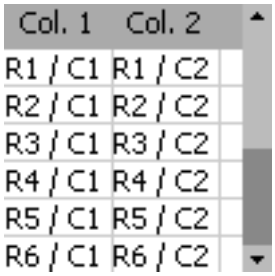
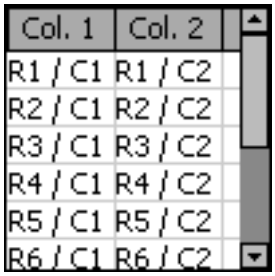
Previous used default effect.

Add. information

The following table shows the appearance of some widgets in dependence of the used effect:

'3D'	'None'	'Simple'
		
		

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

’3D’	’None’	’Simple’
		

WIDGET_SetEffect()

Description

Sets the effect for the given widget.

Prototype

```
void WIDGET_SetEffect(WM_HWIN hObj, const WIDGET_EFFECT* pEffect);
```

Parameter	Meaning
hObj	Handle of widget.
pEffect	Pointer to a WIDGET_EFFECT structure. For details please refer to the function WIDGET_SetDefaultEffect()

15.3.3 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
pDrawItemInfo	Pointer to a WIDGET_ITEM_DRAW_INFO structure.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Elements of WIDGET_ITEM_DRAW_INFO

Data type	Element	Meaning
WM_HWIN	hWin	Handle to the widget.
int	Cmd	(see table below)
int	ItemIndex	Zero based index of item to be drawn.
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 (widgth) in pixels of the given item.
WIDGET_ITEM_GET_YSIZE	The function returns the y-size (height) in pixels of the given item.
WIDGET_ITEM_DRAW	The function draws the given item at the given position.
WIDGET_DRAW_BACKGROUND	The background of the widget should be drawn.
WIDGET_DRAW_OVERLAY	This command is send after all other drawing operations have been finished and enables the possibility to draw some overlaying items above the widget.

Return value

Depends on the given command.

Reaction to commands

The function has to react to the command given in the WIDGET_ITEM_DRAW_INFO structure. This can be done in one of 2 ways:

- By calling the appropriate default function supplied by the particular widget (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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.4 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.

15.4.1 Configuration options

Type	Macro	Default	Explanation
N	BUTTON_3D_MOVE_X	1	Number of pixels that text/bitmap moves in horizontal direction in pressed state.
N	BUTTON_3D_MOVE_Y	1	Number of pixels that text/bitmap moves in vertical direction in pressed state.
N	BUTTON_ALIGN_DEFAULT	GUI_TA_HCENTER | GUI_TA_VCENTER	Alignment used to display the button text.
N	BUTTON_BKCOLOR0_DEFAULT	0xAAAAAA	Background color, unpressed state.
N	BUTTON_BKCOLOR1_DEFAULT	GUI_WHITE	Background color, pressed state.
N	BUTTON_FOCUSCOLOR_DEFAULT	GUI_BLACK	Default color for rendering the focus rectangle.
S	BUTTON_FONT_DEFAULT	&GUI_Font13_1	Font used for button text.
B	BUTTON_REACT_ON_LEVEL	0	See description below.
N	BUTTON_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unpressed state.
N	BUTTON_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, pressed state.

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 avoid 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.4.2 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.

15.4.3 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.

15.4.4 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_SetBMPEx()	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
BUTTON_SetStreamedBitmap()	Set the bitmap used when displaying the BUTTON.
BUTTON_SetStreamedBitmapEx()	Set the bitmap used when displaying the BUTTON.
BUTTON_SetText()	Set the text.
BUTTON_SetTextAlign()	Sets the alignment of the BUTTON text.
BUTTON_SetTextColor()	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
x0	Leftmost pixel of the button (in parent coordinates).
y0	Topmost pixel of the button (in parent coordinates).
xsize	Horizontal size of the button (in pixels).
ysize	Vertical size of the button (in pixels).
Id	ID to be returned when button is pressed.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).

Return value

Handle of 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
x0	X-position of the button relative to the parent window.
y0	Y-position of the buton relative to the parent window.
xsize	Horizontal size of the button (in pixels).
ysize	Vertical size of the button (in pixels).
hParent	Handle of parent window. If 0, the new button window will be a child of the desktop (top-level window).
Id	ID to be returned when button is pressed.
Flags	Window create flags (see BUTTON_Create()).

Return value

Handle of the created BUTTON widget; 0 if the routine fails.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetDefaultTextAlign()

Description

Sets the default text alignment used to display the text of new BUTTON widgets.

Prototype

```
void BUTTON_SetDefaultTextAlign(int Align);
```

Parameter	Meaning
Align	Text alignment to be used. For details refer to GUI_SetTextAlign()

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 BUTTON_SetBitmap()).
pBitmap	Pointer to a bitmap stream.

Add. information

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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

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_SetTextAlign()

Description

Sets the alignement of the button text.

Prototype

```
void BUTTON_SetTextAlign(BUTTON_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of button.
Align	Text alignment to be set (see GUI_SetTextAlign())

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 BUTTON_SetBkColor()).
Color	Text color to be set.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

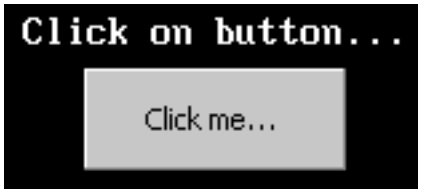
15.4.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_ButtonSimple.c
- WIDGET_ButtonPhone.c
- WIDGET_ButtonRound.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

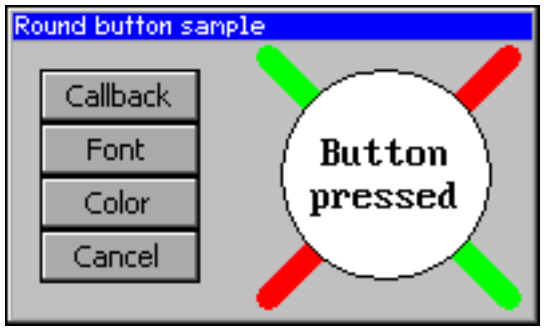
Screenshot of WIDGET_ButtonSimple.c:



Screenshot of WIDGET_ButtonPhone.c:



Screenshot of WIDGET_ButtonRound.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.5 CHECKBOX: Checkbox widget

One of the most familiar widgets for selecting various choices is the check box. A check box may be checked or unchecked by the user, and any number of boxes may be checked at one time. If using a keyboard interface the state of a focused check box can be toggled by the <SPACE> key. A box will appear gray if it is disabled, as seen in the table below where each of the possible check box appearances are illustrated:

	Unchecked	Checked	Third state
Enabled			
Disabled			

All CHECKBOX-related routines are located in the file(s) `CHECKBOX*.c`, `CHECKBOX.h`. All identifiers are prefixed CHECKBOX.

15.5.1 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.

15.5.2 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 of the box without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Status of check box has been changed.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.5.3 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.

15.5.4 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_GetDefaultTextAlign()	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_SetDefaultTextAlign()	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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

CHECKBOX_Check()

Before	After
	

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 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 of 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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 of 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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_GetDefaultTextAlign()

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
hObj	Handle of widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of widget.
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 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
hObj	Handle of check box.

Return value

0: not checked
1: checked

CHECKBOX_SetBkColor()

Before	After
	

Description

Sets the background color used to display the background of the check box.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
Color	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
pFont	Pointer to GUI_FONT 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
pBitmap	Pointer to bitmap.
Index	(see table below)

Permitted values for parameter Index	
CHECKBOX_BI_INACTIV_UNCHECKED	Sets the bitmap displayed when the check box is unchecked and disabled.
CHECKBOX_BI_ACTIV_UNCHECKED	Sets the bitmap displayed when the check box is unchecked and enabled.
CHECKBOX_BI_INACTIV_CHECKED	Sets the bitmap displayed when the check box is checked and disabled.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetDefaultTextAlign(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_SetTextAlign().

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	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
hObj	Handle of check box.
pBitmap	Pointer to bitmap.
Index	(see table shown under CHECKBOX_SetDefaultImage)

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
	

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
	 

Description

Sets the new state of the given check box widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
	

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
	

Description

Sets the alignment used to display the check box text beside the box.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetDefaultTextAlign() and the configuraton macro CHECKBOX_TEXTALIGN_DEFAULT can be used to set a user defined default value.

CHECKBOX_SetTextColor()

Before	After
	

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
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

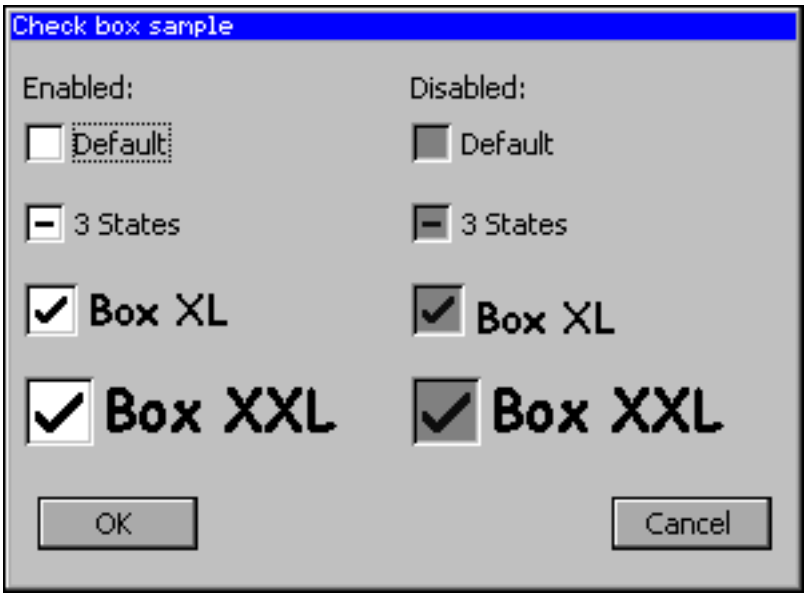
15.5.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- `WIDGET_Checkbox.c`

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

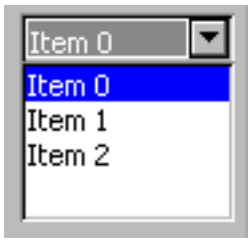
Screenshot of WIDGET_Checkbox.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.6 DROPDOWN: Dropdown widget

DROPDOWN widgets are used to select one element of a list with several columns. It shows the currently selected item in non open state. If the user opens a DROPDOWN widget a LISTBOX appears to select a new item.

DROPDOWN closed	DROPDOWN opened
	

If mouse support is enabled, the open list reacts on moving the mouse over it.

15.6.1 Configuration options

Type	Macro	Default	Explanation
N	DROPDOWN_ALIGN_DEFAULT	GUI_TA_LEFT	Text alignment used to display the dropdown text in closed state.
S	DROPDOWN_FONT_DEFAULT	&GUI_Font13_1	Default font
N	DROPDOWN_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	DROPDOWN_BKCOLOR1_DEFAULT	GUI_GRAY	Background color, selected state without focus.
N	DROPDOWN_BKCOLOR2_DEFAULT	GUI_BLUE	Background color, selected state with focus.
N	DROPDOWN_KEY_EXPAND	GUI_KEY_SPACE	Key which can be used to expand the dropdown list.
N	DROPDOWN_KEY_SELECT	GUI_KEY_ENTER	Key which can be used to select an item from the open dropdown list.
N	DROPDOWN_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	DROPDOWN_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, selected state without focus.
N	DROPDOWN_TEXTCOLOR2_DEFAULT	GUI_WHITE	Enable 3D support.

15.6.2 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.
WM_NOTIFICATION_SEL_CHANGED	The selection of the dropdown list has been changes.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.6.3 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.

15.6.4 DROPDOWN API

The table below lists the available emWin DROPDOWN-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
DROPDOWN_AddString()	Adds an element to the DROPDOWN list.
DROPDOWN_Collapse()	Closes the dropdown list.
DROPDOWN_Create()	Creates a DROPDOWN widget. (Obsolete)
DROPDOWN_CreateEx()	Creates a DROPDOWN widget.
DROPDOWN_CreateIndirect()	Creates a DROPDOWN widget from a resource table entry.
DROPDOWN_DecSel()	Decrements selection.
DROPDOWN_DecSelExp()	Decrements selection in expanded state.
DROPROWN_DeleteItem()	Deletes an item of the DROPDOWN list.
DROPDOWN_Expand()	Opens the dropdown list.
DROPDOWN_GetDefaultFont()	Returns the default font used to create DROPDOWN widgets.
DROPDOWN_GetItemDisabled()	Returns the state of the given item.
DROPDOWN_GetListbox()	Returns the handle of the attached LISTBOX in expanded state.
DROPDOWN_GetNumItems()	Returns the number of items in the dropdown list.
DROPDOWN_GetSel()	Returns the number of the currently selected element.
DROPDOWN_GetSelExp()	Returns the number of the currently selected element in expanded state.
DROPDOWN_IncSel()	Increments selection.
DROPDOWN_IncSelExp()	Increments selection in expanded state.
DROPDOWN_InsertString()	Inserts a string to the dropdown list.
DROPDOWN_SetAutoScroll()	Enables the automatic use of a scrollbar in the dropdown list.
DROPDOWN_SetBkColor()	Sets the background color.
DROPDOWN_SetColor()	Sets the color of the arrow and the button of the widget.
DROPDOWN_SetDefaultColor()	Sets the default color of the arrow and the button for new DROPDOWN widgets.
DROPDOWN_SetDefaultFont()	Sets the default font used to create DROPDOWN widgets.
DROPDOWN_SetDefaultScrollbarColor()	Sets the default colors of the optional scrollbar in the dropdown list.
DROPDOWN_SetFont()	Sets the font of the given DROPDOWN widget
DROPDOWN_SetItemDisabled()	Sets the state of the given item.
DROPDOWN_SetItemSpacing()	Sets an additional spacing between the items of the dropdown list.
DROPDOWN_SetScrollbarColor()	Sets the colors of the scrollbar in the dropdown list.
DROPDOWN_SetSel()	Sets the current selection.
DROPDOWN_SetSelExp()	Sets the current selection in expanded state.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
DROPDOWN_SetTextAlign()	Sets the text alignment used to display the dropdown text in closed state.
DROPDOWN_SetTextColor()	Sets the text color of the given DROPDOWN widget.
DROPDOWN_SetTextHeight()	Sets the height of the rectangle used to display the dropdown text in closed state.

DROPDOWN_AddString()

Description

Adds a new element to the dropdown list.

Prototype

```
void DROPDOWN_AddString(DROPDOWN_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 DROPDOWN widget.

Prototype

```
void DROPDOWN_Collapse(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

DROPDOWN_Create()

(Obsolete, DROPDOWN_CreateEx() should be used instead)

Description

Creates a DROPDOWN widget of a specified size at a specified location.

Prototype

```
DROPDOWN_Handle DROPDOWN_Create(WM_HWIN hWinParent,
                                int x0, int y0,
                                int xsize, int ysize,
                                int Flags);
```

Parameter	Meaning
hWinParent	Handle of parent window
x0	Leftmost pixel of the DROPDOWN widget (in parent coordinates).
y0	Topmost pixel of the DROPDOWN widget (in parent coordinates).
xsize	Horizontal size of the DROPDOWN widget (in pixels).
ysize	Vertical size of the DROPDOWN widget in open state (in pixels).
Flags	Window create flags. Typically WM_CF_SHOW 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 of the created DROPDOWN widget, 0 on failure.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 DROPDOWN widget.

DROPDOWN_CreateEx()

Description

Creates a DROPDOWN widget of a specified size at a specified location.

Prototype

```
DROPDOWN_Handle DROPDOWN_CreateEx(int x0, int y0, int xsize, int ysize,
                                  WM_HWIN hParent, int WinFlags,
                                  int ExFlags, int Id);
```

Parameter	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).
hParent	Handle of parent window. If 0, the new DROPDOWN 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	
0	No function.
DROPDOWN_CF_AUTOSCROLLBAR	Enable automatic use of a scrollbar. For details please refer to DROPDOWN_SetAutoScroll
DROPDOWN_CF_UP	Creates a DROPDOWN widget which opens the dropdown list above the widget. This flag is usefull if the space below the widget is not sufficient for the dropdown list.

Return value

Handle of the created widget; 0 if the routine fails.

DROPDOWN_CreateIndirect()

Prototype explained at the beginning of the chapter.

DROPDOWN_DecSel()

Description

Decrement the selection, moves the selection of a specified DROPDOWN widget up by one item.

Prototype

```
void DROPDOWN_DecSel(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

DROPDOWN_DecSelExp()

Description

Decrements the selection of the attached LISTBOX in expanded state.

Prototype

```
void DROPDOWN_DecSelExp(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

DROPROWN_DeleteItem()

Description

Deletes the given item of the dropdown list.

Prototype

```
void DROPDOWN_DeleteItem(DROPDOWN_Handle hObj, unsigned int Index);
```

Parameter	Meaning
hObj	Handle of widget.
Index	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.

DROPDOWN_Expand()

Description

Opens the dropdown list of the widget.

Prototype

```
void DROPDOWN_Expand(DROPDOWN_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.

DROPDOWN_GetItemDisabled()

Description

Returns the state of the given item.

Prototype

```
unsigned DROPDOWN_GetItemDisabled(DROPDOWN_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of widget
Index	Zerobased index of the item.

Return value

1 if the given item is disabled, 0 if not.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

DROPDOWN_GetListbox()

Description

Returns the handle of the attached LISTBOX widget in expanded state.

Prototype

```
LISTBOX_Handle DROPDOWN_GetListbox(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

Handle of the attached LISTBOX widget in expanded state, 0 if DROPDOWN is in collapsed state.

DROPDOWN_GetNumItems()

Description

Returns the number of items in the given DROPDOWN widget.

Prototype

```
int DROPDOWN_GetNumItems(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

Number of items in the given DROPDOWN widget.

DROPDOWN_GetSel()

Description

Returns the number of the currently selected item in a specified DROPDOWN widget.

Prototype

```
int DROPDOWN_GetSel(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

Number of the currently selected item.

DROPDOWN_GetSelExp()

Description

Returns the number of the currently selected item of the attached LISTBOX in expanded state.

Prototype

```
int DROPDOWN_GetSelExp(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Number of the currently selected item.

DROPDOWN_IncSel()

Description

Increment the selection, moves the selection of a specified DROPDOWN widget down by one item.

Prototype

```
void DROPDOWN_IncSel(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

DROPDOWN_IncSelExp()

Description

Increments the selection of the attached LISTBOX in expanded state.

Prototype

```
void DROPDOWN_IncSelExp(DROPDOWN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

DROPDOWN_InsertString()

Description

Inserts a string to the dropdown list at the given position.

Prototype

```
void DROPDOWN_InsertString(DROPDOWN_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.

DROPDOWN_SetAutoScroll()

Description

Enables the automatic use of a vertical scrollbar in the dropdown list.

Prototype

```
void DROPDOWN_SetAutoScroll(DROPDOWN_Handle hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of widget.
OnOff	(see table below)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

DROPDOWN_SetBkColor()

Before	After
	

Description

Sets the background color of the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetBkColor(DROPDOWN_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_SELFOCUS	Selected element, with focus.

DROPDOWN_SetColor()

Before	After
	

Description

Sets the color of the button or the arrow of the given DROPDOWN widget.

Prototype

```
void DROPDOWN_SetColor(DROPDOWN_Handle hObj,
                       unsigned int Index,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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 DROPDOWN widgets.

Prototype

```
GUI_COLOR DROPDOWN_SetDefaultColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Please refer to the function DROPDOWN_SetColor().
Color	Color to be used.

DROPDOWN_SetDefaultFont()

Description

Sets the default font used for new DROPDOWN widgets.

Prototype

```
void DROPDOWN_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 DROPDOWN_SetDefaultScrollbarColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Please refer to the function DROPDOWN_SetScrollbarColor().
Color	Color to be used.

DROPDOWN_SetFont()

Description

Sets the font used to display the given DROPDOWN widget.

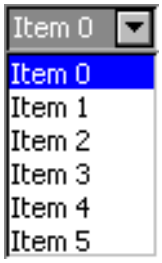
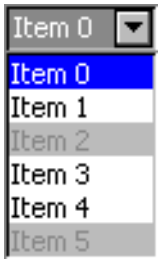
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void DROPDOWN_SetFont(DROPDOWN_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
hObj	Handle of widget
pFont	Pointer to the font.

DROPDOWN_SetItemDisabled()

Before	After
	

Description

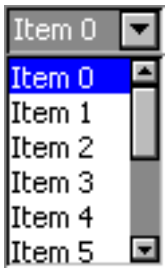
Sets the enabled state of the given item.

Prototype

```
void DROPDOWN_SetItemDisabled(DROPDOWN_Handle hObj,
                              unsigned Index, int OnOff);
```

Parameter	Meaning
hObj	Handle of widget
Index	Zerobased index of the item.
OnOff	1 for enabled, 0 for disabled.

DROPDOWN_SetScrollbarColor()

Before	After
	

Description

Sets the colors of the optional scrollbar in the dropdown list.

Prototype

```
void DROPDOWN_SetScrollbarColor(DROPDOWN_Handle hObj,
                                unsigned int Index,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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 DROPDOWN widget.

Prototype

```
void DROPDOWN_SetScrollbarWidth(DROPDOWN_Handle hObj, unsigned Width);
```

Parameter	Meaning
hObj	Handle of widget.
Width	Width of the scrollbar(s) used by the dropdown list of the given DROPDOWN widget.

DROPDOWN_SetSel()

Description

Sets the selected item of a specified DROPDOWN widget.

Prototype

```
void DROPDOWN_SetSel(DROPDOWN_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of widget
Sel	Element to be selected.

DROPDOWN_SetSelExp()

Description

Sets the selected item of the attached LISTBOX in expanded state.

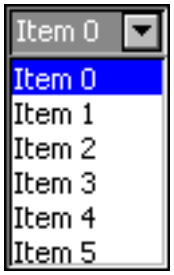
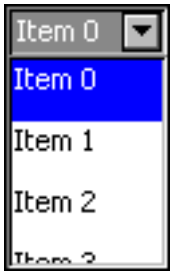
Prototype

```
void DROPDOWN_SetSelExp(DROPDOWN_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of widget
Sel	Element to be selected.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

DROPDOWN_SetItemSpacing()

Before	After
	

Description

Sets an additional spacing below the items of the dropdown list.

Prototype

```
void DROPDOWN_SetItemSpacing(DROPDOWN_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 DROPDOWN_SetTextAlign(DROPDOWN_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 DROPDOWN widget.

Prototype

```
void DROPDOWN_SetTextColor(DROPDOWN_Handle hObj,
                           unsigned int Index,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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_SELFOCUS	Selected element, with focus.

DROPDOWN_SetTextHeight()

Before	After
	

Description

Sets the height of the rectangle used to display the DROPDOWN text in closed state.

Prototype

```
void DROPDOWN_SetTextHeight(DROPDOWN_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 DROPDOWN 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.

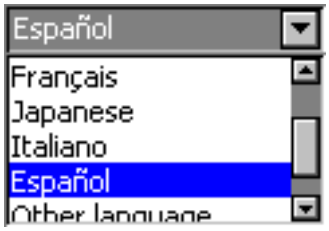
15.6.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- `WIDGET_Dropdown.c`

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Dropdown.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.7 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.

15.7.1 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.

15.7.2 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_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	Value (content) of the edit widget has changed.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.7.3 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().

15.7.4 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_EnableBlink()	Enables/disables a blinking cursor
EDIT_GetCursorCharPos()	Returns the number of the character at the cursor position.
EDIT_GetCursorPixelPos()	Returns the pixel position of the cursor.
EDIT_GetDefaultBkColor()	Returns the default background color.
EDIT_GetDefaultFont()	Returns the default font.
EDIT_GetDefaultTextAlign()	Returns the default texe 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
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.
EDIT_SetText()	Sets the text.
EDIT_SetTextAlign()	Sets the text alignment for the edit field.
EDIT_SetTextColor()	Sets the color(s) for the text.
EDIT_SetTextMode()	Sets the edit mode of the widget back to text mode.
EDIT_SetValue()	Sets the current value.
EDIT_SetUlongMode()	Enables the unsigned long decimal edit mode.
GUI_EditBin()	Edits a binary value at the current cursor position.
GUI_EditDec()	Edits a decimal value at the current cursor position.
GUI_EditHex()	Edits a hexadecimal value at the current cursor position.
GUI_EditString()	Edits a string at the current cursor position.

EDIT_AddKey()

Description

Adds user input to a specified edit field.

Prototype

```
void EDIT_AddKey(EDIT_Handle hObj, int Key);
```

Parameter	Meaning
hObj	Handle of edit field.
Key	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
x0	Leftmost pixel of the edit field (in parent coordinates).
y0	Topmost pixel of the edit field (in parent coordinates).
xsize	Horizontal size of the edit field (in pixels).
ysize	Vertical size of the edit field (in pixels.
Id	ID to be returned.
MaxLen	Maximum count of characters.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Handle of 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 of 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 of the created widget; 0 if the routine fails.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_EnableBlink()

Description

Enables/disables a blinking cursor.

Prototype

```
void EDIT_EnableBlink(EDIT_Handle hObj, int Period, int OnOff);
```

Parameter	Meaning
hObj	Handle of edit field.
Period	Blinking period
OnOff	1 enables blinking, 0 disables blinking

Add. Information

This function calls GUI_X_GetTime().

EDIT_GetCursorCharPos()

Description

Returns the number of the character at the cursor position.

Prototype

```
int EDIT_GetCursorCharPos(EDIT_Handle hObj);
```

Parameter	Meaning
hObj	Handle of edit field.

Return value

Number of the character at the cursor position.

Add. Information

The widget returns the character position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

EDIT_GetCursorPixelPos()

Description

Returns the pixel position of the cursor in window coordinates.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void EDIT_GetCursorPixelPos(EDIT_Handle hObj, int * pxPos, int * pyPos);
```

Parameter	Meaning
hObj	Handle of edit field.
pxPos	Pointer to integer variable for the X-position in window coordinates.
pyPos	Pointer to integer variable for the Y-position in window coordinates.

Add. Information

The widget returns the pixel position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

EDIT_GetDefaultBkColor()

Description

Returns the default background color used for EDIT widgets.

Prototype

```
GUI_COLOR EDIT_GetDefaultBkColor(unsigned int Index);
```

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 fields 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetDefaultTextAlign(int Align);
```

Parameter	Meaning
Align	Default text alignment. For details please refer to GUI_SetTextAlign().

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.
GUI_EDIT_SUPPRESS_LEADING_ZEROES	Does not show leading zeroes.

EDIT_SetFloatValue

Description

The function can be used to set the floating point value of the edit 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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	See table below.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

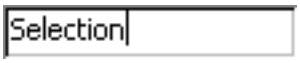
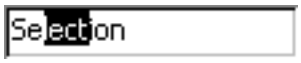
```
void EDIT_SetpfAddKeyEx(EDIT_Handle hObj, tEDIT_AddKeyEx * pfAddKeyEx);
```

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

Before	After
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 GUI_SetTextAlign())

EDIT_SetTextColor()

Description

Sets the edit fields text color.

Prototype

```
void EDIT_SetTextColor(EDIT_Handle hObj, unsigned int Index,
                       GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of edit field.
Index	Index for text color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
EDIT_CI_DISABLED	Sets the text color for disabled state.
EDIT_CI_ENABLED	Sets the text color for enabled state.

EDIT_SetTextMode()

Description

Sets the edit mode of the 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_SetUlongMode()

Description

Enables the unsigned long decimal edit mode of the edit field. The given value can be modified in the given range.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void EDIT_SetUlongMode(EDIT_Handle hEdit, 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_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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                    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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. Information

The routine returns after pressing <ENTER> or <ESC>. The contents of the given text will be modified only if <ENTER> is pressed.

15.7.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- `WIDGET_Edit.c`
- `WIDGET_EditWinmode.c`

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screen shot of WIDGET_Edit.c:



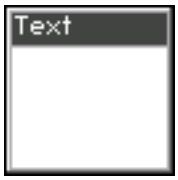
Screen shot of WIDGET_EditWinmode.c:



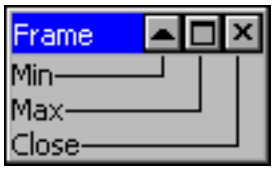
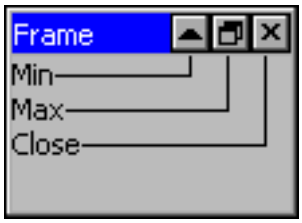
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.8 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
	

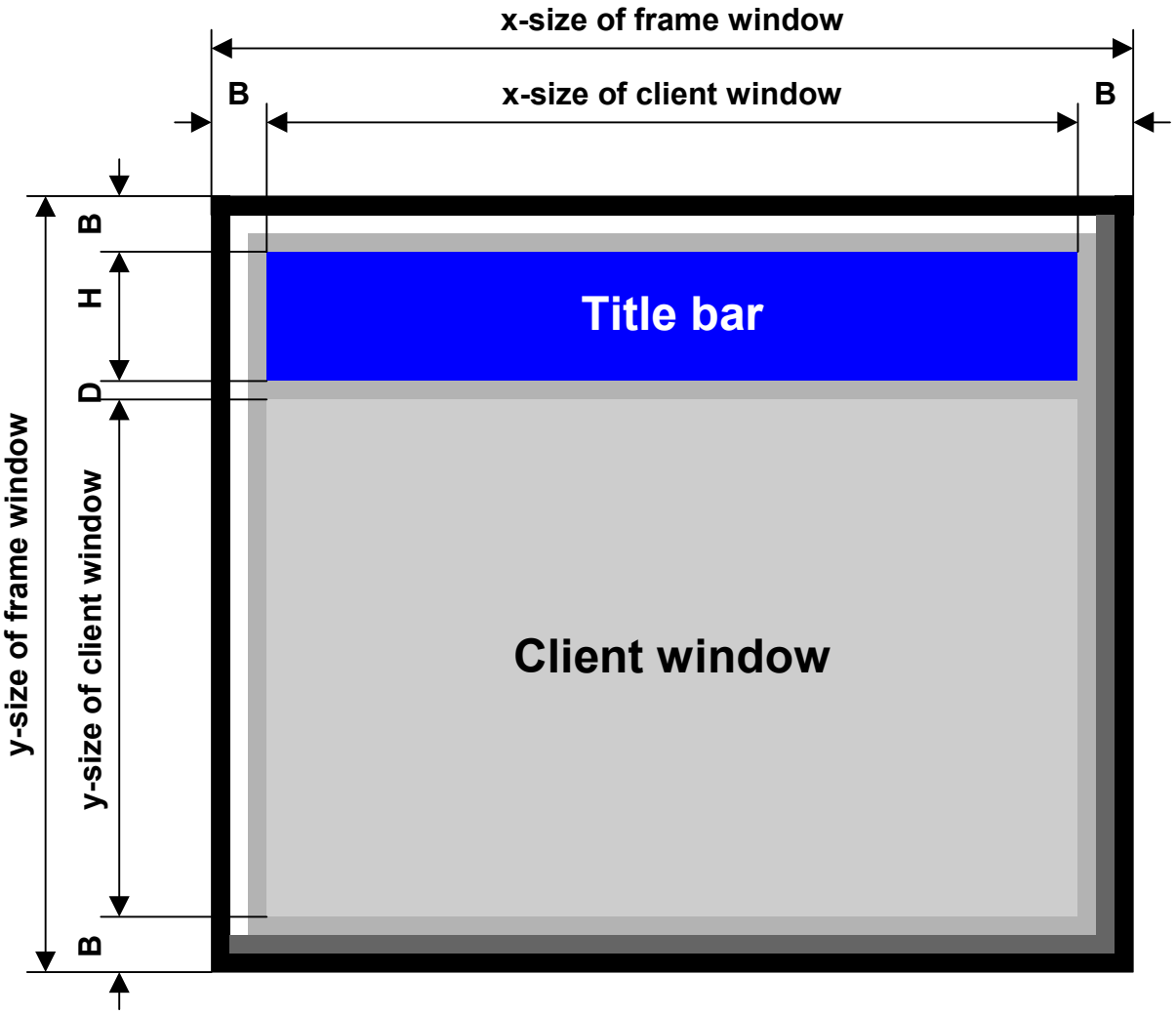
You can attach predefined buttons to the title bar as seen below or you can attach your own buttons to a title bar:

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	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.8.1 Structure of the frame window

The following diagram shows the detailed structure and looks of a frame window:



The frame window actually consists of 2 windows; the main window and a child window. The child window is called `Client window`. It is important to be aware of this when dealing with callback functions: There are 2 windows with 2 different callback functions. When creating child windows, these child windows are typically created as children of the client window; their parent is therefor the client window.

Detail	Description
`B`	Border size of the frame window. The default size of the border is 3 pixels.
`H`	Height of the title bar. Depends on the size of the used font for the title.
`D`	Spacing between title bar and client window. (1 pixel)
`Title bar`	The title bar is part of the frame window and not a separate window.
`Client window`	The client window is a separate window created as a child window of the frame window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.8.2 Configuration options

Type	Macro	Default	Explanation
B	FRAMEWIN_ALLOW_DRAG_ON_FRAME	1	Allows dragging the widget on the surrounding frame.
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	0xaaaaaa	Frame color.
N	FRAMEWIN_IBORDER_DEFAULT	1	Inner border width, in pixels.
N	FRAMEWIN_TITLEHEIGHT_DEFAULT	0	Default height of title bar.

15.8.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.8.4 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_GetActive()	Returns if the frame window is in avtive state.
FRAMEWIN_GetBarColor()	Returns the color of the title bar.
FRAMEWIN_GetBorderSize()	Returns the size of the border.
FRAMEWIN_GetDefaultBarColor()	Returns the default color of the title bar.
FRAMEWIN_GetDefaultBorderSize()	Returns the default border size
FRAMEWIN_GetDefaultClientColor()	Returns the default color of the client area.
FRAMEWIN_GetDefaultFont()	Returns the default font used for the title bar
FRAMEWIN_GetDefaultTextColor()	Returns the default text color of the title.
FRAMEWIN_GetDefaultTitleHeight()	Returns the default size of the title bar
FRAMEWIN_GetFont()	Returns the font used for the title text.
FRAMEWIN_GetText()	Returns the title text.
FRAMEWIN_GetTextAlign()	Returns the alignment of the title text.
FRAMEWIN_GetTitleHeight()	Returns the height of the title bar.
FRAMEWIN_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_OwnerDraw()	Default function for drawing the title bar.
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
FRAMEWIN_SetClientColor()	Sets the color of the client area
FRAMEWIN_SetDefaultBarColor()	Sets the default color of the title bar
FRAMEWIN_SetDefaultBorderSize()	Sets the default border size
FRAMEWIN_SetDefaultClientColor()	Sets the default color of the client area.
FRAMEWIN_SetDefaultFont()	Sets the default font used for the title bar.
FRAMEWIN_SetDefaultTextColor()	Sets the default text color of the title.
FRAMEWIN_SetDefaultTitleHeight()	Sets the default height of the title bar
FRAMEWIN_SetFont()	Selects the font used for the title text.
FRAMEWIN_SetMoveable()	Sets the frame window to a moveable/non-moveable state.
FRAMEWIN_SetOwnerDraw()	Enables the frame window to be owner drawn.
FRAMEWIN_SetResizeable()	Sets the frame window to resizable state.
FRAMEWIN_SetText()	Sets the title text.
FRAMEWIN_SetTextAlign()	Sets the alignment of the title text.
FRAMEWIN_SetTextColor()	Sets the color(s) for the title text.
FRAMEWIN_SetTextColorEx()	Sets the color(s) for the title text.
FRAMEWIN_SetTitleHeight()	Sets the height of the title bar.
FRAMEWIN_SetTitleVis()	Sets the visibility flag of the title bar

FRAMEWIN_AddButton()

Before	After
	

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
hObj	Handle of frame window.
Flags	(see table below)
Off	X-offset used to create the BUTTON widget
Id	ID of the BUTTON widget

Permitted values for parameter Flags	
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

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()

Before	After	Maximized
		

Description

Adds a maximize button to the title bar of the frame window.

Prototype

```
WM_HWIN FRAMEWIN_AddMaxButton(FRAMEWIN_Handle hObj,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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.

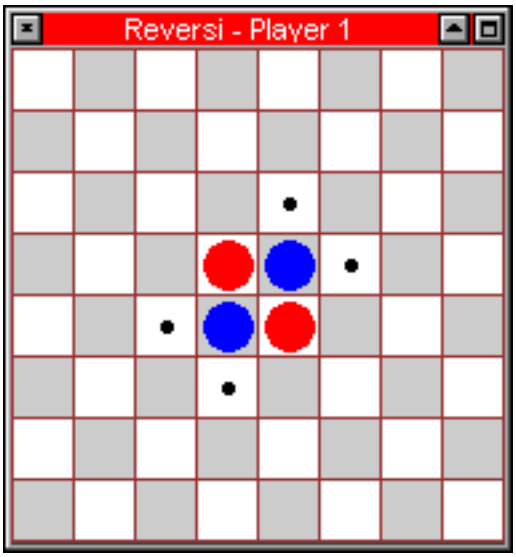
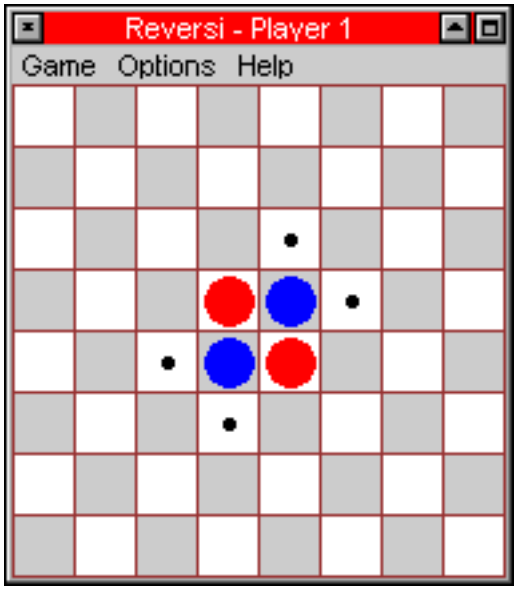
Return value

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()

Before	After
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After	Minimized window
		

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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 of 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 of 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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Parameter	Meaning
ysize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new FRAMEWIN widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (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.
pTitle	Title displayed in the title bar.
cb	Pointer to user callback routine. This user callback routine is called by the window callback routine of the client window.

Permitted values for parameter ExFlags	
0	No function.
FRAMEWIN_CF_MOVEABLE	Sets the new frame window to a moveable state. For details please refer to FRAMEWIN_SetMoveable

Return value

Handle of 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_GetActive()

Description

Returns if the given frame window is in active state or not.

Prototype

```
GUI_COLOR FRAMEWIN_GetBarColor(FRAMEWIN_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of frame window.

Return value

1 if frame window is in active state, 0 if not.

FRAMEWIN_GetBarColor()

Description

Returns the color of the title bar of the given frame window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
GUI_COLOR FRAMEWIN_GetBarColor(FRAMEWIN_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of frame window.
Index	(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

Color of the title bar as RGB value.

FRAMEWIN_GetBorderSize()

Description

Returns the border size of the given frame window.

Prototype

```
int FRAMEWIN_GetBorderSize(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	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	(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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_GetFont()

Description

Returns a pointer to the font used to draw the title text.

Prototype

```
const GUI_FONT GUI_UNI_PTR * FRAMEWIN_GetFont(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Return value

Pointer to the font used to draw the title text.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

FRAMEWIN_GetText()

Description

Returns the title text.

Prototype

```
void FRAMEWIN_GetText(FRAMEWIN_Handle hObj, char * pBuffer, int MaxLen);
```

Parameter	Meaning
hObj	Handle of frame window.
pBuffer	Pointer to buffer to be filled with the title text.
MaxLen	Buffer size in bytes.

Add. information

If the buffer size is smaller than the title text the function copies MaxNum bytes.

FRAMEWIN_GetTextAlign()

Description

Returns the text alignment of the title text.

Prototype

```
int FRAMEWIN_GetTextAlign(FRAMEWIN_Handle hObj);
```

Parameter	Meaning
hObj	Handle of frame window.

Return value

The currently used text alignment. For details about text aligment please refer to the function GUI_SetTextAlign().

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

Description

Enlarges a frame window to the size of its parent window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

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_OwnerDraw()

Description

Default function for drawing the title bar of a frame window.

Prototypes

```
int FRAMEWIN_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 FRAMEWIN_SetOwnerDrawn() is used. It should be called for all unhandled commands passed to the owner draw function. For more information, refer to the section explaining user drawn widgets and FRAMEWIN_SetOwnerDraw().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 automaticly. It is not recomended 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

FRAMEWIN_SetBarColor()

Before	After
	

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
hObj	Handle of frame window.
Size	New border size of the frame window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of frame window.
Color	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
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_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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

FRAMEWIN_SetFont()

Before	After
	

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.
Moveable state means, the frame window can be dragged with a pointer input device (PID). To move the frame window, it first needs to be touched with a PID in pressed state in the title area. Moving the pressed PID now moves also the widget.
If the config macro FRAMEWIN_ALLOW_DRAG_ON_FRAME is 1 (default), the frame window can also be dragged on the surrounding frame. This works only if the frame window is not in resizable state.

FRAMEWIN_SetOwnerDraw()

Description

Enables the frame window to be owner drawn.

Prototype

```
void FRAMEWIN_SetOwnerDraw(FRAMEWIN_Handle hObj,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

Parameter	Meaning
hObj	Handle of frame window.
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 frame window has to be drawn. It gives you the possibility to draw a complete customized title bar, not just plain text. pfDrawItem is a pointer to an application-defined function of type WIDGET_DRAW_ITEM_FUNC which is explained at the beginning of the chapter.

Example

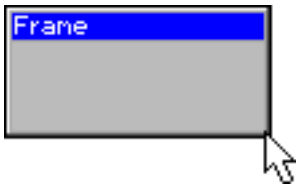
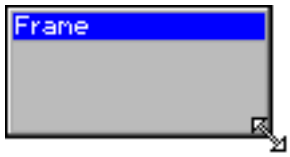
The following shows a typical owner draw function:

```
int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  GUI_RECT Rect;
  char acBuffer[20];
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW:
    Rect.x0 = pDrawItemInfo->x0 + 1;
    Rect.x1 = pDrawItemInfo->x1 - 1;
    Rect.y0 = pDrawItemInfo->y0 + 1;
    Rect.y1 = pDrawItemInfo->y1;
    FRAMEWIN_GetText(pDrawItemInfo->hWin, acBuffer, sizeof(acBuffer));
    GUI_DrawGradientH(pDrawItemInfo->x0, pDrawItemInfo->y0,
                      pDrawItemInfo->x1, pDrawItemInfo->y1,
                      GUI_RED, GUI_GREEN);
    GUI_SetFont(FRAMEWIN_GetFont(pDrawItemInfo->hWin));
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_SetColor(GUI_YELLOW);
    GUI_DispStringInRect(acBuffer, &Rect,
                         FRAMEWIN_GetTextAlign(pDrawItemInfo->hWin));
    return 0;
  }
  return FRAMEWIN_OwnerDraw(pDrawItemInfo);
}

void CreateFrameWindow(void) {
  ...
  FRAMEWIN_SetOwnerDraw(hWin, _OwnerDraw);
  ...
}
```

Screenshot of above example



FRAMEWIN_SetResizeable()

Before	After
	

Description

Sets the resizable state of the given frame window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 in resizable state its size can be changed by dragging the borders. If a pointer input device points over the border, the cursor will change to a resize cursor (if cursor is on and if optional mouse support is enabled).
If pointing to the edge of the border, the X and Y size of the window can be changed simultaneously.

FRAMEWIN_SetText()

Before	After
	

Description

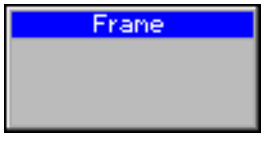
Sets the title text.

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()

Before	After
	

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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

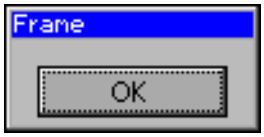
```
int FRAMEWIN_SetTitleHeight(FRAMEWIN_Handle hObj, int Height);
```

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()

Before	After
	

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

15.8.5 Examples

The sample folder contains the following samples which show how the widget can be used:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

- `WIDGET_FrameWin.c`

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screen shot of WIDGET_FrameWin.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.9 GRAPH: Graph widget

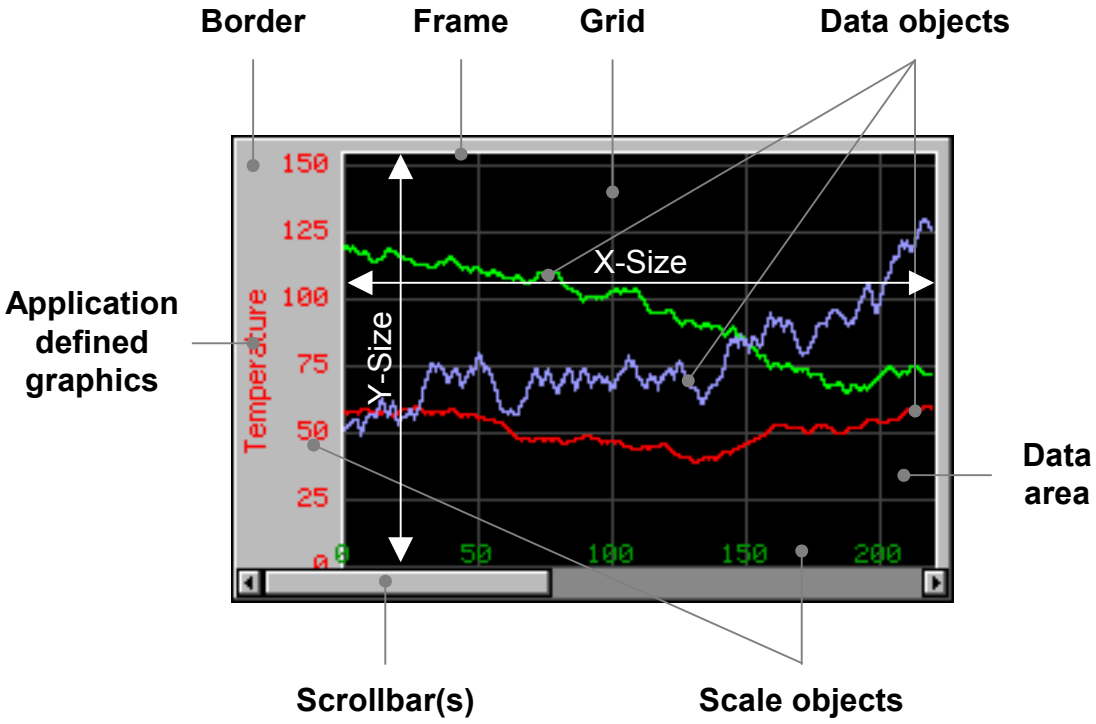
Graph widgets can be used to visualize data. Typical applications for graph widgets are showing measurement values or the curve of a function graph. Multiple curves can be shown simultaneously. Horizontal and vertical scales can be used to label the curves. A grid with different horizontal and vertical 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.

15.9.1 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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.9.2 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 somting with the widget...
*/
WM_DeleteWindow(hGraph);
```

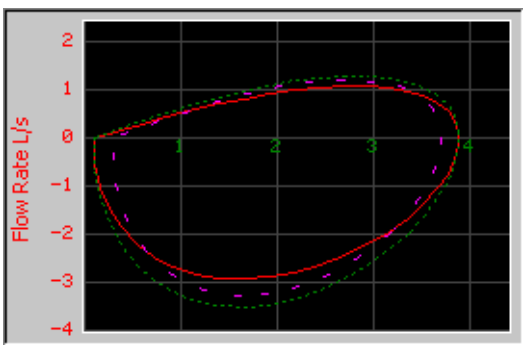
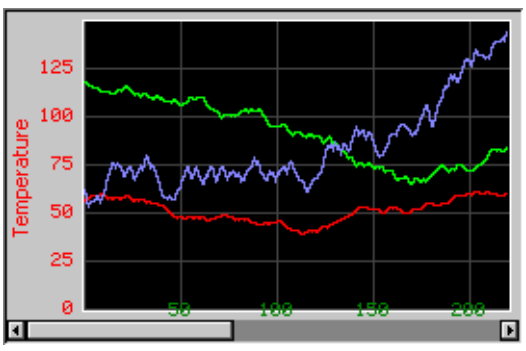
15.9.3 Drawing process

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.

15.9.4 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
	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.9.4.1 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.

15.9.4.2 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 with continuously updated measurement values.

15.9.5 Configuration options

15.9.5.1 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.

15.9.5.2 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.

15.9.6 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.9.7 GRAPH API

The table below lists the available emWin GRAPH-related routines in alphabetical order. Detailed descriptions of the routines follow.

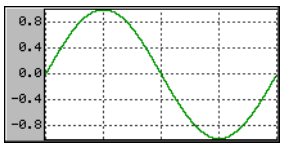
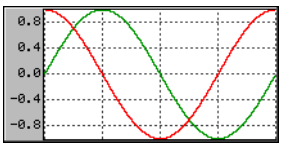
Routine	Explanation
Common routines	
GRAPH_AttachData()	Attaches a data object to a graph widget.
GRAPH_AttachScale()	Attaches a scale object to a graph widget.
GRAPH_CreateEx()	Creates a graph widget.
GRAPH_DetachData()	Detaches a data object from a graph widget.
GRAPH_DetachScale()	Detaches a scale object from 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
GRAPH_SetGridFixedX()	Fixes the grid in X-axis.
GRAPH_SetGridVis()	Enables the drawing of a grid.
GRAPH_SetLineStyleH()	Sets the line style for the horizontal grid lines.
GRAPH_SetLineStyleV()	Sets the line style for the vertical grid lines.
GRAPH_SetVSizeX()	Sets the horizontal range of the graph widget.
GRAPH_SetVSizeY()	Sets the vertical range of the graph widget.
GRAPH_SetUserDraw()	Sets the user callback function.
GRAPH_DATA_YT related routines	
GRAPH_DATA_YT_AddValue()	Adds one data item to the data object.
GRAPH_DATA_YT_Create()	Creates a GRAPH_DATA_YT object.
GRAPH_DATA_YT_Delete()	Deletes the given GRAPH_DATA_YT object.
GRAPH_DATA_YT_MirrorX()	Mirrors the x-axis.
GRAPH_DATA_YT_SetAlign()	Sets the alignment of the given GRAPH_DATA_YT object.
GRAPH_DATA_YT_SetOffY()	Sets a vertical offset for drawing the data.
GRAPH_DATA_XY related routines	
GRAPH_DATA_XY_AddPoint()	Adds one point to the data object.
GRAPH_DATA_XY_Create()	Creates a GRAPH_DATA_YT object.
GRAPH_DATA_XY_Delete()	Deletes the given GRAPH_DATA_XY object.
GRAPH_DATA_XY_SetLineStyle()	Sets the line style used to draw the data.
GRAPH_DATA_XY_SetOffX()	Sets a horizontal offset for drawing the data.
GRAPH_DATA_XY_SetOffY()	Sets a vertical offset for drawing the data.
GRAPH_DATA_XY_SetPenSize()	Sets the pen size used to draw the data.
Scale related routines	
GRAPH_SCALE_Create()	Creates a scale object.
GRAPH_SCALE_Delete()	Deletes the given scale object.
GRAPH_SCALE_SetFactor()	Sets a calculation factor used to calculate from pixels to the desired unit.
GRAPH_SCALE_SetFixed()	Avoids scrolling of the scale.
GRAPH_SCALE_SetFont()	Sets the font used to draw the numbers.
GRAPH_SCALE_SetNumDecs()	Sets the number of digits of the fractional portion.
GRAPH_SCALE_SetOff()	Sets an optional offset which is added to the numbers.
GRAPH_SCALE_SetPos()	Sets the horizontal or vertical position of the scale.
GRAPH_SCALE_SetTextColor()	Sets the text color of the scale.
GRAPH_SCALE_SetTickDist()	Sets the distance in pixels between the tick marks.

15.9.7.1 Common routines

GRAPH_AttachData()

Before	After
	

Description

Attaches a data object to an existing graph widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

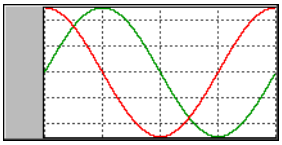
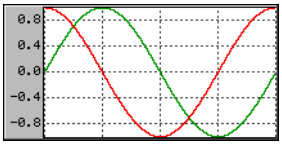
```
void GRAPH_AddGraph(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

Parameter	Meaning
hObj	Handle of widget
hData	Handle of the data object to be added to the widget. The data object should be created with GRAPH_DATA_YT_Create() or GRAPH_DATA_XY_Create()

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()

Before	After
	

Description

Attaches a scale object to an existing graph widget.

Prototype

```
void GRAPH_AttachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

Parameter	Meaning
hObj	Handle of widget
hScale	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
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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Parameter	Meaning
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	(see table below)
Id	Window Id of the widget.

Permitted values for parameter x	
GRAPH_CF_GRID_FIXED_X	This flag 'fixes' the grid in X-axis. That means if horizontal scrolling is used, the grid remains on its position.

Return value

Handle of the created graph widget; 0 if the routine fails.

GRAPH_DetachData()

Description

Detaches a data object from a graph widget.

Prototype

```
void GRAPH_DetachData(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

Parameter	Meaning
hObj	Handle of widget
hData	Handle of the data object to be detached from the widget.

Add. Information

Once detached from a graph widget the application needs to destroy the data object. Detaching a data object does not delete it. For more details about deleting data objects please refer to GRAPH_DATA_YT_Delete() and GRAPH_DATA_XY_Delete().

GRAPH_DetachScale()

Description

Detaches a scale object from a graph widget.

Prototype

```
void GRAPH_DetachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

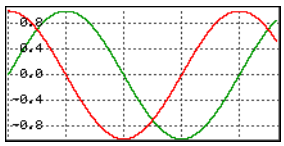
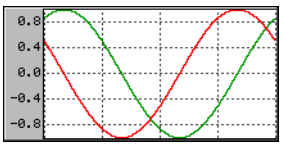
Parameter	Meaning
hObj	Handle of widget
hScale	Handle of the scale object to be detached from the widget.

Add. Information

Once detached from a graph widget the application needs to destroy the scale object. Detaching a scale object does not delete it. For more details about deleting scale objects please refer to GRAPH_SCALE_Delete().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GRAPH_SetBorder()

Before	After
	

Description

Sets the left, top, right and bottom border of the given graph widget.

Prototype

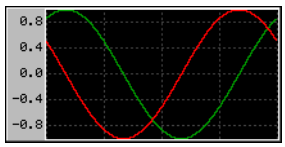
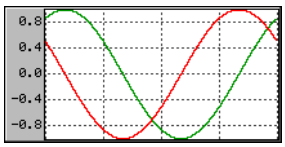
```
void GRAPH_SetBorder(GRAPH_Handle hObj,
                     unsigned BorderL, unsigned BorderT,
                     unsigned BorderR, unsigned BorderB);
```

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)

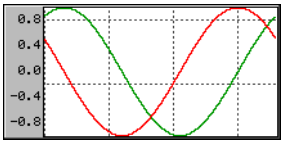
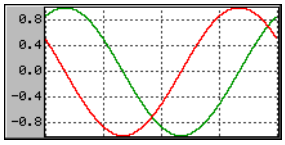
© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Index	
GRAPH_CI_BK	Sets the background color.
GRAPH_CI_BORDER	Sets the color of the border area.
GRAPH_CI_FRAME	Sets the color of the thin frame line.
GRAPH_CI_GRID	Sets the color of the grid.

Return value

Previous color used for the desired item.

GRAPH_SetGridDistX(), GRAPH_SetGridDistY()

Before	After
	

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
hObj	Handle of widget.
OnOff	1 if grid should be fixed in X-axis, 0 if not (default).

Return value

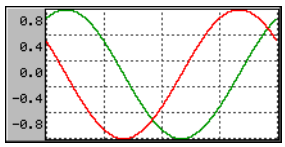
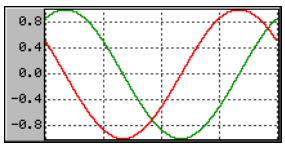
Previous value used

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

Description

Adds an offset used to show the horizontal grid lines.

Prototype

```
unsigned GRAPH_SetGridOffY(GRAPH_Handle hObj, unsigned Value);
```

Parameter	Meaning
`hObj`	Handle of widget.
`Value`	Offset to be used.

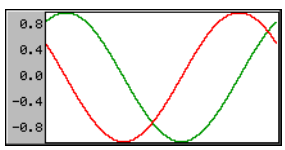
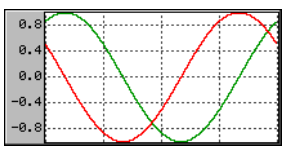
Return value

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

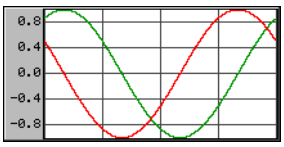
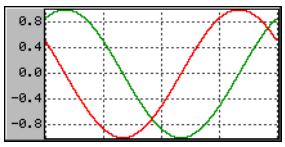
```
unsigned GRAPH_SetGridVis(GRAPH_Handle hObj, unsigned OnOff);
```

Parameter	Meaning
hObj	Handle of widget.
OnOff	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
hObj	Handle of widget.
LineStyle	Line style to be used. For details about the supported line styles please refer to the function GUI_SetLineStyle(). Default is GUI_LS_SOLID.

Return value

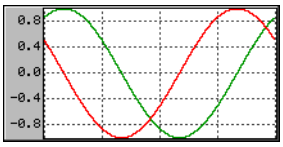
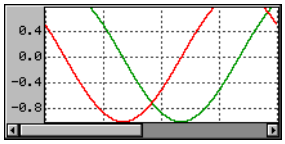
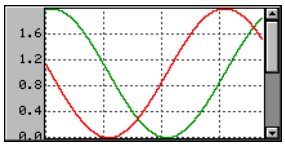
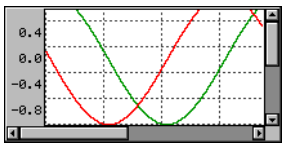
Previous line style used to draw the horizontal/vertical grid lines.

Add. Information

Please note that using other styles than GUI_LS_SOLID will need more time to show the grid.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

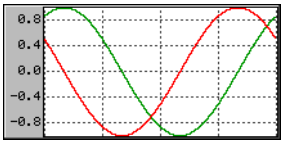
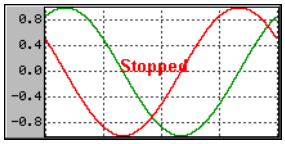
Previous virtual size of the widgets data area in X or Y-axis.

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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GRAPH_SetUserDraw()

Before	After
	

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
hObj	Handle of widget
pUserDraw	Pointer to application function to be called by the widget during the drawing process.

Permitted values for parameter Stage	
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.

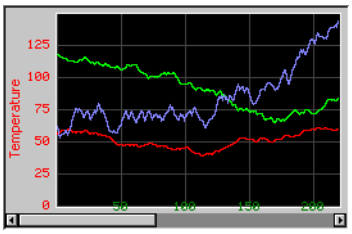
Example

The following small sample shows the use of a user draw function:

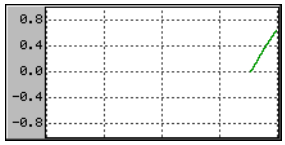
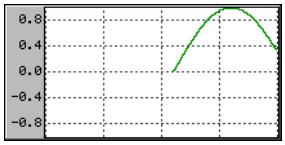
```
static void _UserDraw(WM_HWIN hWin, int Stage) {
  switch (Stage) {
  case GRAPH_DRAW_FIRST:
    /* Draw for example a user defined grid... */
    break;
  case GRAPH_DRAW_LAST:
    /* Draw for example a user defined scale or additional text... */
    break;
  }
}

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 */
  ...
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.9.7.2 GRAPH_DATA_YT related routines



GRAPH_DATA_YT_AddValue()

Before	After
	

Description

Adds a new data item to a GRAPH_DATA_YT object.

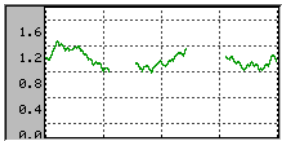
Prototype

```
void GRAPH_DATA_YT_AddValue(GRAPH_DATA_Handle hDataObj, I16 Value);
```

Parameter	Meaning
hDataObj	Handle of data object.
Value	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.
The value 0x7FFF can be used to handle invalid data values. These values are excluded when drawing the graph. The following screenshot shows a graph with 2 gaps of invalid data:



GRAPH_DATA_YT_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,
```


© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    unsigned NumItems);
```

Parameter	Meaning
Color	Color to be used to draw the data.
MaxNumItems	Maximum number of data items.
pData	Pointer to data to be added to the object. The pointer should point to an array of I16 values.
NumItems	Number of data items to be added.

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_Delete()

Description

Deletes the given data object.

Prototype

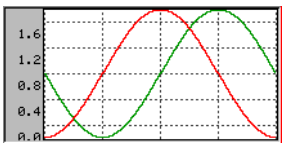
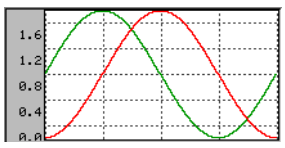
```
void GRAPH_DATA_YT_Delete(GRAPH_DATA_Handle hDataObj);
```

Parameter	Meaning
hDataObj	Data object to be deleted.

Add. Information

When a graph widget is deleted it deletes all currently attached data objects. So the application needs only to delete unattached data objects.

GRAPH_DATA_YT_MirrorX()

Before	After
	

Description

Mirrors the x-axis of the widget.

Prototype

```
void GRAPH_DATA_YT_MirrorX(GRAPH_DATA_Handle hDataObj, int Value);
```

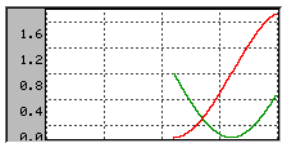
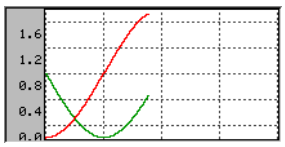
Parameter	Meaning
hDataObj	Handle of data object.
OnOff	1 for mirroring the x-axis, 0 for default view.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. Information

Per default the data is drawn from the right to the left. After calling this function the data is drawn from the left to the right.

GRAPH_DATA_YT_SetAlign()

Before	After
	

Description

Sets the alignment of the data.

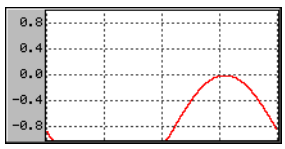
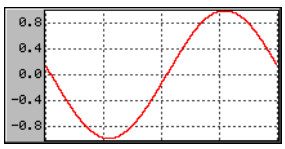
Prototype

```
void GRAPH_DATA_YT_SetAlign(GRAPH_DATA_Handle hDataObj, int Align);
```

Parameter	Meaning
hDataObj	Handle of data object.
Align	(see table below)

Permitted values for parameter Align	
GRAPH_ALIGN_RIGHT	The data is aligned at the right edge (default).
GRAPH_ALIGN_LEFT	The data is aligned at the left edge.

GRAPH_DATA_YT_SetOffY()

Before	After
	

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
hDataObj	Handle of data object.
Off	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.

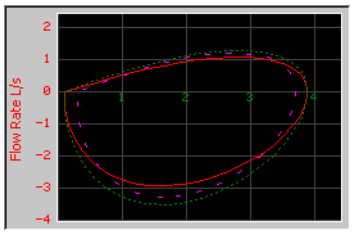
© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Example

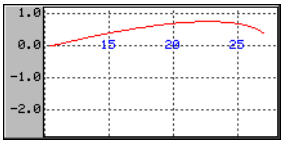
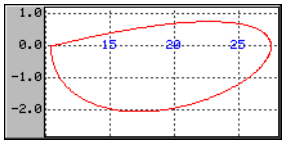
If for example the visible data range should be -200 to -100 the data needs to be shiftet in positive direction by 200 pixels:

```
GRAPH_DATA_YT_SetOffY(hDataObj, 200);
```

15.9.7.3 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
hDataObj	Handle of data object.
pPoint	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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    unsigned NumItems);
```

Parameter	Meaning
Color	Color to be used to draw the data.
MaxNumItems	Maximum number of points.
pData	Pointer to data to be added to the object. The pointer should point to an array of GUI_POINT's.
NumItems	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_Delete()

Description

Deletes the given data object.

Prototype

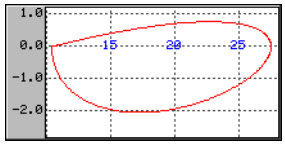
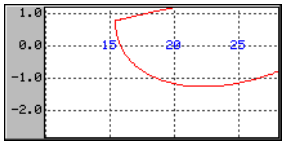
```
void GRAPH_DATA_XY_Delete(GRAPH_DATA_Handle hDataObj);
```

Parameter	Meaning
hDataObj	Data object to be deleted.

Add. Information

When a graph widget is deleted it deletes all currently attached data objects. So the application needs only to delete unattached data objects.

GRAPH_DATA_XY_SetOffX(), GRAPH_DATA_XY_SetOffY()

Before	After
	

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
hDataObj	Handle of data object.
Off	Horizontal/vertical offset which should be used to draw the polyline.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. Information

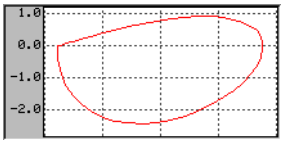
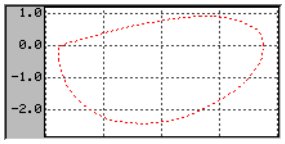
The range of data shown by the data object is (0, 0) - (X-size of data area - 1, Y-size of data area - 1). In case of using scroll bars the current scroll position is added to the respective range. To make other ranges of data visible this functions should be used to set an offset, so that the data is 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_SetLineStyle()

Before	After
	

Description

Sets the line style used to draw the polyline.

Prototype

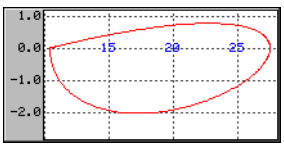
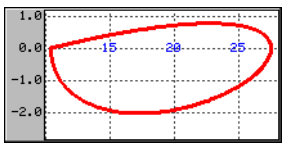
```
void GRAPH_DATA_XY_SetLineStyle(GRAPH_DATA_Handle hDataObj, U8 LineStyle);
```

Parameter	Meaning
hDataObj	Handle of data object.
LineStyle	New line style to be used. For details about the supported line styles please refer to the function GUI_SetLineStyle().

Limitations

Please note, that only curves with line style GUI_LS_SOLID (default) can be drawn with a pen size >1.

GRAPH_DATA_XY_SetPenSize()

Before	After
	

Description

Sets the pen size used to draw the polyline.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void GRAPH_DATA_XY_SetPenSize(GRAPH_DATA_Handle hDataObj, U8 PenSize);
```

Parameter	Meaning
hDataObj	Handle of data object.
PenSize	Pen size which should be used to draw the polyline.

Limitations

Please note, that only curves with line style GUI_LS_SOLID (default) can be drawn with a pen size >1.

15.9.7.4 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GRAPH_SCALE_Delete()

Description

Deletes the given scale object.

Prototype

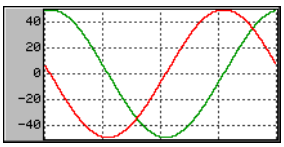
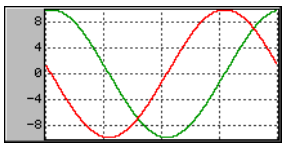
```
void GRAPH_SCALE_Delete(GRAPH_SCALE_Handle hScaleObj);
```

Parameter	Meaning
hScaleObj	Scale object to be deleted.

Add. Information

When a graph widget is deleted it deletes all currently attached scale objects. So the application needs only to delete unattached scale objects.

GRAPH_SCALE_SetFactor()

Before	After
	

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
hScaleObj	Handle of scale object.
Factor	Factor to be used to calculate the number.

Return value

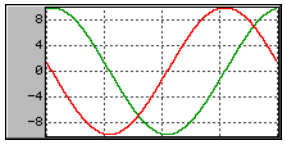
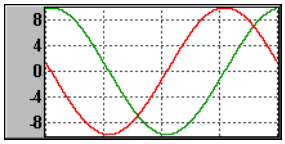
Old factor used to calculate the numbers.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GRAPH_SCALE_SetFont()

Before	After
	

Description

Sets the font used to draw the scale numbers.

Prototype

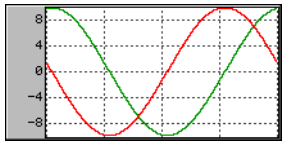
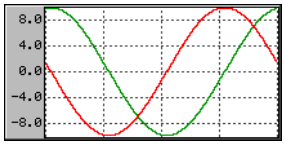
```
const GUI_FONT * GRAPH_SCALE_SetFont(GRAPH_SCALE_Handle hScaleObj,
                                     const GUI_FONT * pFont);
```

Parameter	Meaning
hScaleObj	Handle of scale object.
pFont	Font to be used.

Return value

Previous used font used to draw the numbers.

GRAPH_SCALE_SetNumDecs()

Before	After
	

Description

Sets the number of post decimal positions to be shown.

Prototype

```
int GRAPH_SCALE_SetNumDecs(GRAPH_SCALE_Handle hScaleObj, int NumDecs);
```

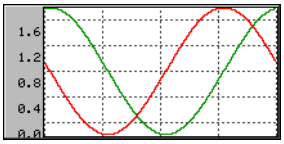
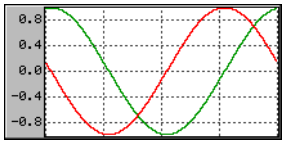
Parameter	Meaning
hScaleObj	Handle of scale object.
NumDecs	Number of post decimal positions.

Return value

Previous number of post decimal positions.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GRAPH_SCALE_SetOff()

Before	After
	

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
hScaleObj	Handle of scale object.
Off	Offset used for drawing the scale.

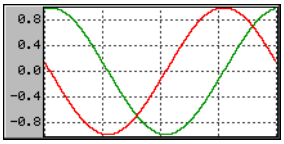
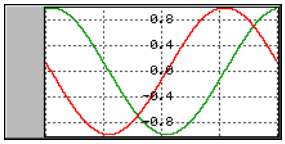
Return value

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
hScaleObj	Handle of scale object.
Pos	Position, at which the scale should be shown.

Return value

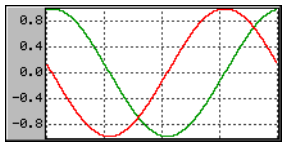
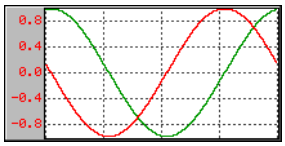
Previous position of the scale object.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

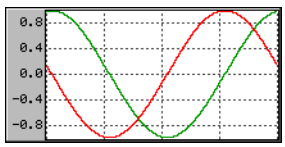
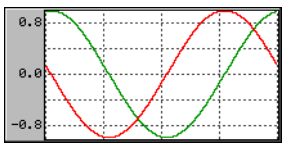
```
GUI_COLOR GRAPH_SCALE_SetTextColor(GRAPH_SCALE_Handle hScaleObj,
                                   GUI_COLOR Color);
```

Parameter	Meaning
hScaleObj	Handle of scale object.
Color	Color to be used to show the numbers.

Return value

Previous color used to show the numbers.

GRAPH_SCALE_SetTickDist()

Before	After
	

Description

Sets the distance from one number to the next.

Prototype

```
unsigned GRAPH_SCALE_SetTickDist(GRAPH_SCALE_Handle hScaleObj,
                                 unsigned Dist);
```

Parameter	Meaning
hScaleObj	Handle of scale object.
Dist	Distance in pixels between the numbers.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Previous distance between the numbers.

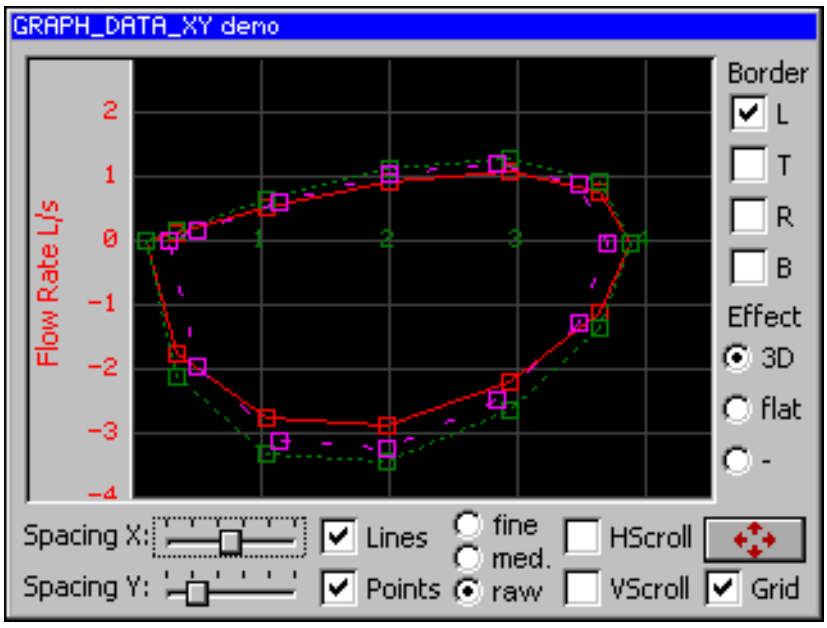
15.9.8 Examples

The sample folder contains the following samples which show how the widget can be used:

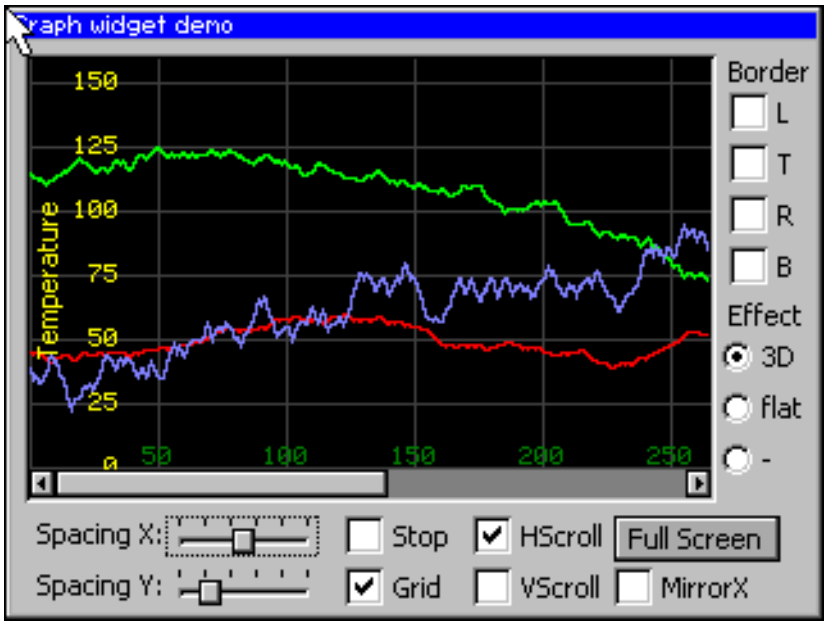
- `WIDGET_GraphXY.c`
- `WIDGET_GraphYT.c`

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screen shot of WIDGET_GraphXY.c:



Screen shot of WIDGET_GraphYT.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.10 HEADER: Header widget

HEADER widgets are used to label columns of a table:



If a pointer input device (PID) is used, the width of the header items can be managed by dragging the dividers by the PID.

Behavior with mouse

If mouse support is enabled, the cursor is on and the PID is moved nearby a divider the cursor will change to signal, that the divider can be dragged at the current position.

Behavior with touch screen

If the widget is pressed nearby a divider and the cursor is on the cursor will change to signal, that the divider can now be dragged.

Screenshot of drag-able divider



Predefined cursors

There are 2 predefined cursors as shown below:

GUI_CursorHeaderM (default)	GUI_CursorHeaderMI
	

You can also create and use your own cursors when using a HEADER widget as described later in this chapter.

15.10.1 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

15.10.2 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.10.3 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.10.4 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_SetBMPEx()	Sets the bitmap used when displaying the given item.
HEADER_SetDefaultBkColor()	Sets the default background color.
HEADER_SetDefaultBorderH()	Sets the default value for the horizontal spacing.
HEADER_SetDefaultBorderV()	Sets the default value for the vertical spacing.
HEADER_SetDefaultCursor()	Sets the default cursor.
HEADER_SetDefaultFont()	Sets the default font.
HEADER_SetDefaultTextColor()	Sets the default text color.
HEADER_SetDragLimit()	Sets the limit for dragging the items on or off.
HEADER_SetFont()	Sets the font of the widget.
HEADER_SetHeight()	Sets the height of the widget.
HEADER_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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                    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 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 of the created HEADER widget; 0 if the routine fails.

HEADER_CreateAttached()

Description

Creates a HEADER widget which is attached to an existing window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
HEADER_Handle HEADER_CreateAttached(WM_HWIN hParent,
                                    int Id,
                                    int SpecialFlags);
```

Parameter	Meaning
hObj	Handle of widget
Id	Id of the HEADER widget
SpecialFlags	(Not used, reserved for later use)

Return value

Handle of 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
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xsize	Horizontal size of the widget (in pixels).
ysize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new HEADER widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (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 of 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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 diplayed when dragging the width of an item.

Prototype

```
const GUI_CURSOR * HEADER_GetDefaultCursor(void);
```

Return value

pointer to the cursor diplayed 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                        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(HEADER_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:

```
...
HEADER_Handle hHeader;
GUI_Init();
HEADER_SetDefaultTextColor(GUI_YELLOW);
HEADER_SetDefaultFont(&GUI_Font8x8);
hHeader = HEADER_Create(10, 10, 100, 40, WM_HBKWIN, 1234, WM_CF_SHOW, 0);
HEADER_AddItem(hHeader, 50, "Phone", GUI_TA_BOTTOM | GUI_TA_HCENTER);
HEADER_AddItem(hHeader, 50, "Code" , GUI_TA_BOTTOM | GUI_TA_HCENTER);
HEADER_SetBitmapEx(hHeader, 0, &bmPhone, 0, -15);
HEADER_SetBitmapEx(hHeader, 1, &bmCode,  0, -15);
...
```

Screenshot of example above:



HEADER_SetBkColor()

Description

Sets the background color of the given HEADER widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetDragLimit()

Description

Sets the limit for dragging the dividers on or off. If the limit is on, a divider can only be dragged within the widget area. If the limit is off, it can be dragged outside the widget.

Prototype

```
void HEADER_SetDragLimit(HEADER_Handle hObj, unsigned OnOff);
```

Parameter	Meaning
hObj	Handle of widget
OnOff	1 for setting the drag limit on, 0 for off.

HEADER_SetFont()

Description

Sets the font used when displaying the given HEADER widget

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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

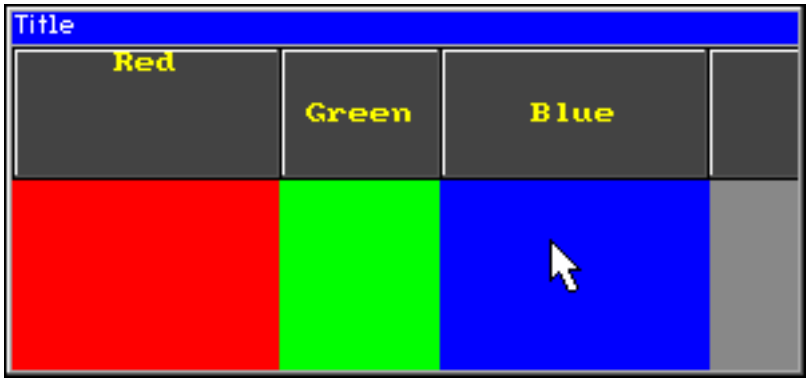
15.10.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_Header.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

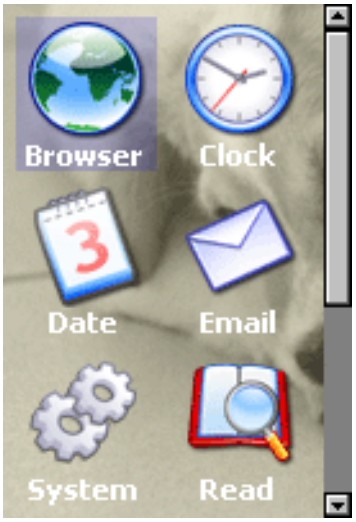
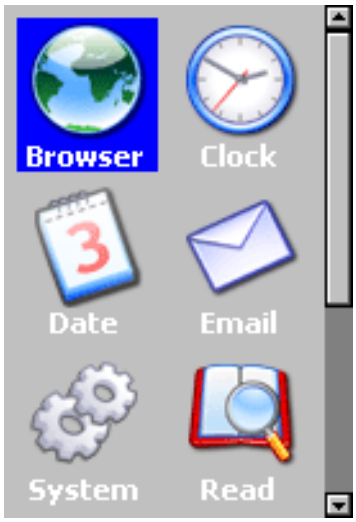
Screen shot of WIDGET_Header.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.11 ICONVIEW: Icon view widget

The iconview widget can be used for icon based menus often required in hand held devices like mobile telephones or pocket organizers. It shows a list of icons where each icon can be labeled with optional text. Iconview widgets support transparency and alpha blending. So any content can be shown in the background. The currently selected icon can be highlighted by a solid color or with an alpha blending effect, which lets the background shine through. If required a scrollbar can be shown.

ICONVIEW with transparency	ICONVIEW without transparency
	

All ICONVIEW-related routines are in the file(s) `ICONVIEW*.c`, `ICONVIEW*.h`. All identifiers are prefixed ICONVIEW.

15.11.1 Configuration options

Type	Macro	Default	Explanation
N	ICONVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	ICONVIEW_BKCOLOR1_DEFAULT	GUI_BLUE	Background color, selected state.
N	ICONVIEW_TEXTCOLOR0_DEFAULT	GUI_WHITE	Text color, unselected state.
N	ICONVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color, selected state.
S	ICONVIEW_FONT_DEFAULT	GUI_Font13_1	Font to be used for drawing the labels.
N	ICONVIEW_FRAMEX_DEFAULT	5	Free space between the icons and the left and right border of the widget.
N	ICONVIEW_FRAMEY_DEFAULT	5	Free space between the icons and the top and bottom border of the widget.
N	ICONVIEW_SPACEX_DEFAULT	5	Free horizontal space between the icons.
N	ICONVIEW_SPACEY_DEFAULT	5	Free vertical space between the icons.
N	ICONVIEW_ALIGN_DEFAULT	GUI_TA_HCENTER | GUI_TA_BOTTOM	Default alignment to be used for drawing the labels.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.11.2 Notification codes

The following events are sent from an ICONVIEW 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 area without releasing.
WM_NOTIFICATION_SCROLL_CHANGED	The scroll position of the optional scrollbar has been changed.
WM_NOTIFICATION_SEL_CHANGED	The selection of the widget has been changed.

15.11.3 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

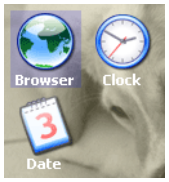
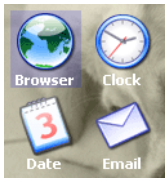
Key	Reaction
GUI_KEY_RIGHT	Moves the selection to the next icon.
GUI_KEY_LEFT	Moves the selection to the previous icon.
GUI_KEY_DOWN	Moves the selection down.
GUI_KEY_UP	Moves the selection up.
GUI_KEY_HOME	Moves the selection to the first icon.
GUI_KEY_END	Moves the selection to the last icon.

15.11.4 ICONVIEW API

The table below lists the available emWin ICONVIEW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
ICONVIEW_AddBitmapItem()	Adds a new icon to the widget.
ICONVIEW_CreateEx()	Creates a new iconview widget.
ICONVIEW_GetSel()	Returns the index of the currently selected icon.
ICONVIEW_SetBkColor()	Sets the background color.
ICONVIEW_SetFont()	Sets the font to be used for drawing the labels.
ICONVIEW_SetSel()	Sets the current selection.
ICONVIEW_SetTextColor()	Sets the color to be used to draw the labels.

ICONVIEW_AddBitmapItem()

Before	After
	

Description

Adds a new bitmap icon to the widget.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int ICONVIEW_AddBitmapItem(ICONVIEW_Handle hObj,
                           const GUI_BITMAP * pBitmap,
                           const char * pText);
```

Parameter	Meaning
hObj	Handle of widget.
pBitmap	Pointer to a bitmap structure used to draw the icon.
pText	Text to be used to label the icon.

Return value

0 on success, !=0 on error.

Add. information

Please note that the bitmap pointer needs to remain valid.

ICONVIEW_CreateEx()

Description

Creates a new imageview widget of the given size.

Prototype

```
ICONVIEW_Handle ICONVIEW_CreateEx(int x0, int y0,
                                  int xSize, int ySize,
                                  WM_HWIN hParent, int WinFlags,
                                  int ExFlags, int Id,
                                  int xSizeItem, int ySizeItem);
```

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 widget will be a child window of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (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.
xSizeItem	Horizontal icon size in pixels.
ySizeItem	Vertical icon size in pixels.

Permitted values for parameter ExFlags	
0	(default)
ICONVIEW_CF_AUTOSCROLLBAR_V	A vertical scrollbar will be added if the widget area is too small to show all icons.

Return value

Handle of the new widget, 0 if the routine fails.

Add. information

If the widget should be transparent, the parameter WinFlags should be or-combined with WM_CF_HASTRANS.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

ICONVIEW_GetSel()

Description

Returns the zero based index of the currently selected icon.

Prototype

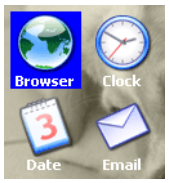
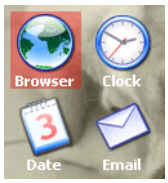
```
int ICONVIEW_GetSel(ICONVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Return value

Zero based index of the currently selected icon.

ICONVIEW_SetBkColor()

Before	After
	

Description

Sets the background color of the widget.

Prototype

```
void ICONVIEW_SetBkColor(ICONVIEW_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Index	(see table below)
Color	Color to be used for drawing the background.

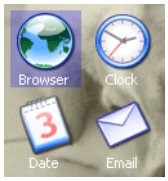
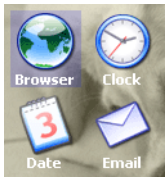
Permitted values for parameter Index	
ICONVIEW_CI_BK	Color used to draw the widget background.
ICONVIEW_CI_SEL	Color used to highlight the currently selected item.

Add. information

The upper 8 bits of the 32 bit color value can be used for an alpha blending effect. For more details about alpha blending please refer to the function GUI_SetAlpha().

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

ICONVIEW_SetFont()

Before	After
	

Description

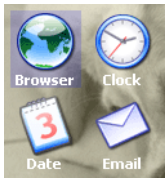
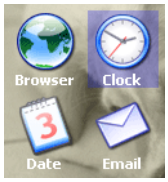
Sets the font to be used for drawing the icon labels.

Prototype

```
void ICONVIEW_SetFont(ICONVIEW_Handle hObj,
                      const GUI_FONT GUI_UNI_PTR * pFont);
```

Parameter	Meaning
hObj	Handle of widget.
pFont	Pointer to a GUI_FONT structure to be used to draw the icon labels.

ICONVIEW_SetSel()

Before	After
	

Description

Sets the current selection.

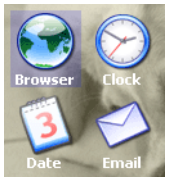
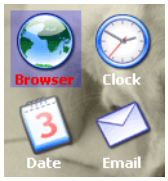
Prototype

```
void ICONVIEW_SetSel(ICONVIEW_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of widget.
Sel	New selection.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

ICONVIEW_SetTextColor()

Before	After
	

Description

Sets the color to be used to draw the lables.

Prototype

```
void ICONVIEW_SetTextColor(ICONVIEW_Handle hObj,
                           int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Index	(see table below)
Color	Color to be used

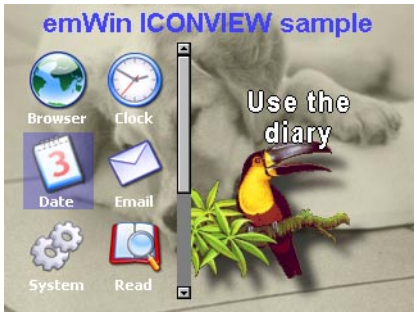
Permitted values for parameter Index	
ICONVIEW_CI_UNSEL	Color used to draw the labels in unselected state.
ICONVIEW_CI_SEL	Color used to draw the labels in selected state.

15.11.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_IconView

Screen shot of WIDGET_Header.c:



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.12 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.

15.12.1 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.

15.12.2 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 off 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.

15.12.3 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

15.12.4 LISTBOX API

The table below lists the available emWin LISTBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
LISTBOX_AddString()	Adds an item to a list box.
LISTBOX_Create()	Creates the list box. (Obsolete)
LISTBOX_CreateAsChild()	Creates the list box as a child window. (Obsolete)
LISTBOX_CreateEx()	Creates the list box.
LISTBOX_CreateIndirect()	Creates the list box from resource table entry.
LISTBOX_DecSel()	Decrements selection.
LISTBOX_DeleteItem()	Deletes an element.
LISTBOX_GetDefaultBkColor()	Returns the default background color for LISTBOX widgets.
LISTBOX_GetDefaultFont()	Returns the default font for LISTBOX widgets.
LISTBOX_GetDefaultScrollStepH()	Returns the default number of pixels to be scrolled horizontal.
LISTBOX_GetDefaultTextAlign()	Returns the default text alignment for new list boxes.
LISTBOX_GetDefaultTextColor()	Returns the default text color for new list boxes.
LISTBOX_GetFont()	Returns the font of the list box.
LISTBOX_GetItemDisabled()	Returns the disabled state of the given item.
LISTBOX_GetItemSel()	Returns the selection state of a LISTBOX entry.
LISTBOX_GetItemText()	Returns the text of a list box entry.
LISTBOX_GetMulti()	Returns if the multi select mode is active.
LISTBOX_GetNumItems()	Returns the number of items in a list box.
LISTBOX_GetScrollStepH()	Returns the number of pixels to be scrolled horizontal.
LISTBOX_GetSel()	Returns the number of the selected item.
LISTBOX_GetTextAlign()	Returns the text alignment of the LISTBOX.
LISTBOX_IncSel()	Increments selection.
LISTBOX_InsertString()	Inserts an element.
LISTBOX_InvalidateItem()	Invalidates an item of an owner drawn LISTBOX.
LISTBOX_OwnerDraw()	Default function for drawing a LISTBOX entry.
LISTBOX_SetAutoScrollH()	Activates automatic use of a horizontal scrollbar.
LISTBOX_SetAutoScrollV()	Activates automatic use of a vertical scrollbar.
LISTBOX_SetBkColor()	Sets the background color.
LISTBOX_SetDefaultBkColor()	Sets the default background color for LISTBOX widgets.
LISTBOX_SetDefaultFont()	Changes the default font for LISTBOX widgets.
LISTBOX_SetDefaultScrollStepH()	Sets the default number of pixels to be scrolled horizontal.
LISTBOX_SetDefaultTextAlign()	Sets the default text alignment for new LISTBOX widgets.
LISTBOX_SetDefaultTextColor()	Sets the default text color for LISTBOX widgets.
LISTBOX_SetFont()	Selects the font.
LISTBOX_SetItemDisabled()	Sets the disabled state of the given item.
LISTBOX_SetItemSel()	Sets the selection state of the given item.
LISTBOX_SetItemSpacing()	Sets a spacing between the items.
LISTBOX_SetMulti()	Sets the multi selection mode on or off.
LISTBOX_SetOwnerDraw()	Enables the list box to be owner drawn.
LISTBOX_SetScrollbarColor()	Sets the colors of the optional scrollbar.
LISTBOX_SetScrollbarWidth()	Sets the width of the scrollbars used by the LISTBOX.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    int xSize, int ySize, int Flags);
```

Parameter	Meaning
ppText	Pointer to an array of string pointers containing the elements to be displayed.
hParent	Handle of parent window.
x0	X-position of the list box relative to the parent window.
y0	Y-position of the list box relative to the parent window.
xSize	Horizontal size of the list box (in pixels).
ySize	Vertical size of the list box (in pixels).
Flags	Window create flags (see LISTBOX_Create()).

Return value

Handle of 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
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xsize	Horizontal size of the widget (in pixels).
ysize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new HEADER widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (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.
ppText	Pointer to an array of string pointers containing the elements to be displayed.

Return value

Handle of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Pointer to the default font.

LISTBOX_GetDefaultScrollStepH()

Description

Returns the default horizontal scroll step used for creating LISTBOX widgets. The horizontal scroll step defines the number of pixels to be scrolled if needed.

Prototype

```
int LISTBOX_GetDefaultScrollStepH(void);
```

Return value

Default horizontal scroll step.

LISTBOX_GetDefaultTextAlign()

Description

Returns the default text alignment for new LISTBOX widgets.

Prototype

```
int LISTBOX_GetDefaultTextAlign(void);
```

Return value

Default text alignment for new LISTBOX widgets.

Add. information

For more information please refer to `LISTBOX_SetTextAlign()`.

LISTBOX_GetDefaultTextColor()

Description

Returns the default text color for new LISTBOX widgets.

Prototype

```
GUI_COLOR LISTBOX_GetDefaultTextColor(unsigned Index);
```

Parameter	Meaning
Index	Zero based index for text 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 text color for new LISTBOX widgets.

LISTBOX_GetFont()

Description

Returns a pointer to the font used to display the text of the list box.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of list box.
OnOff	(See table below)

Permitted values for parameter OnOff	
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
hObj	Handle of list box.
Index	Index for background color (see table below).
Color	Color to be set.

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.
LISTBOX_CI_DISABLED	Disabled element.

LISTBOX_SetDefaultBkColor()

Description

Sets the default background color for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultBkColor(unsigned Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Zero based index for background color (see table below)
Color	Desired background color.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetDefaultTextAlign()

Description

Sets the default text alignment for new LISTBOX widgets.

Prototype

```
void LISTBOX_SetDefaultTextAlign(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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Index	
LISTBOX_CI_UNSEL	Unselected element.
LISTBOX_CI_SEL	Selected element, without focus.
LISTBOX_CI_SELFOCUS	Selected element, with focus.

LISTBOX_SetFont()

Description

Sets the 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);
```

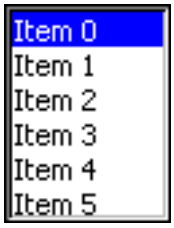
Parameter	Meaning
hObj	Handle of list box.
Index	Zero based index of the listbox item.
OnOff	1 for selected, 0 for not selected.

Add. information

Setting the selection state of a list box item makes only sense when using the multi selection mode. See also LISTBOX_SetMulti().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 entrys are 30 pixels wider than and have the same height as the item drawn by the default function:

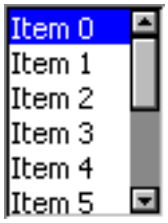
```
static int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_GET_XSIZE:
    return LISTBOX_OwnerDraw(pDrawItemInfo) + 30; /* Returns the default xsize+10 */
  case WIDGET_ITEM_DRAW:

    /* 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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

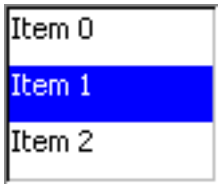
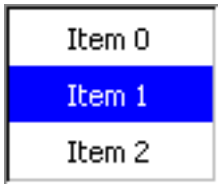
```
void LISTBOX_SetString(LISTBOX_Handle hObj, const char* s,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                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()

Before	After
	

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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of list box.
Index	Index for text color (see LISTBOX_SetBackColor()).
Color	Color to be set.

15.12.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_SimpleListBox.c
- WIDGET_ListBox.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_SimpleListBox.c:



Screenshot(s) of WIDGET_ListBox.c:

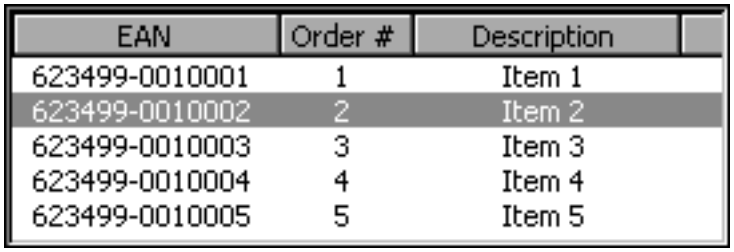
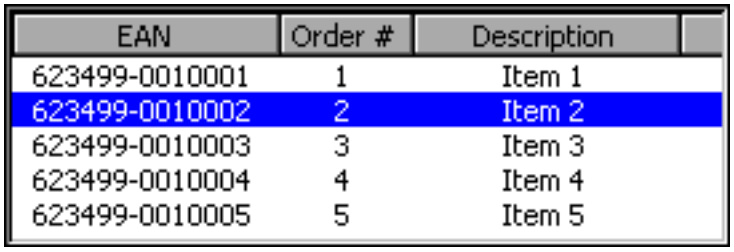
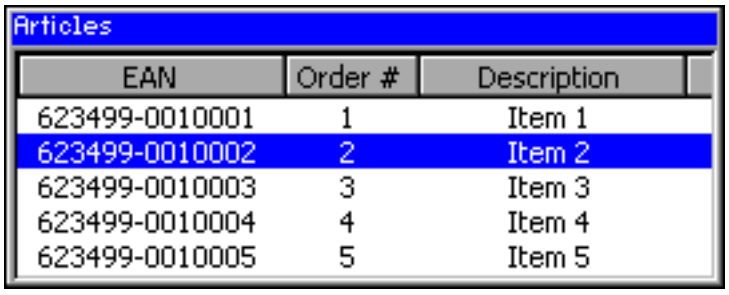
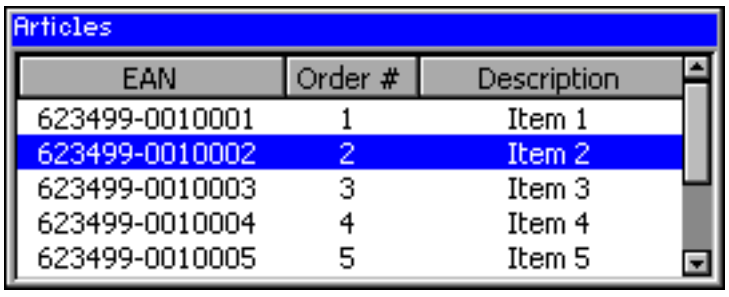




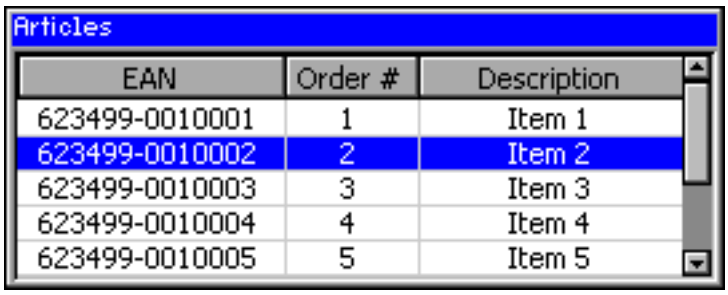
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.13 LISTVIEW: Listview widget

LISTVIEW widgets are used to select one element of a list with several columns. To manage the columns a LISTWIEW 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	
Has input focus No surrounding FRAMEWIN No SCROLLBAR attached Grid lines not visible	
Has input focus With surrounding FRAMEWIN No SCROLLBAR attached Grid lines not visible	
Has input focus With surrounding FRAMEWIN SCROLLBAR attached Grid lines not visible	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Explanation	LISTVIEW widget
Has input focus With surrounding FRAMEWIN SCROLLBAR attached Grid lines visible	

15.13.1 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_SCROLLSTEP_H_DEFAULT	10	Defines the number of pixels to be scrolled if needed.
N	LISTVIEW_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	LISTVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color, selected state 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

15.13.2 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.

15.13.3 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_UP	Moves the selection bar up.
GUI_KEY_DOWN	Moves the selection bar down.
GUI_KEY_RIGHT	If the total amount of the column width is > than the inside area of the listview, the contents scrolles to the left.
GUI_KEY_LEFT	If the total amount of the column width is > than the inside area of the listview, the contents scrolles to the right.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.13.4 LISTVIEW API

The table below lists the available emWin LISTVIEW-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
LISTVIEW_AddColumn()	Adds a column to a LISTVIEW.
LISTVIEW_AddRow()	Adds a row to a LISTVIEW.
LISTVIEW_CompareDec()	Compare function for comparing 2 integer values.
LISTVIEW_CompareText()	Compare function for comparing 2 strings.
LISTVIEW_Create()	Creates a LISTVIEW widget. (Obsolete)
LISTVIEW_CreateAttached()	Creates a LISTVIEW widget attached to a window.
LISTVIEW_CreateEx()	Creates a LISTVIEW widget.
LISTVIEW_CreateIndirect()	Creates a LISTVIEW widget from a resource table entry.
LISTVIEW_DecSel()	Decrements selection.
LISTVIEW_DeleteColumn()	Deletes the given column.
LISTVIEW_DeleteRow()	Deletes the given row.
LISTVIEW_DisableRow()	Sets the state of the given row to disabled.
LISTVIEW_DisableSort()	Disables sorting of the LISTVIEW.
LISTVIEW_EnableRow()	Sets the state of the given row to enabled.
LISTVIEW_EnableSort()	Enables sorting of the LISTVIEW.
LISTVIEW_GetBkColor()	Returns the background color of the LISTVIEW.
LISTVIEW_GetFont()	Returns the font of the LISTVIEW.
LISTVIEW_GetHeader()	Returns the handle of the attached HEADER widget.
LISTVIEW_GetItemText()	Returns the text of the given cell.
LISTVIEW_GetNumColumns()	Returns the number of columns.
LISTVIEW_GetNumRows()	Returns the number of rows.
LISTVIEW_GetSel()	Returns the number of the selected item.
LISTVIEW_GetSelUnsorted()	Returns the number of the selected item in unsorted state.
LISTVIEW_GetTextColor()	Returns the text color of the LISTVIEW.
LISTVIEW_GetUserData()	Returns the user data of the given row.
LISTVIEW_IncSel()	Increments selection.
LISTVIEW_InsertRow()	Inserts a new row at the given position.
LISTVIEW_SetAutoScrollH()	Enables the automatic use of a horizontal scrollbar.
LISTVIEW_SetAutoScrollV()	Enables the automatic use of a vertical scrollbar.
LISTVIEW_SetBkColor()	Sets the background color.
LISTVIEW_SetColumnWidth()	Sets the column width.
LISTVIEW_SetCompareFunc()	Sets the compare function for the given column.
LISTVIEW_SetDefaultBkColor()	Sets the default background color for HEADER widgets.
LISTVIEW_SetDefaultFont()	Sets the default font for HEADER widgets.
LISTVIEW_SetDefaultGridColor()	Sets the default text color for HEADER widgets.
LISTVIEW_SetDefaultTextColor()	Sets the default color of the grid lines for HEADER widgets.
LISTVIEW_SetFixed()	Fixes the given number of columns.
LISTVIEW_SetFont()	Sets the font of the LISTVIEW.
LISTVIEW_SetGridVis()	Sets the visibility flag of the grid lines.
LISTVIEW_SetItemBkColor()	Sets the background color of a LISTVIEW cell
LISTVIEW_SetItemText()	Sets the text of a LISTVIEW cell.
LISTVIEW_SetItemTextColor()	Sets the text color of a LISTVIEW cell
LISTVIEW_SetLBorder()	Sets the number of pixels used for the left border.
LISTVIEW_SetRBorder()	Sets the number of pixels used for the right border.
LISTVIEW_SetRowHeight()	Sets the row height of the LISTVIEW
LISTVIEW_SetSel()	Sets the current selection.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
LISTVIEW_SetSelUnsorted()	Sets the current selection in unsorted state.
LISTVIEW_SetSort()	Sets the column and sorting order to be sorted by.
LISTVIEW_SetTextAlign()	Sets the text alignment of a column.
LISTVIEW_SetTextColor()	Sets the text color.
LISTVIEW_SetUserData()	Sets the user data of the given row.

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. information

The ppText-array should contain one item for each column. If it contains less items the remaining cells left blank.

LISTVIEW_CompareDec()

Description

Compare function for comparing 2 integer values.

Prototype

```
int LISTVIEW_CompareDec(const void * p0, const void * p1);
```

Parameter	Meaning
p0	Void pointer to first value:
p1	Void pointer to second value.

Return value

< 0 if value of cell 0 greater than value of cell 1.
0 if value of cell 0 identical to value of cell 1.
> 0 if value of cell 0 less than value of cell 1.

Add. information

The purpose of this function is to be used by the listviews sorting algorithm if the cell text represens integer values.
For details about how to use this function for sorting please also refer to LISTVIEW_SetCompareFunc().
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_CompareText()

Description

Compare function for comparing 2 integer values.

Prototype

```
int LISTVIEW_CompareText(const void * p0, const void * p1);
```

Parameter	Meaning
p0	Void pointer to first text:
p1	Void pointer to second text.

Return value

> 0 if text of cell 0 greater than text of cell 1.
0 if text of cell 0 identical to text of cell 1.
< 0 if text of cell 0 less than text of cell 1.

Add. information

The purpose of this function is to be used by the listviews sorting algorithm.
For details about how to use this function for sorting please also refer to LISTVIEW_SetCompareFunc().
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_Create()

(Obsolete, LISTVIEW_CreateEx should be used instead)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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 of 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    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).
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 of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LISTVIEW_DeleteRow()

Description

Deletes the specified row of the listview.

Prototype

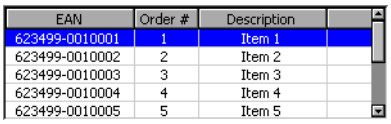
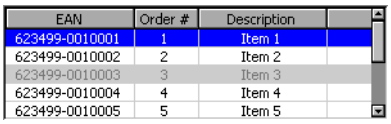
```
void LISTVIEW_DeleteRow(LISTVIEW_Handle hObj, unsigned Index);
```

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_DisableSort()

Description

Disables sorting of the given listview. After calling this function the contents of the listview will be shown unsorted.

Prototype

```
void LISTVIEW_DisableSort(LISTVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

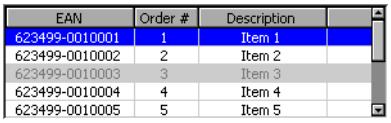
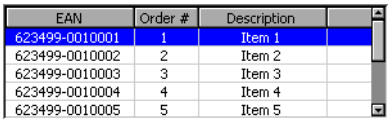
Add. information

For details about how to use sorting in listview widgets please also refer to the functions `LISTVIEW_SetCompareFunc()` and `LISTVIEW_SetSort()`.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

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_EnableSort()

Description

Enables sorting for the given listview. After calling this function the contents of the listview can be rendered sorted after clicking on the header item of the desired column, by which the listview should sort its data. Please note, that this works only after a compare function for the desired column has been set.

Prototype

```
void LISTVIEW_EnableSort(LISTVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Add. information

For details about how to set a compare function please refer to LISTVIEW_SetCompareFunc().
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_GetBkColor()

Description

Returns the background color of the given listview.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
GUI_COLOR LISTVIEW_GetBkColor(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

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
hObj	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
hObj	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void LISTVIEW_GetItemText(LISTVIEW_Handle hObj,
                          unsigned Column, unsigned Row,
                          char* pBuffer, unsigned MaxSize);
```

Parameter	Meaning
hObj	Handle of widget
Column	Zero based index of the cell's column.
Row	Zero based index of the cell's row
pBuffer	Pointer to a buffer to be filled by the routine.
MaxSize	Size in bytes of the buffer.

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
hObj	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
hObj	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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Number of the currently selected row.

LISTVIEW_GetSelUnsorted()

Description

Returns the index of the currently selected row in unsorted state.

Prototype

```
int LISTVIEW_GetSelUnsorted(LISTVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget

Return value

Index of the currently selected row in unsorted state.

Add. information

This function returns the actually index of the selected row, whereas the function LISTVIEW_GetSel() only returns the index of the sorted row. The actuall (unsorted) row index should be used in function calls as row index.
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

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_GetUserData()

Description

Returns the user data of the given row.

Prototype

```
U32 LISTVIEW_GetUserData(LISTVIEW_Handle hObj, unsigned Row);
```

Parameter	Meaning
hObj	Handle of widget
Row	Zero based index of row.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

User data of the given row.

Add. information

For details about how to set user data of a row please refer to the function LISTVIEW_SetUserData().

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_InsertRow()

Description

Inserts a new row into the listview at the given position.

Prototype

```
int LISTVIEW_InsertRow(LISTVIEW_Handle hObj,
                       unsigned Index,
                       const GUI_ConstString * ppText);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index of the new row.
ppText	Pointer to a string array containing the cell data of the new row.

Return value

0 if function succeed, 1 if an error occures.

Add. information

The ppText-array should contain one item for each column. If it contains less items the remaining cells left blank.
If the given index is >= the current number of rows, the function LISTVIEW_AddRow() will be used to add the new row.
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_SetAutoScrollH()

Description

Enables/disables the automatic use of a horizontal scrollbar.

Prototype

```
void LISTVIEW_SetAutoScrollH(LISTVIEW_Handle hObj, int OnOff);
```

Parameter	Meaning
hObj	Handle of widget
OnOff	(see table below)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 listview checks if all columns fit into the widgets area. If not a horizontal scrollbar will be added.

LISTVIEW_SetAutoScrollV()

Description

Enables/disables the automatic use of a vertical scrollbar.

Prototype

```
void LISTVIEW_SetAutoScrollV(LISTVIEW_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 vertical scrollbar.
1	Enable automatic use of a vertical scrollbar.

Add. information

If enabled the listview checks if all rows fit into the widgets area. If not a vertical scrollbar will be added.

LISTVIEW_SetBkColor()

Description

Sets the background color of the given LISTVIEW widget.

Prototype

```
void LISTVIEW_SetBkColor(LISTVIEW_Handle hObj,
                         unsigned int Index,
                         GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget
Index	Index for background 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.
LISTVIEW_CI_DISABLED	Disabled element.

Add. Information

To set the background color for a single cell the function LISTVIEW_SetItemBkColor() should be used.
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LISTVIEW_SetColumnWidth()

Description

Sets the width of the given column.

Prototype

```
void LISTVIEW_SetColumnWidth(LISTVIEW_Handle hObj,
                             unsigned int Index,
                             int Width);
```

Parameter	Meaning
hObj	Handle of widget
Index	Number of column
Width	New width

LISTVIEW_SetCompareFunc()

Description

Sets the compare function for the given column. A compare function needs to be set if the listview widget should be sorted by the given column.

Prototype

```
void LISTVIEW_SetCompareFunc(LISTVIEW_Handle hObj,
                     unsigned Column,
                     int (* fpCompare)(const void * p0, const void * p1));
```

Parameter	Meaning
hObj	Handle of widget
Column	Index of the desired column for which the compare function should be set.
fpCompare	Function pointer to compare function.

Add. information

If the sorting feature of the listview widget is used, the widget uses a compare function to decide if the contents of one cell is greater, equal or less than the contents of the other cell.
Per default no compare function is set for the listview columns. For each column which should be used for sorting, a compare function needs to be set.
The cells of the listview widget contain text. But sometimes the text represents data of other types like dates, integers or others. So different compare functions are required for sorting. emWin provides 2 compare functions:

`LISTVIEW_CompareText()`: Function can be used for comparing cells containing text.

`LISTVIEW_CompareDec()`: Function can be used for comparing cells which text, where the contents represents integer values.

The compare function should return a value >0, if the contents of the second cell is greater than the contents of the first cell and <0, if the contents of the second cell is less than the contents of the first cell or 0 if equal.
Also user defined compare functions can be used. The prototype of a application-defined function should be defined as follows:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int APPLICATION_Compare(const void * p0, const void * p1);
```

Parameter	Meaning
p0	Pointer to NULL terminated string data of the first cell.
p1	Pointer to NULL terminated string data of the second cell.

Example

```
int APPLICATION_Compare(const void * p0, const void * p1) {
  return strcmp((const char *)p1, (const char *)p0);
}

void SetAppCompareFunc(WM_HWIN hListView, int Column) {
  LISTVIEW_SetCompareFunc(hListView, Column, APPLICATION_Compare);
}
```

The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_SetDefaultBkColor()

Description

Sets the default background color for new LISTVIEW widgets.

Prototype

```
GUI_COLOR LISTVIEW_SetDefaultBkColor(unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Index of default background color (see table below)
Color	Color to be set as default

Permitted values for parameter Index	
LISTVIEW_CI_UNSEL	Unselected element.
LISTVIEW_CI_SEL	Selected element, without focus.
LISTVIEW_CI_SELFOCUS	Selected element, with focus.
LISTVIEW_CI_DISABLED	Disabled element.

Return value

Previous default 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetFixed()

Description

Fixes the given number of columns at their horizontal positions.

Prototype

```
unsigned LISTVIEW_SetFixed(LISTVIEW_Handle hObj, unsigned Fixed);
```

Parameter	Meaning
hObj	Handle of listview.
Fixed	Number of columns to be fixed at their horizontal positions.

Add. Information

Using this function makes sense if one or more columns should remain at their horizontal positions during scrolling operations.

LISTVIEW_SetFont()

Description

Sets the listview font.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
```

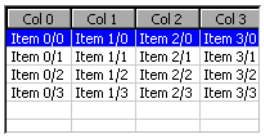
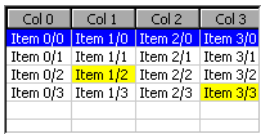
Parameter	Meaning
hObj	Handle of widget
Show	Sets the visibility of the grid lines

Permitted values for parameter Show	
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
hObj	Handle of widget
Column	Number of column.
Row	Number of row.
Index	Index of background color (see table below).
Color	Color to be used.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 background color for the given cell set by LISTVIEW_SetBkColor().

LISTVIEW_SetItemText()

Description

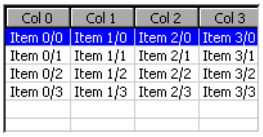
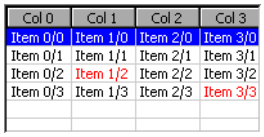
Sets the text of one cell of the LISTVIEW widget specified by row and column.

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
	

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.

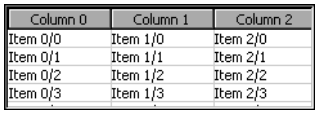
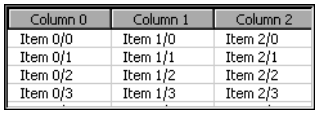
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
	

Description

Sets the number of pixels used for the left border within each cell of the listview.

Prototype

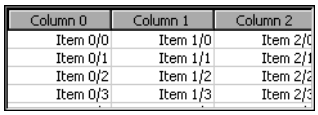
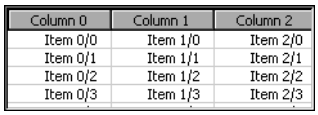
```
void LISTVIEW_SetLBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

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
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetSelUnsorted()

Description

Sets the index of the currently selected row in unsorted state.

Prototype

```
void LISTVIEW_SetSelUnsorted(LISTVIEW_Handle hObj, int Sel);
```

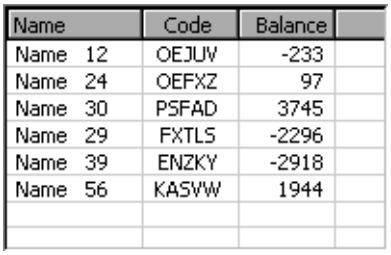
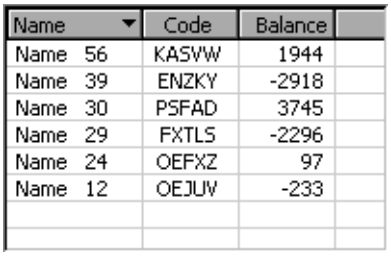
Parameter	Meaning
hObj	Handle of widget.
Sel	Zero based selection index in unsorted state.

Add. information

This function sets the actually index of the selected row, whereas the function LISTVIEW_SetSel() sets the index of the sorted row. The actuall (unsorted) row index should be used in function calls as row index.
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LISTVIEW_SetSort()

Before	After
	

Description

This function sets the column to be sorted by and the sorting order.

Prototype

```
unsigned LISTVIEW_SetSort(LISTVIEW_Handle hObj,
                          unsigned Column, unsigned Reverse);
```

Parameter	Meaning
hObj	Handle of widget.
Column	Column to be sorted by.
Reverse	0 for normal sorting order (greatest element at the top), 1 for reverse order.

Return value

0 if function was successfully, 1 if not.

Add. information

Before calling this function a compare function needs to be set for the desired column. For details about how to set a compare function please refer to the function LISTVIEW_SetCompareFunc().
The sample folder contains the sample WIDGET_SortedListview.c which shows how to use the function.

LISTVIEW_SetTextAlign()

Description

Sets the alignment for the given column.

Prototype

```
void LISTVIEW_SetTextAlign(LISTVIEW_Handle hObj,
                           unsigned int Index,
                           int Align);
```

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

LISTVIEW_SetUserData()

Description

Sets the user data of the given row.

Prototype

```
void LISTVIEW_SetUserData(LISTVIEW_Handle hObj, unsigned Row, U32 UserData);
```

Parameter	Meaning
hObj	Handle of widget.
Row	Row for which the user data should be set.
UserData	Value to be associated with the row.

Add. information

Sets the 32-bit value associated with the row. Each row has a corresponding 32-bit value intended for use by the application.

15.13.5 Example

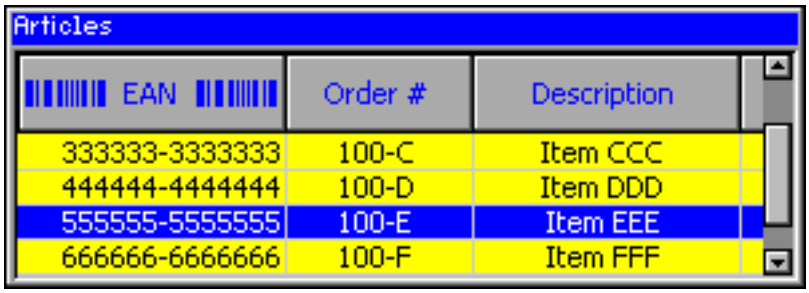
The sample folder contains the following samples which show how the widget can be used:

- WIDGET_ListView.c

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_ListView.c:



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.14 LISTWHEEL: Listwheel widget

This widget is similar to the LISTBOX widget described earlier in this chapter. Whereas the data of a LISTBOX is selected by moving the cursor with the keyboard or by using a SRCOLLBAR the LISTWHEEL works completely different: The whole data area can be moved via pointer input device (PID). Striking over the widget from top to bottom or vice versa moves the data up or downwards. When releasing the PID during the data area is moving it slows down its motion and stops by snapping in a new item at the snap position. Further the data is shown in a loop. After the last data item it continues with the first item like in a chain. So the data can be 'rotated' like on a wheel:

Explanation	LISTWHEEL widget
Application sample showing three wheels for selecting a date. The sample uses the owner draw mechanism to overlay the widget with a customized alpha mask for the shading effect.	

The table above shows a screenshot of the the sample WIDGET_ListWheel.c located in the sample folder `Sample\Tutorial\` of emWin.

15.14.1 Configuration options

Type	Macro	Default	Explanation
S	LISTWHEEL_FONT_DEFAULT	GUI_Font13_1	Font used.
N	LISTWHEEL_BKCOLOR0_DEFAULT	GUI_WHITE	Background color of normal text.
N	LISTWHEEL_BKCOLOR1_DEFAULT	GUI_WHITE	Background color of selected text.
N	LISTWHEEL_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color of normal text.
N	LISTWHEEL_TEXTCOLOR1_DEFAULT	GUI_BLUE	Text color of selected text.
N	LISTWHEEL_TEXTALIGN_DEFAULT	GUI_TA_LEFT	Default text alignment

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_SEL_CHANGED	An item has been snapped at the snap position.

15.14.3 Keyboard reaction

This widget currently does not react on keyboard input.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.14.4 LISTWHEEL API

(Subject to change)
The table below lists the available emWin LISTWHEEL-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
LISTWHEEL_AddString()	Adds a new string.
LISTWHEEL_CreateEx()	Creates a LISTWHEEL widget.
LISTWHEEL_CreateIndirect()	Creates a LISTWHEEL widget from a resource table entry.
LISTWHEEL_GetFont()	Returns the font used to draw the data.
LISTWHEEL_GetItemText()	Returns the text of the requested item.
LISTWHEEL_GetLBorder()	Returns the size in pixels of the left border.
LISTWHEEL_GetLineHeight()	Returns the height used for one item.
LISTWHEEL_GetNumItems()	Returns the number of data items.
LISTWHEEL_GetPos()	Returns the item index of the currently engaged item.
LISTWHEEL_GetRBorder()	Returns the size in pixels of the right border.
LISTWHEEL_GetSel()	Returns the currently selected item.
LISTWHEEL_GetTextAlign()	Returns the text alignment used to draw the data items.
LISTWHEEL_GetUserData()	Returns the custom void pointer of the widget.
LISTWHEEL_MoveToPos()	Moves to the given position.
LISTWHEEL_OwnerDraw()	Default function for drawing the widget.
LISTWHEEL_SetFont()	Sets the font used to draw the item text.
LISTWHEEL_SetItemData()	Assigns a custom void pointer to the given data item.
LISTWHEEL_SetLBorder()	Sets the size in pixels of the left border.
LISTWHEEL_SetLineHeight()	Sets the height used for drawing one data item.
LISTWHEEL_SetOwnerDraw()	Sets a owner draw function for drawing the widget.
LISTWHEEL_SetRBorder()	Sets the size in pixels of the right border.
LISTWHEEL_SetSel()	Sets the currently selected item.
LISTWHEEL_SetSnapPosition()	Sets the snap position in pixels from the top of the widget.
LISTWHEEL_SetText()	Sets the contents of the widget.
LISTWHEEL_SetTextAlign()	Sets the alignment used to draw the data items.
LISTWHEEL_SetTextColor()	Sets the color used to draw the data items.
LISTWHEEL_SetUserData()	Assigns a custom void pointer to the widget.
LISTWHEEL_SetVelocity()	Starts moving the wheel with the given velocity.

LISTWHEEL_AddString()

Description

Adds a new data item (typically a string) to the widget.

Prototype

```
void LISTWHEEL_AddString(LISTWHEEL_Handle hObj, const char * s);
```

Parameter	Meaning
hObj	Handle of the widget.
s	Pointer to the string to be added.

Add. Information

The width of the given text should fit into the horizontal widget area. Otherwise the text will be clipped during the drawing operation.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LISTWHEEL_CreateEx()

Description

Creates a LISTWHEEL widget of a specified size at a specified location.

Prototype

```
LISTWHEEL_Handle LISTWHEEL_CreateEx(int x0, int y0, int xSize, int ySize,
                                    WM_HWIN hParent, int WinFlags,
                                    int ExFlags, int Id,
                                    const GUI_ConstString * ppText);
```

Parameter	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).
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.
ppText	Pointer to an array of string pointers containing the elements to be displayed.

Return value

Handle of the created widget; 0 if the routine fails.

Add. Information

If the parameter ppText is used the last element of the array needs to be a NULL element.

Sample

```
char * apText[] = {
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday",
  "Sunday",
  NULL
};

LISTWHEEL_CreateEx(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW,
                   0, GUI_ID_LISTWHEEL0, apText);
```

LISTWHEEL_CreateIndirect()

Description

Prototype explained at the beginning of the chapter.

LISTWHEEL_GetFont()

Description

Returns the font which is used to draw the data items of the given widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
const GUI_FONT GUI_UNI_PTR * LISTWHEEL_GetFont(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Pointer to a GUI_FONT structure which is used to draw the data items.

LISTWHEEL_GetItemText()

Description

Returns the text of the requested data item.

Prototype

```
void LISTWHEEL_GetItemText(LISTWHEEL_Handle hObj, unsigned Index,
                           char * pBuffer, int MaxSize);
```

Parameter	Meaning
hObj	Handle of the widget.
Index	Index of the requested item.
pBuffer	Buffer for storing the text.
MaxSize	Size in bytes of the buffer.

Add. Information

The function copies the text of the given item into the given buffer. If the size of the buffer is too small the text will be clipped.

LISTWHEEL_GetLBorder()

Description

Returns the size in pixels between the left border of the widget and the beginning of the text.

Prototype

```
int LISTWHEEL_GetLBorder(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Number of pixels between left border and text.

LISTWHEEL_GetLineHeight()

Description

Returns the height of one data item.

Prototype

```
unsigned LISTWHEEL_GetLineHeight(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Height of one data item.

Add. Information

This function returns the value set by the function LISTWHEEL_SetLineHeight(). A return value of zero means the height of one item depends on the size of the current font. For more details please refer to the functions LISTWHEEL_SetLineHeight(), LISTWHEEL_GetFont() and GUI_GetYSizeOfFont().

LISTWHEEL_GetNumItems()

Description

Returns the number of data items of the given widget.

Prototype

```
int LISTWHEEL_GetNumItems(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Number of data items of the given widget.

LISTWHEEL_GetPos()

Description

Returns the zero based index of the item which is currently snapped in.

Prototype

```
int LISTWHEEL_GetPos(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Index of the item which is currently snapped in.

Add. Information

The position at which the items being snapped can be set with the function LISTWHEEL_SetSnapPosition(). For more details please refer to the function LISTWHEEL_SetSnapPosition().

LISTWHEEL_GetRBorder()

Description

Returns the size in pixels between the right border of the widget and the end of the text.

Prototype

```
int LISTWHEEL_GetRBorder(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Number of pixels between right border and text.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LISTWHEEL_GetSel()

Description

Returns the zero based index of the currently selected item.

Prototype

```
int LISTWHEEL_GetSel(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Index of the currently selected item.

Add. Information

For more information please refer to the function LISTWHEEL_SetSel() described later in this chapter.

LISTWHEEL_GetSnapPosition()

Description

Returns the position in pixels from the top of the widget at which the data items should be 'snapped in'.

Prototype

```
int LISTWHEEL_GetSnapPosition(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Snap position in pixels from the top edge of the widget.

Add. Information

The default value is 0.

LISTWHEEL_GetTextAlign()

Description

Returns the text alignment of the given widget.

Prototype

```
int LISTWHEEL_GetTextAlign(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Text alignment of the given widget.

Add. Information

For more information please refer to the function LISTWHEEL_SetTextAlign().

LISTWHEEL_GetUserData()

Description

Returns the user defined data pointer which can be set with LISTWHEEL_SetUserData().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void * LISTWHEEL_GetUserData(LISTWHEEL_Handle hObj);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

User defined data pointer.

Add. Information

The user data pointer can be used in the application and is not used by the widget itself. For more information please refer to the function LISTWHEEL_SetUserData().

LISTWHEEL_MoveToPos()

Description

Lets the data area of the widget move to the position of the given item.

Prototype

```
void LISTWHEEL_MoveToPos(LISTWHEEL_Handle hObj, unsigned int Index);
```

Parameter	Meaning
hObj	Handle of the widget.
Index	Zero based index of the item to which the 'wheel' should move.

Add. Information

The widget starts moving by choosing the shortest way. If for example 7 items are available and item 2 is currently snapped and the widget should move to the last item it beginns moving backwards until the seventh item has been reached.

LISTWHEEL_OwnerDraw()

Description

Default function for managing drawing operations of one data item.

Prototype

```
int LISTWHEEL_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

Parameter	Meaning
hObj	Handle of the widget.

Return value

Depends on the command in the Cmd element of the WIDGET_ITEM_DRAW_INFO structure pointed by pDrawItemInfo.

Add. Information

This function is usefull if LISTWHEEL_SetOwnerDrawn() is used. It can be used to retrieve the original size of a data item and/or to draw the text of a data item and should be called for all commands which are not managed by the application defined owner draw function.
The following commands are managed by the default function:

- WIDGET_ITEM_GET_XSIZE
- WIDGET_ITEM_GET_YSIZE
- WIDGET_ITEM_DRAW

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

For more information please refer to the section explaining user drawn widgets, LISTWHEEL_SetOwnerDraw() and to the provided sample.

LISTWHEEL_SetFont()

Before	After
	

Description

Sets the font which should be used to draw the data items.

Prototype

```
void LISTWHEEL_SetFont(LISTWHEEL_Handle hObj,
                       const GUI_FONT GUI_UNI_PTR * pFont);
```

Parameter	Meaning
hObj	Handle of the widget.
pFont	Pointer to a GUI_FONT structure.

LISTWHEEL_SetLBorder()

Before	After
	

Description

Sets the border size between the left edge of the widget and the beginning of the text.

Prototype

```
void LISTWHEEL_SetLBorder(LISTWHEEL_Handle hObj, unsigned BorderSize);
```

Parameter	Meaning
hObj	Handle of the widget.
BorderSize	Desired border size.

Add. Information

The default value of the border size is 0.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LISTWHEEL_SetLineHeight()

Before	After
	

Description

Sets the line height used to draw a data item.

Prototype

```
void LISTWHEEL_SetLineHeight(LISTWHEEL_Handle hObj, unsigned LineHeight);
```

Parameter	Meaning
hObj	Handle of the widget.
LineHeight	Desired height. Default is 0 which means the font size determines the height of a line.

Add. Information

Per default the height of a line depends on the used font. The value set by this function 'overwrites' this default behavior.

LISTWHEEL_SetOwnerDraw()

Before	After
	

Description

Sets an application defined owner draw function for the widget which is responsible for drawing the widget items.

Prototype

```
void LISTWHEEL_SetOwnerDraw(LISTWHEEL_Handle hObj,
                            WIDGET_DRAW_ITEM_FUNC * pfOwnerDraw);
```

Parameter	Meaning
hObj	Handle of the widget.
pfOwnerDraw	Pointer to owner draw function.

Add. Information

This function sets a pointer to an application defined function which will be called by the widget when a data item has to be drawn or when the x or y size of a item is needed. It gives you the possibility to draw anything as data item, not just plain text. `pfDrawItem` is a pointer to an application-defined function of type `WIDGET_DRAW_ITEM_FUNC` which is explained at the beginning of the chapter.
The following commands are supported: WIDGET_ITEM_GET_YSIZE, WIDGET_ITEM_DRAW, WIDGET_DRAW_BACKGROUND and WIDGET_DRAW_OVERLAY.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Sample

The following sample routine draws 2 red indicator lines over the widget:

```
static int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_DRAW_OVERLAY:
    GUI_SetColor(GUI_RED);
    GUI_DrawHLine(40, 0, 99);
    GUI_DrawHLine(59, 0, 99);
    break;
  default:
    return LISTWHEEL_OwnerDraw(pDrawItemInfo);
  }
  return 0;
}
```

LISTWHEEL_SetRBorder()

Before	After
	

Description

Sets the border size between the left edge of the widget and the beginning of the text.

Prototype

```
void LISTWHEEL_SetRBorder(LISTWHEEL_Handle hObj, unsigned BorderSize);
```

Parameter	Meaning
hObj	Handle of the widget.
BorderSize	Desired border size.

Add. Information

The default value of the border size is 0.

LISTWHEEL_SetSel()

Before	After
	

Description

The function sets the selected item.

Prototype

```
void LISTWHEEL_SetSel(LISTWHEEL_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of the widget.
Sel	Zero based index of item to be selected.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. Information

Only one item can be selected. Per default the item with index 0 is selected.

LISTWHEEL_SetSnapPosition()

Before	After
	

Description

The function sets the relative position from the top of the widget at which the items should snap in. Per default the snap position is 0 which means the items are snapped in at the top of the widget.

Prototype

```
void LISTWHEEL_SetSnapPosition(LISTWHEEL_Handle hObj, int SnapPosition);
```

Parameter	Meaning
hObj	Handle of the widget.
SnapPosition	Relative position in pixels from the top of the widget at which the items should be snapped in.

Add. Information

The function LISTWHEEL_GetPos() can be used to get the zero based index of the current item which has been snapped in.

LISTWHEEL_SetText()

Before	After
	

Description

It removes any existing item and adds the given items passed by the function.

Prototype

```
void LISTWHEEL_SetText(LISTWHEEL_Handle hObj,
                       const GUI_ConstString * ppText);
```

Parameter	Meaning
hObj	Handle of the widget.
ppText	Pointer to an array of strings. The last item needs to be a NULL pointer.

Add. Information

Please note that the last element of the pointed by ppText needs to be a NULL pointer.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Sample

The following should show how the function should be used:

```
static char * _apText[] = {
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday",
  "Sunday",
  NULL
};

static void _SetContents(void) {
  LISTWHEEL_SetText(hWin, _apText);
}
```

LISTWHEEL_SetTextAlign()

Before	After
	

Description

Sets the text alignment used to draw the items of the widget.

Prototype

```
void LISTWHEEL_SetTextAlign(LISTWHEEL_Handle hObj, int Align);
```

Parameter	Meaning
hObj	Handle of the widget.
Align	Alignment to be used to draw the items of the widget.

Add. Information

For details about text alignment please refer to the function GUI_SetTextAlign().

LISTWHEEL_SetTextColor()

Before	After
	

Description

Sets the color to be used to draw the text.

Prototype

```
void LISTWHEEL_SetTextColor(LISTWHEEL_Handle hObj,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of the widget.
Index	(see table below)
Color	Color to be used.

Permitted values for parameter Index	
LISTWHEEL_CI_UNSEL	Sets the color of the not selected text.
LISTWHEEL_CI_SEL	Sets the color of the selected text.

LISTWHEEL_SetUserData()

Description

Sets a user defined data pointer which can be used by the application.

Prototype

```
void LISTWHEEL_SetUserData(LISTWHEEL_Handle hObj, void * pData);
```

Parameter	Meaning
hObj	Handle of the widget.
pData	Pointer to be used by the application.

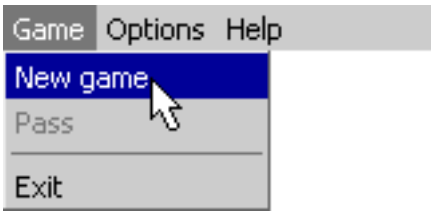
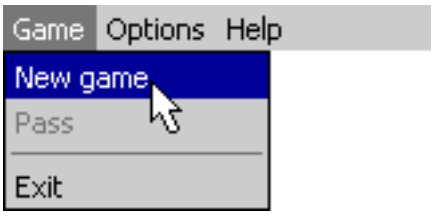
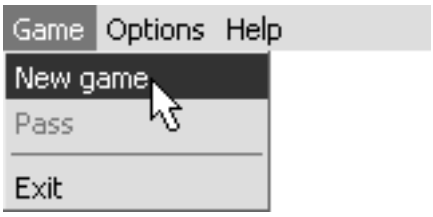
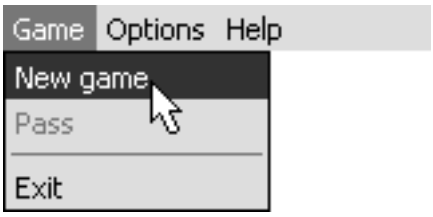
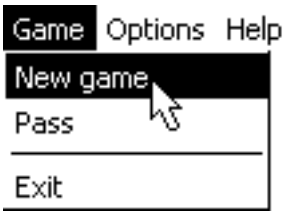
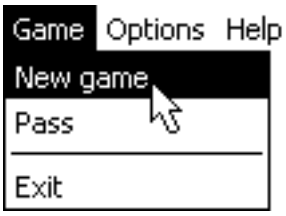
Add. Information

The user data pointer can be used in the application and is not used by the widget itself. For further details please refer to the function LISTWHEEL_GetUserData().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.15 MENU: Menu widget

The MENU widget can be used to create several kinds of menus. Each menu item represents an application command or a submenu. MENUs can be shown horizontally and/or vertically. Menu items can be grouped using separators. Separators are supported for horizontal and vertical menus. Selecting a menu item sends a WM_MENU message to the owner of the menu or opens a submenu. If mouse support is enabled the MENU widget reacts on moving the mouse over the items of a menu.
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.15.1 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_ITEMACTIVATE	The owner window of a menu will receive this message after a menu item has been highlighted. The message is not send after highlighting a sub menu.
MENU_ON_ITEMPRESSED	After pressing a menu item this message will be send to the owner window of the widget. It will be send also for disabled menu items.
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_ITEMACTIVATE:
      _UpdateStatusbar(pData->ItemId);
      break;
    case MENU_ON_INITMENU:
      _OnInitMenu();
      break;
    case MENU_ON_ITEMSELECT:
      switch (pData->ItemId) {
      case ID_MENU_ITEM0:
        ... /* React on selection of menu item 0 */
        break;
      case ID_MENU_ITEM1:
        ... /* React on selection of menu item 1 */
        break;
      case ...
        ...
      }
      break;
    }
    break;
  }
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.15.2 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	hSubmenu	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.15.3 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 sub-menu items.
N	MENU_BORDER_BOTTOM_DEFAULT	2	Border between item text and item bottom.
N	MENU_BORDER_LEFT_DEFAULT	4	Border between item text and left edge of item.
N	MENU_BORDER_RIGHT_DEFAULT	4	Border between item text and right edge of item.
N	MENU_BORDER_TOP_DEFAULT	2	Border between item text and item top.
S	MENU_EFFECT_DEFAULT	WIDGET_Effect_3D1L	Default effect.
S	MENU_FONT_DEFAULT	GUI_Font13_1	Font used.
N	MENU_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color for enabled and 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.

15.15.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	- If the menu is horizontal, the selection moves one item to the right. - If the menu is vertical and the current item is a submenu, the submenu opens and the input focus moves to the submenu. - If the menu is vertical and the current item is not a submenu and the top level menu is horizontal, the next item of the top level menu opens and the input focus moves to it.
GUI_KEY_LEFT	- If the menu is horizontal the selection moves one item to the left. - If the menu is vertical and the menu is not the top level menu, the current menu closes and the focus moves to the previous menu. If the previous menu is horizontal the previous submenu of it opens and the focus moves to the previous submenu.
GUI_KEY_DOWN	- If the menu is horizontal and the current menu item is a submenu this submenu opens. - If the menu is vertical, the selection moves to the next item.
GUI_KEY_UP	- If the menu is vertical, the selection moves to the previous item.
GUI_KEY_ESCAPE	- If the menu is not the top level menu the current menu will be closed and the focus moves to the previous menu. - If the menu is the top level menu, the current menu item becomes unselected.
GUI_KEY_ENTER	- If the current menu item is a submenu, the submenu opens and the focus moves to the submenu. - If the current menu item is not a submenu, all submenus of the top level menu closes and a MENU_ON_ITEMSELECT message will be send to the owner of the menu.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.15.5 MENU API

The table below lists the available emWin MENU-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
MENU_AddItem()	Adds an item to an existing menu.
MENU_Attach()	Attaches a menu with the given size at the given position to a specified window.
MENU_CreateEx()	Creates a menu widget.
MENU_DeleteItem()	Deletes the specified menu item.
MENU_DisableItem()	Disables the specified menu item.
MENU_EnableItem()	Enables the specified menu item.
MENU_GetDefaultBkColor()	Returns the default background color for new menus.
MENU_GetDefaultBorderSize()	Returns the default border size for new menus.
MENU_GetDefaultEffect()	Returns the default effect for new menus.
MENU_GetDefaultFont()	Returns a pointer to the default font used to display the menu item text of new menus.
MENU_GetDefaultTextColor()	Returns the default text color for new menus.
MENU_GetItem()	Retrieves information about the given menu item.
MENU_GetItemText()	Returns the text of the given menu item.
MENU_GetNumItems()	Returns the number of items of the given menu.
MENU_GetOwner()	Returns the owner window of the given menu.
MENU_InsertItem()	Inserts a menu item.
MENU_Popup()	Opens a popup menu at the given position.
MENU_SetBkColor()	Sets the background color of the given menu.
MENU_SetBorderSize()	Sets the border size of the given menu.
MENU_SetDefaultBkColor()	Sets the default background color for new menus.
MENU_SetDefaultBorderSize()	Sets the default border size for new menus.
MENU_SetDefaultEffect()	Sets the default effect for new menus.
MENU_SetDefaultFont()	Sets a pointer to the default font used to display the menu item text of new menus.
MENU_SetDefaultTextColor()	Sets the default text color for new menus.
MENU_SetFont()	Sets the font used to display the menu item text of the given menu.
MENU_SetItem()	Changes the information about the given menu item.
MENU_SetOwner()	Sets the window to be informed by the menu.
MENU_SetTextColor()	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void MENU_AddItem(MENU_Handle hObj, const MENU_ITEM_DATA * pItemData);
```

Parameter	Meaning
hObj	Handle of widget.
pItemData	Poiner 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
hObj	Handle of widget.
hDestWin	Handle to the window to which the menu should be attached.
x	X position in window coordinates of the menu.
y	Y position in window coordinates of the menu.
xSize	Fixed X size of the menu. For details please refer to MENU_CreateEx().
ySize	Fixed Y size of the menu. For details please refer to MENU_CreateEx().
Flags	Reserved for future use

Add. Information

After creating a menu widget this function can be used to attach the menu to an existing window.

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
x0	Leftmost pixel of the widget (in parent coordinates).
y0	Topmost pixel of the widget (in parent coordinates).
xSize	Fixed horizontal size of the widget (in pixels). 0 if menu should handle the xSize.
ySize	Fixed vertical size of the widget (in pixels). 0 if menu should handle the ySize.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Parameter	Meaning
hParent	Handle of parent window. If 0, the new widget will be a child of the desktop (top-level window). In some cases it can be useful to create the menu widget in 'unattached' state and attach it later to an existing window. For this case WM_UNATTACHED can be used as parameter.
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (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	
MENU_CF_HORIZONTAL	Creates a horizontal menu.
MENU_CF_VERTICAL	Creates a vertical menu.
MENU_CF_OPEN_ON_POINTEROVER	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 of 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
hObj	Handle of widget.
ItemId	Id of the menu item to be deleted.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
ColorIndex	Index of color to be returned (see table below)

Permitted values for parameter ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Text color of active submenu items.
MENU_CI_DISABLED	Text color of disabled menu items.
MENU_CI_DISABLED_SEL	Text color of disabled and selected menu items.
MENU_CI_ENABLED	Text color of enabled and not selected menu items.
MENU_CI_SELECTED	Text color of enabled and selected menu items.

Return value

Default text color for new 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
hObj	Handle of widget.
ItemId	Id of the requested menu item.
pItemData	Poiner to a MENU_ITEM_DATA structure to be filled by the function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of widget.
ItemId	Id of the requested menu item.
pBuffer	Buffer to be filled by the function.
BufferSize	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
hObj	Handle of widget.

Return value

Number of items of the given menu.

MENU_GetOwner()

Description

Returns the owner window of the given menu.

Prototype

```
WM_HWIN MENU_GetOwner(MENU_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Return value

Owner window of the given menu.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of widget.
ItemId	Id of the menu item the new item should be inserted before.
pItemData	Poiner 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_Popup()

Description

Opens the given menu at the given position. After selecting a menu item or after touching the display outside the menu the popup menu will be closed.

Prototype

```
void MENU_Popup(MENU_Handle hObj, WM_HWIN hDestWin,
                int x, int y,
                int xSize, int ySize, int Flags);
```

Parameter	Meaning
hObj	Handle of widget.
hDestWin	Handle to the window to which the menu should be attached.
x	X position in window coordinates of the menu.
y	Y position in window coordinates of the menu.
xSize	Fixed X size of the menu. For details please refer to MENU_CreateEx().
ySize	Fixed Y size of the menu. For details please refer to MENU_CreateEx().
Flags	Reserved for future use

Add. Information

After selecting a menu item or after touching the display outside the popup menu the menu will be closed. Please note that the menu will not be deleted automatically. The sample folder contains the sample WIDGET_PopupMenu.c which shows how to use the function.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MENU_SetBkColor()

Before	After
	

Description

Sets the background color of the given menu.

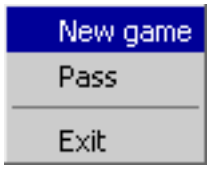
Prototype

```
void MENU_SetBkColor(MENU_Handle hObj, unsigned ColorIndex,
                     GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
ColorIndex	Index of color (see table below)
Color	Color to be used.

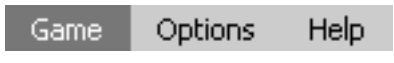
Permitted values for parameter ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Background color of active submenu items.
MENU_CI_DISABLED	Background color of disabled menu items.
MENU_CI_DISABLED_SEL	Background color of disabled and selected menu items.
MENU_CI_ENABLED	Background color of enabled and not selected menu items.
MENU_CI_SELECTED	Background color of enabled and selected menu items.

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);
```


© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 ColorIndex	
MENU_CI_ACTIVE_SUBMENU	Text color of active submenu items.
MENU_CI_DISABLED	Text color of disabled menu items.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter ColorIndex	
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

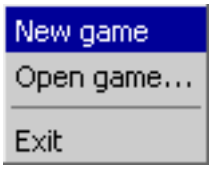
Sets the pointer to the default font used to display the menu item text of new menus.

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
hObj	Handle of widget.
ItemId	Id of the menu item to be changed.
pItemData	Poiner to a MENU_ITEM_DATA structure containing the new information.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of widget.
hOwner	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_SetSel()

Before	After
	

Description

Sets the selected item of the given menu.

Prototype

```
void MENU_SetSel(MENU_Handle hObj, int Sel);
```

Parameter	Meaning
hObj	Handle of widget.
Sel	Zero based index of menu item to be selected.

Return value

The function returns the zero based index of the previous selected menu item.

Add. Information

A value <0 for parameter Sel deselects the menu items.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

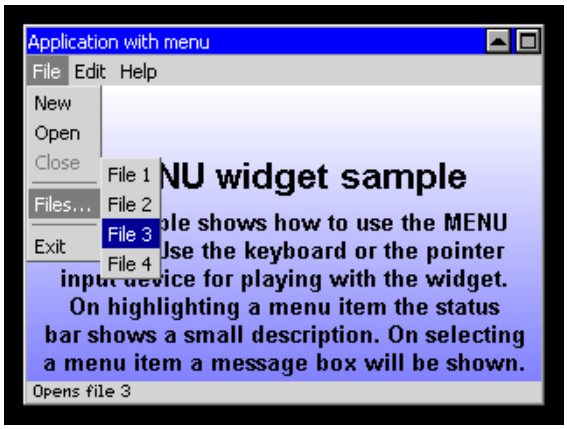
15.15.6 Example

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_Menu.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Menu.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.16 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



15.16.1 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.

15.16.2 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.

15.16.3 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	(see table below)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Flags	
GUI_MESSAGEBOX_CF_MOVEABLE	The message box can be moved by dragging the title bar or the frame.
0	No function.

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.
For details about dragging please refer to the function FRAMEWIN_SetMoveable().

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

The function creates a messagebox consisting of a frame window with the caption text in the title bar, a text widget with the message text and a button widget representing the 'OK' button. After creating the message box the dialog behaviour could be changed by using a user defined callback function or the properties of the box items can be modified using the widget API functions. The following Ids can be used for accessing the items:

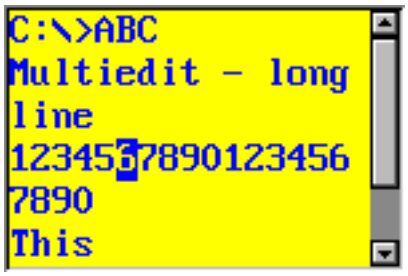
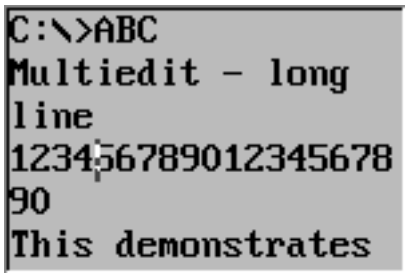
Id	Explanation
GUI_ID_TEXT0	Id of the TEXT widget containing the message text.
GUI_ID_OK	Id of the 'OK' BUTTON widget.

The frame window can be accessed by the handle returned by this function. For more details about this please refer to the sample DIALOG_MessageBoxUser.c.
The function GUI_ExecCreatedDialog() should be used to execute the message box.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.17 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	

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.17.1 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.

15.17.2 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.

15.17.3 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.

15.17.4 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_AddText()	Adds additional text at the current cursor position.
MULTIEDIT_Create()	Creates a multiedit widget. (Obsolete)
MULTIEDIT_CreateEx()	Creates a multiedit widget.
MULTIEDIT_EnableBlink()	Enables/disables a blinking cursor.
MULTIEDIT_GetCursorCharPos()	Returns the number of the character at the cursor position.
MULTIEDIT_GetCursorPixelPos()	Returns the pixel position of the cursor.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
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.
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_SetTextAlign()	Sets the text alignment.
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_AddText()

Description

Adds the given text at the current cursor position.

Prototype

```
int MULTIEDIT_AddText(MULTIEDIT_HANDLE hObj, const char * s);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
s	Pointer to a NULL terminated text to be added.

Add. information

If the number of characters exceeds the limit set with the function MULTIEDIT_SetMaxNumChars() the function will add only the characters of the text which fit into the widget respecting the limit.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.
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 of 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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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	(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 of the created widget; 0 if the routine fails.

MULTIEDIT_CreateIndirect()

Prototype explained at the beginning of the chapter.

MULTIEDIT_EnableBlink()

Description

Enables/disables a blinking cursor.

Prototype

```
void MULTIEDIT_EnableBlink(EDIT_Handle hObj, int Period, int OnOff);
```

Parameter	Meaning
hObj	Handle of edit field.
Period	Blinking period
OnOff	1 enables blinking, 0 disables blinking

Add. Information

This function calls GUI_X_GetTime().

MULTIEDIT_GetCursorCharPos()

Description

Returns the number of the character at the cursor position.

Prototype

```
int MULTIEDIT_GetCursorCharPos(MULTIEDIT_Handle hObj);
```

Parameter	Meaning
hObj	Handle of multiedit widget.

Return value

Number of the character at the cursor position.

Add. Information

The widget returns the character position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MULTIEDIT_GetCursorPixelPos()

Description

Returns the pixel position of the cursor in window coordinates.

Prototype

```
void MULTIEDIT_GetCursorPixelPos(MULTIEDIT_Handle hObj,
                                 int * pxPos, int * pyPos);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
pxPos	Pointer to integer variable for the X-position in window coordinates.
pyPos	Pointer to integer variable for the Y-position in window coordinates.

Add. Information

The widget returns the pixel position if it has the focus or not. This means the cursor position is also returned, if the cursor is currently not visible in the widget.

MULTIEDIT_GetPrompt()

Description

Returns the current prompt text.

Prototype

```
void MULTIEDIT_GetPrompt(MULTIEDIT_HANDLE hObj, char * sDest, int MaxLen);
```

Parameter	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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MULTIEDIT_SetTextAlign()

Description

Sets the text alignment for the given MULTIEDIT widget.

Prototype

```
void MULTIEDIT_SetTextAlign(MULTIEDIT_HANDLE hObj, int Align);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
Align	(see table below)

Permitted values for parameter Align	
GUI_TA_LEFT	Left text align.
GUI_TA_RIGHT	Right text align.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

15.17.5 Example

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_MultiEdit.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

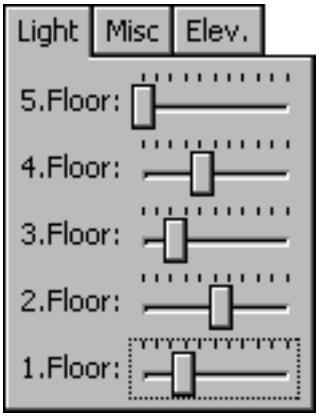
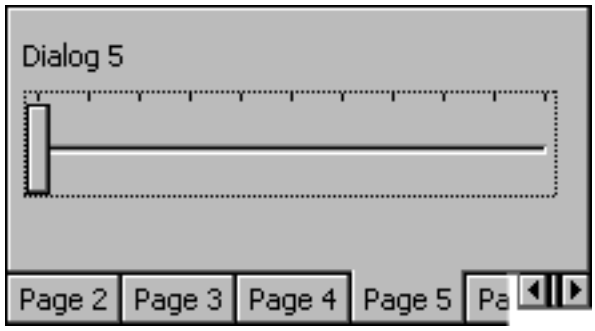
Screenshot of WIDGET_Multiedit.c:



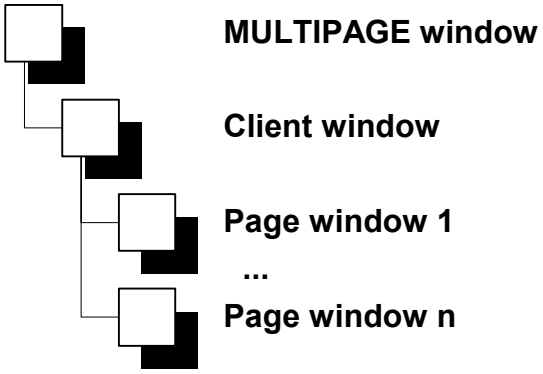
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.18 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.18.1 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.

15.18.2 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.

15.18.3 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_PGUP	Switches to the next page.
GUI_KEY_PGDOWN	Switches to the previous page.

15.18.4 MULTIPAGE API

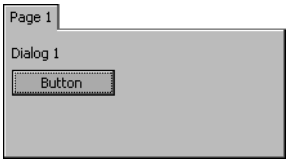
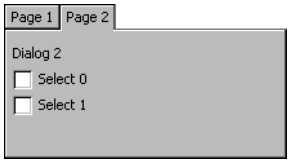
The table below lists the available emWin MULTIPAGE-related routines in alphabetical order. Detailed descriptions of the routines follow.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
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.
MULTIPAGE_SetDefaultAlign()	Sets the default alignment for new MULTIPAGE widgets.
MULTIPAGE_SetDefaultBkColor()	Sets the default background color for new MULTIPAGE widgets.
MULTIPAGE_SetDefaultFont()	Sets the default font used by new MULTIPAGE widgets.
MULTIPAGE_SetDefaultTextColor()	Sets the default text color used by new MULTIPAGE widgets.
MULTIPAGE_SetFont()	Selects the font for the widget.
MULTIPAGE_SetRotation()	Sets the rotation mode for the widget.
MULTIPAGE_SetText()	Sets the text displayed in a tab of a MULTIPAGE widget.
MULTIPAGE_SetTextColor()	Sets the text color.

MULTIPAGE_AddPage()

Before	After
	

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
hObj	Handle of MULTIPAGE widget.
hWin	Handle of window to be shown in the given page.
pText	Pointer to text to be displayed in the tab of the page.

Add. information

It is recomended, 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    int WinFlags, int ExFlags, int Id);
```

Parameter	Meaning
x0	X-position of the widget (in parent coordinates).
y0	Y-position of the widget (in parent coordinates).
xsize	Horizontal size of the widget (in pixels).
ysize	Vertical size of the widget (in pixels).
hParent	Handle of parent window. If 0, the new widget will be a child 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 13: "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 of the new widget.

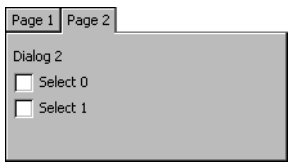
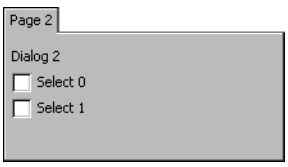
Add. information

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.

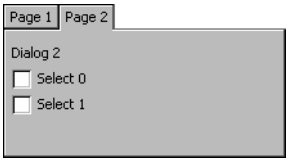
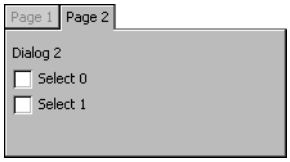
Prototype

```
void MULTIPAGE_DeletePage(MULTIPAGE_Handle hObj,
                          unsigned Index, int Delete);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of the page to be removed from the MULTIPAGE widget.
Delete	If >0 the window attached to the page will be deleted.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MULTIPAGE_DisablePage()

Before	After
	

Description

Disables a page from a MULTIPAGE widget.

Prototype

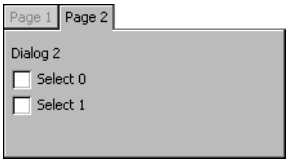
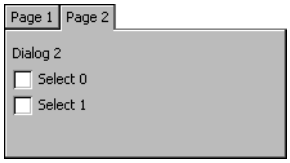
```
void MULTIPAGE_DisablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

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()

Before	After
	

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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

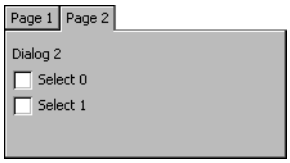
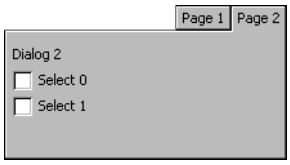
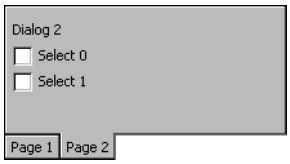
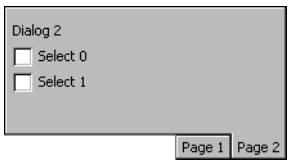
```
unsigned MULTIPAGE_GetDefaultAlign(void);
```

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

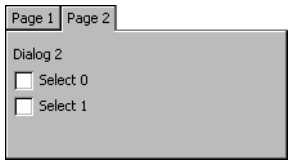
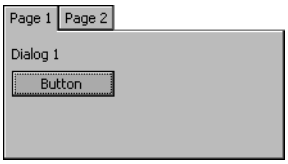
```
int MULTIPAGE_IsPageEnabled (MULTIPAGE_Handle hObj, unsigned Index);
```

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()

Before	After
	

Description

Sets the currently selected page of a MULTIPAGE widget.

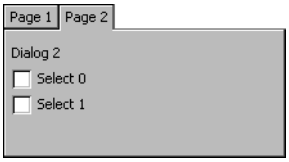
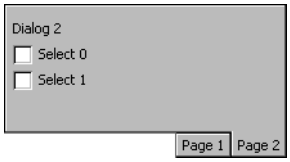
Prototype

```
void MULTIPAGE_SelectPage(MULTIPAGE_Handle hObj, unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Index	Zero based index of page to be selected.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MULTIPAGE_SetAlign()

Before	After
	

Description

Sets the tab alignment for the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetAlign(MULTIPAGE_Handle hObj, unsigned Align);
```

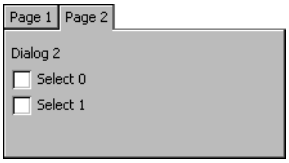
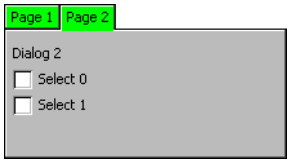
Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Align	See table below.

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()

Before	After
	

Description

Sets the background color of the given MULTIPAGE widget.

Prototype

```
void MULTIPAGE_SetBkColor(MULTIPAGE_Handle hObj,
                          GUI_COLOR Color,
```


© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                unsigned Index);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Color	Color to be used.
Index	See table below.

Permitted values for parameter Index	
MULTIPAGE_CI_DISABLED	Sets the default text color for disabled pages.
MULTIPAGE_CI_ENABLED	Sets the default text color for enabled pages.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

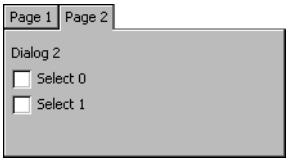
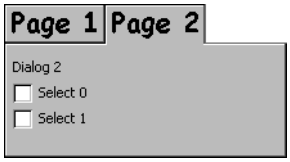
Sets the default text color used to display the text in the tabs of new MULTIPAGE widgets.

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);
```

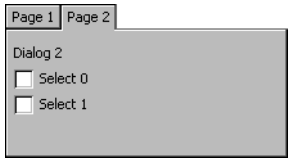
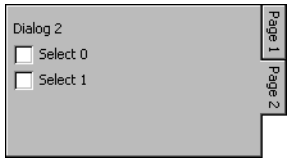
Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
pFont	Pointer to GUI_FONT structure used to display the text in the tabs.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MULTIPAGE_SetRotation()

Before	After
	

Description

Sets the rotation mode of the given widget.

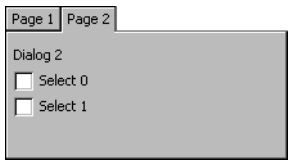
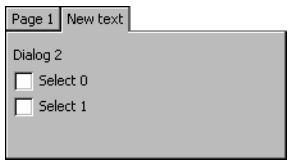
Prototype

```
void MULTIPAGE_SetRotation(MULTIPAGE_Handle hObj, unsigned Rotation);
```

Parameter	Meaning
hObj	Handle of MULTIPAGE widget.
Rotation	Rotation mode (see table below)

Permitted values for parameter Index	
MULTIPAGE_CF_ROTATE_CW	Arranges the tabs at the vertical side and rotates the tab text by 90 degrees clockwise.
0	Default horizontal mode.

MULTIPAGE_SetText()

Before	After
	

Description

Sets the text displayed in the tab of a given page.

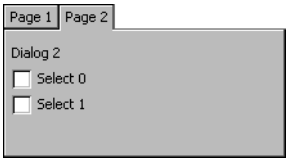
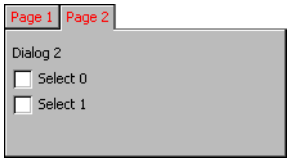
Prototype

```
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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

MULTIPAGE_SetTextColor()

Before	After
	

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.

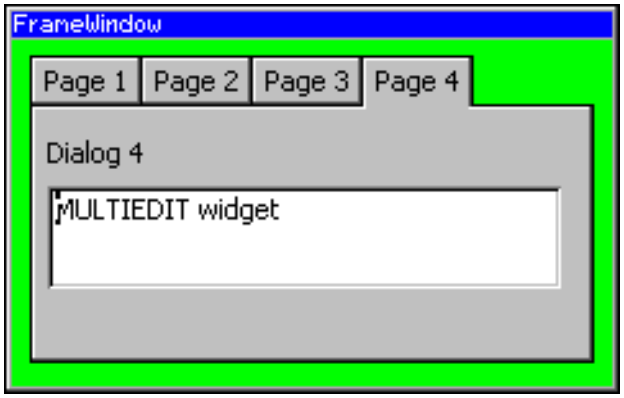
15.18.5 Example

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_Multipage.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_Multipage.c:



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.19 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.

15.19.1 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	0xFFFFFF	Text color, left bar.
N	PROGBAR_DEFAULT_TEXTCOLOR1	0x000000	Text color, right bar.

15.19.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.19.3 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 xsize, 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).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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 of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetTextColor()

Description

Sets the text color of the progress bar.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void PROGBAR_SetTextColor(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 text.
1	Right portion of the text.

PROGBAR_SetTextPos()

Description

Sets the text position in pixels.

Prototype

```
void PROGBAR_SetTextPos(PROGBAR_Handle hObj, int XOff, int YOff);
```

Parameter	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-Min)/(Max-Min)
The default value after creation of the widget is 0.

15.19.4 Examples

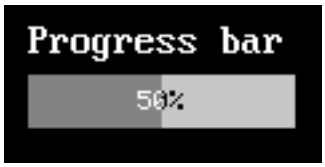
The sample folder contains the following samples which show how the widget can be used:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

- WIDGET_SimpleProgbar:C
- WIDGET_Progbar.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_SimpleProgbar.c:



Screenshot of WIDGET_Progbar.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.20 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		

15.20.1 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.

15.20.2 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 of the button without releasing.
WM_NOTIFICATION_VALUE_CHANGED	Value (selection) of the radio button widget has changed.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.20.3 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.

15.20.4 RADIO API

The table below lists the available emWin RADIO-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
RADIO_Create()	Create a group of radio buttons. (Obsolete)
RADIO_CreateEx()	Create a group of radio buttons.
RADIO_CreateIndirect()	Create a group of radio buttons from resource table entry.
RADIO_Dec()	Decrement the button selection by a value of 1.
RADIO_GetDefaultFont()	Returns the default font used to show the text of new radio buttons.
RADIO_GetDefaultTextColor()	Returns the default text color used to show the text of new radio buttons.
RADIO_GetText()	Returns the text of a radio button item.
RADIO_GetValue()	Return the current button selection.
RADIO_Inc()	Increment the button selection by a value of 1.
RADIO_SetBkColor()	Sets the background color of the radio button.
RADIO_SetDefaultFocusColor()	Sets the default focus rectangle color for new radio buttons.
RADIO_SetDefaultFont()	Sets the default font used to show the text of new radio buttons.
RADIO_SetDefaultImage()	Sets the images to be used for new radio buttons.
RADIO_SetDefaultTextColor()	Sets the default text color used to show the text of new radio buttons.
RADIO_SetFocusColor()	Sets the color of the focus rectangle.
RADIO_SetFont()	Sets the font used to show the text of the radio button.
RADIO_SetGroupId()	Sets the group Id of the given radio widget.
RADIO_SetImage()	Sets the images used to display the radio button.
RADIO_SetText()	Sets the text
RADIO_SetTextColor()	Sets the text color used to show the text of radio button.
RADIO_SetValue()	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
x0	Leftmost pixel of the radio button widget (in parent coordinates).
y0	Topmost pixel of the radio button widget (in parent coordinates).
xsize	Horizontal size of the radio button widget (in pixels).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Parameter	Meaning
ysize	Vertical size of the radio button widget (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
Para	Number of buttons in the group.

Return value

Handle of 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
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 RADIO widget will be a child of the desktop (top-level window).
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (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.
NumItems	Number of items of the radio widget. (default is 2)
Spacing	Number of vertical pixels used for each item of the radio widget.

Return value

Handle of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 0xFFFFFF 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
pFont	Pointer to GUI_FONT 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
pBitmap	Pointer to the bitmap.
Index	(see table below)

Permitted values for parameter Index	
RADIO_BI_INACTIV	Outer image used to show a disabled radio button.
RADIO_BI_ACTIV	Outer image used to show a enabled radio button.
RADIO_BI_CHECK	Inner image used to mark the selected item.

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_ACTIV. The second image (referenced by RADIO_BI_CHECK) is used to mark the currently selected button.

RADIO_SetDefaultTextColor()

Description

Sets the default text color used to display the optional text next to new radio buttons.

Prototype

```
void RADIO_SetDefaultTextColor (GUI_COLOR TextColor);
```

Parameter	Meaning
TextColor	New color to be used.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
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.

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);
```

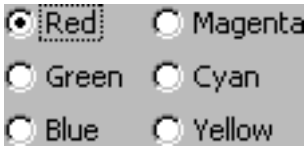
Parameter	Meaning
hObj	Handle of radio button widget.
pFont	Pointer to GUI_FONT structure to be used to display the text.

Add. information

For information about how to add text to a radio widget please refer to the function RADIO_SetText().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

RADIO_SetGroupID()

Before	After
	

Description

Sets the group ID of the radio widget.

Prototype

```
void RADIO_SetGroupID(RADIO_Handle hObj, U8 GroupID);
```

Parameter	Meaning
hObj	Handle of radio button widget.
GroupID	ID of the radio button group. Must be between 1 and 255. If the value is 0 the radio widget is not assigned to a radio button group.

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);
```

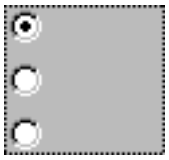
Parameter	Meaning
hObj	Handle of radio button widget.
pBitmap	Pointer to the bitmap.
Index	(see table shown under RADIO_SetDefaultImage)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
hObj	Handle of radio button widget.
pText	Pointer to the text to be shown next to the specified radio button.
Index	Zero based index of the radio button.

Add. information

If using a RADIO widget without text (old style) the focus rectangle is drawn arround the buttons of the widget. If using radio button text the focus rectangle is shown arround 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 besite the radio buttons.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

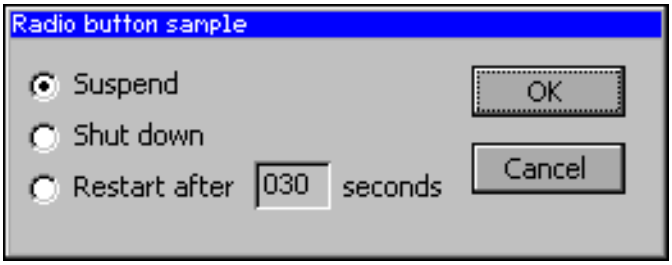
15.20.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- DIALOG_Radio.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of DIALOG_Radio.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.21 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.

15.21.1 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.
N	SCROLLBAR_THUMB_SIZE_MIN_DEFAULT	4	Minimum thumb size.

15.21.2 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.

15.21.3 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_PGDOWN	Increments the current value of the scroll bar by a value of 1 page.
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.
GUI_KEY_PGUP	Decrements the current value of the scroll bar by a value of 1 page.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.21.4 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.
SCROLLBAR_CreateIndirect()	Creates the scroll bar from resource table entry.
SCROLLBAR_Dec()	Decrements the value of the scroll bar by a value of 1.
SCROLLBAR_GetDefaultWidth()	Returns the default width of a scroll bar.
SCROLLBAR_GetValue()	Returns the current item value.
SCROLLBAR_Inc()	Increments the value of the scroll bar by a value of 1.
SCROLLBAR_SetColor()	Sets the color of a scroll bar.
SCROLLBAR_SetDefaultColor()	Sets the default colors for new scroll bars.
SCROLLBAR_SetDefaultWidth()	Sets the default width of a scroll bar.
SCROLLBAR_SetNumItems()	Sets the number of items for scrolling.
SCROLLBAR_SetPageSize()	Sets the page size (in number of items).
SCROLLBAR_SetState()	Sets the state of a scroll bar.
SCROLLBAR_SetValue()	Sets the current value of the scroll bar.
SCROLLBAR_SetWidth()	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
hObj	Handle of scroll bar.
Add	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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    int SpecialFlags);
```

Parameter	Meaning
x0	Leftmost pixel of the scroll bar (in parent coordinates).
y0	Topmost pixel of the scroll bar (in parent coordinates).
xsize	Horizontal size of the scroll bar (in pixels).
ysize	Vertical size of the scroll bar (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
SpecialFlags	Special creation flags (see indirect creation flags under SCROLLBAR_CreateIndirect()).

Return value

Handle of 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
hParent	Handle of parent window.
SpecialFlags	Special creation flags (see indirect creation flags under SCROLLBAR_CreateIndirect()).

Return value

Handle of 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:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG





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 of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()

Before	After
	

Description

Sets the given color attribute of the scroll bar.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

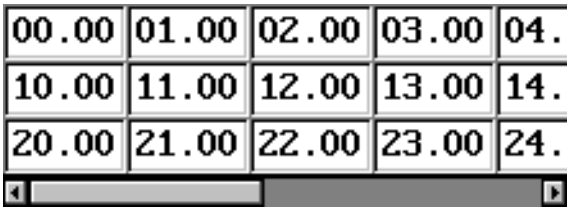
15.21.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_ScrollbarMove.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

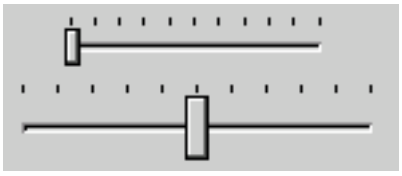
Screenshot of WIDGET_ScrollbarMove.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.22 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.

15.22.1 Configuration options

Type	Macro	Default	Explanation
N	SLIDER_BKCOLOR0_DEFAULT	0xc0c0c0	Background color.
N	SLIDER_COLOR0_DEFAULT	0xc0c0c0	Slider (thumb) color.
N	SLIDER_FOCUSCOLOR_DEFAULT	GUI_BLACK	Default color for rendering the focus rectangle.

15.22.2 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
WM_NOTIFICATION_CLICKED	Slider widget has been clicked.
WM_NOTIFICATION_RELEASED	Slider widget has been released.
WM_NOTIFICATION_VALUE_CHANGED	Value of the slider widget has changed by moving the thumb.

15.22.3 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 slider bar by one item.
GUI_KEY_LEFT	Decrements the current value of the slider bar by one item.

15.22.4 SLIDER API

The table below lists the available emWin SLIDER-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
SLIDER_Create()	Create the slider. (Obsolete)
SLIDER_CreateEx()	Create the slider.
SLIDER_CreateIndirect()	Create the slider from resource table entry.
SLIDER_Dec()	Decrement the value of the slider bar.
SLIDER_GetValue()	Return the current value of the slider bar.
SLIDER_Inc()	Increment the value of the slider bar.
SLIDER_SetBkColor()	Sets the background color of the slider bar.
SLIDER_SetDefaultFocusColor()	Sets the default focus rectangle color for new slider bars.
SLIDER_SetFocusColor()	Sets the color of the focus rectangle.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Routine	Explanation
SLIDER_SetNumTicks()	Sets the number of tick marks of the slider bar.
SLIDER_SetRange()	Set the range of the slider value.
SLIDER_SetValue()	Set the current value of the slider bar.
SLIDER_SetWidth()	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
x0	Leftmost pixel of the slider (in parent coordinates).
y0	Topmost pixel of the slider (in parent coordinates).
xsize	Horizontal size of the slider (in pixels).
ysize	Vertical size of the slider (in pixels).
hParent	Handle of the parent window.
Id	Id to be returned
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 14: "The Window Manager" for a list of available parameter values).
SpecialFlags	Special creation flag (see indirect creation flag under SLIDER_CreateIndirect()).

Return value

Handle of 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
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 SLIDER widget will be a child of the desktop (top-level window).

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 0xFFFFFF 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().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

SLIDER_SetFocusColor()

Before	After
	

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()

Before	After
	

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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

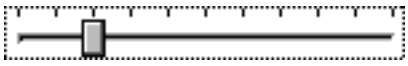
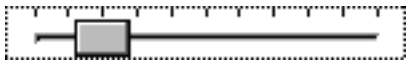
Sets the current value of the slider bar.

Prototype

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

Parameter	Meaning
hObj	Handle of slider widget.
v	Value to be set.

SLIDER_SetWidth()

Before	After
	

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

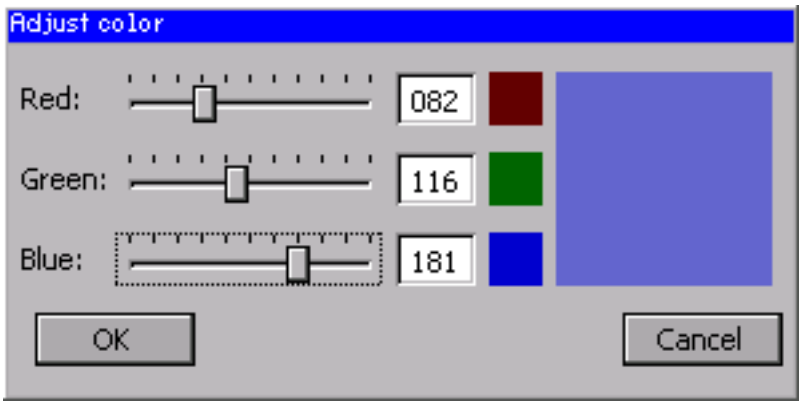
15.22.5 Examples

The sample folder contains the following samples which show how the widget can be used:

- `DIALOG_SliderColor.c`

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of DIALOG_SliderColor.c:



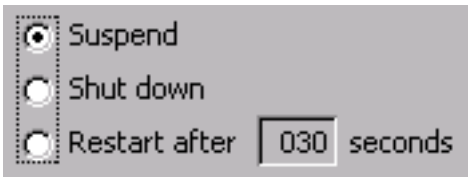
© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.23 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.

15.23.1 Configuration options

Type	Macro	Default	Explanation
N	TEXT_DEFAULT_BK_COLOR	GUI_INVALID_COLOR	Transparent background per default
N	TEXT_DEFAULT_TEXT_COLOR	GUI_BLACK	Default text color.
N	TEXT_DEFAULT_WRAPMODE	GUI_WRAPMODE_NONE	Default wrapping mode.
S	TEXT_FONT_DEFAULT	&GUI_Font13_1	Font used.

15.23.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.23.3 TEXT API

The table below lists the available emWin TEXT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
TEXT_Create()	Creates the text widget. (Obsolete)
TEXT_CreateAsChild()	Creates the text widget as a child window. (Obsolete)
TEXT_CreateEx()	Creates the text widget.
TEXT_CreateIndirect()	Creates the text widget from resource table entry.
TEXT_GetDefaultFont()	Returns the default font used for text.
TEXT_SetBkColor()	Sets the background color for the text.
TEXT_SetDefaultFont()	Sets the default font used for text.
TEXT_SetDefaultTextColor()	Sets the default text color used for text.
TEXT_SetDefaultWrapMode()	Sets the default wrap mode for new text widgets.
TEXT_SetFont()	Sets the font used for a specified text widget.
TEXT_SetText()	Sets the text for a specified text widget.
TEXT_SetTextAlign()	Sets the text alignment of a specified text widget.
TEXT_SetTextColor()	Sets the text color of the given widget.
TEXT_SetWrapMode()	Sets the wrap mode of a specified text widget.

TEXT_Create()

(Obsolete, `TEXT_CreateEx` should be used instead)

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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 of the created TEXT widget; 0 if the routine fails.

TEXT_CreateEx()

Description

Creates a TEXT widget of a specified size at a specified location.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 of 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 0xFFFFFF 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_SetDefaultWrapMode()

Description

Sets the default text wrapping mode used for new text widgets.

Prototype

```
GUI_WRAPMODE TEXT_SetDefaultWrapMode(GUI_WRAPMODE WrapMode);
```

Parameter	Meaning
WrapMode	Default text wrapping mode used for new text widgets.

Return value

Previous default text wrapping mode.

Add. information

For details about text wrapping within the text widget please refer to the function TEXT_SetWrapMode().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

TEXT_SetWrapMode()

Description

Sets the wrapping mode of a specified text widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void TEXT_SetWrapMode(TEXT_Handle hObj, GUI_WRAPMODE WrapMode);
```

Parameter	Meaning
hObj	Handle of text widget.
WrapMode	(see table below)

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

Add. information

For more details about text wrapping please refer to the function GUI_DispStringInRectWrap().

15.23.4 Examples

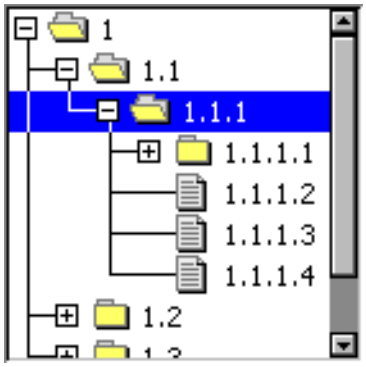
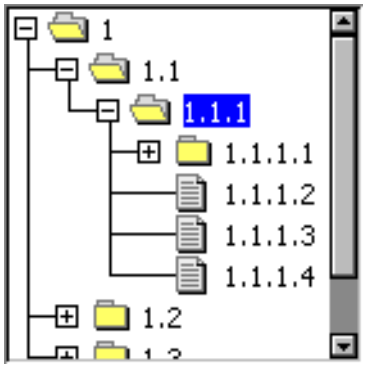
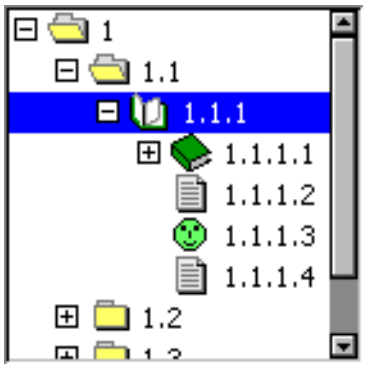
There is no special sample for this widget. Many of the samples use this widget:

- DIALOG_Count.c
- DIALOG_Radio.c
- WIDGET_GraphXY.c
- ...

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.24 TREEVIEW: Treeview widget

A treeview widget can be used to show a hierarchical view of information like files in a directory or items of an index, whereas each item can be a node or a leaf. Each node can have a number of sub items and can be closed or opened.
A node consists of a button image, which shows a plus sign in closed state or a minus sign in open state, two item images (one for closed and one for open state) and the item text. Pressing the button image or double clicking the item image toggles the state (open or closed) of the node.
A leaf consists of an item image and the item text.
The current selection can be marked by highlighting the item text or by highlighting the whole row. All items of a tree are joined by lines per default.
All TREEVIEW-related routines are located in the file(s) `TREEVIEW*.c`, `TREEVIEW*.h`. All identifiers are prefixed TREEVIEW. The table below shows the appearances of the TREEVIEW widget:

Explanation	TREEVIEW widget
Treeview widget with row selection enabled.	
Treeview widget with text selection enabled.	
Treeview widget with some application defined bitmaps and lines off.	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.24.1 Explanation of terms

Item

This means a treeview item which can be a leaf or a node.

Leaf

A leaf is a treeview item which is not able to have any children. It is represented by the leaf bitmap and the item text.

Node

A node is a treeview item which is able to have children. It is represented by the button bitmap, the node bitmap and the item text. The state of the node can be toggled by pressing the button bitmap or by double clicking the node bitmap or the selected area of the item. In open state the children are visible below the node at the next level of indentation.

Button bitmap

This means the bitmap visible at nodes which can be pressed to toggle the state of the node.

Item bitmap

Left beside the item text the item bitmap is shown. Which bitmap is shown depends in the item (leaf or node) and in case of a node it also depends on the state, collapsed or expanded.

Expanded state

In expanded state the children of a node are visible and the minus sign is shown in the button bitmap.

Collapsed state

In collapsed state the children of a node are hidden and the plus sign is shown in the button bitmap.

Joining lines

Lines which are used to connect the items of a tree. The lines connect the button bitmaps of the nodes and the item bitmaps of the leafs according to the hierarchy of the tree.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.24.2 Configuration options

Type	Macro	Default	Explanation
	TREEVIEW_FONT_DEFAULT	&GUI_Font13_1	Default font used to draw the text.
	TREEVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	Background color for unselected state
	TREEVIEW_BKCOLOR1_DEFAULT	GUI_BLUE	Background color for selected state.
	TREEVIEW_BKCOLOR2_DEFAULT	0xC0C0C0	Background color for disabled state.
	TREEVIEW_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color for unselected state.
	TREEVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	Text color for selected state.
	TREEVIEW_TEXTCOLOR2_DEFAULT	GUI_GRAY	Text color for disabled state.
	TREEVIEW_LINECOLOR0_DEFAULT	GUI_BLACK	Line color for unselected state.
	TREEVIEW_LINECOLOR1_DEFAULT	GUI_WHITE	Line color for selected state.
	TREEVIEW_LINECOLOR2_DEFAULT	GUI_GRAY	Line color for disabled state.
	TREEVIEW_IMAGE_CLOSED_DEFAULT		Item image for node in closed state.
	TREEVIEW_IMAGE_OPEN_DEFAULT		Item image for node in open state.
	TREEVIEW_IMAGE_LEAF_DEFAULT		Item image for leaf.
	TREEVIEW_IMAGE_PLUS_DEFAULT		Plus sign
	TREEVIEW_IMAGE_MINUS_DEFAULT		Minus sign
	TREEVIEW_INDENT_DEFAULT	16	Number of pixels for indenting.
	TREEVIEW_TEXT_INDENT_DEFAULT	20	Number of pixels for indenting text.

15.24.3 Notification codes

The following events are sent from a treeview widget to its parent window as part of a WM_NOTIFY_PARENT message:

Message	Explanation
WM_NOTIFICATION_CLICKED	Treeview has been clicked.
WM_NOTIFICATION_RELEASED	Treeview has been released.
WM_NOTIFICATION_MOVED_OUT	Treeview has been clicked and pointer has been moved out off of the widget area without releasing.
WM_NOTIFICATION_SEL_CHANGED	Value (selection) of the treeview widget has changed.

15.24.4 Keyboard reaction

The widget reacts to the following keys if it has the input focus:

Key	Reaction
GUI_KEY_RIGHT	If the cursor is at a closed node, the node is opened. If the cursor is at an open node the cursor moves to the first child of the node.
GUI_KEY_DOWN	The cursor moves to the next visible item below the current position.
GUI_KEY_LEFT	If the cursor is at a leaf the cursor moves to the parent node of the item. If the cursor is at an open node, the node will be closed. If the cursor is at a closed node, the cursor moves to the next parent node.
GUI_KEY_UP	The cursor moves to the previous visible item above the current position.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.24.5 TREEVIEW API

The table below lists the available TREEVIEW-related routines of emWin in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
Common routines	
TREEVIEW_AttachItem()	Attaches an already existing item to the given treeview.
TREEVIEW_CreateEx()	Creates a new treeview widget.
TREEVIEW_CreateIndirect()	Creates a new treeview widget from a resource table.
TREEVIEW_DecSel()	Moves the cursor to the previous visible item.
TREEVIEW_GetDefaultBkColor()	Returns the default background color.
TREEVIEW_GetDefaultFont()	Returns the default font used to draw the item text.
TREEVIEW_GetDefaultLineColor()	Returns the default line color.
TREEVIEW_GetDefaultTextColor()	Returns the default text color.
TREEVIEW_GetItem()	Returns the requested item.
TREEVIEW_GetSel()	Returns the current selected item.
TREEVIEW_IncSel()	Moves the cursor to the next visible item.
TREEVIEW_InsertItem()	Inserts the given item at the given position.
TREEVIEW_SetAutoScrollH()	Manages the automatic use of a horizontal scrollbar.
TREEVIEW_SetAutoScrollV()	Manages the automatic use of a vertical scrollbar.
TREEVIEW_SetBkColor()	Sets the background color.
TREEVIEW_SetDefaultBkColor()	Sets the default background color for new treeviews.
TREEVIEW_SetDefaultFont()	Sets the default font for new treeviews.
TREEVIEW_SetDefaultLineColor()	Sets the default line color for new treeviews.
TREEVIEW_SetDefaultTextColor()	Sets the default text color for new treeviews.
TREEVIEW_SetFont()	Sets the font used to draw the item text.
TREEVIEW_SetHasLines()	Manages the visibility of the joining lines.
TREEVIEW_SetImage()	Sets the images used to draw the treeview items.
TREEVIEW_SetLineColor()	Sets the color used to draw the joining lines.
TREEVIEW_SetSel()	Sets the selection of the treeview.
TREEVIEW_SetSelMode()	Manages the highlighting of the current selection.
TREEVIEW_SetTextColor()	Sets the color used to draw the treeview items.
Item related routines	
TREEVIEW_ITEM_Collapse()	Collapses the given node.
TREEVIEW_ITEM_Create()	Creates a new treeview item.
TREEVIEW_ITEM_Delete()	Deletes the given treeview item.
TREEVIEW_ITEM_Detach()	Detaches the given item without deleting it.
TREEVIEW_ITEM_Expand()	Expands the given node.
TREEVIEW_ITEM_GetInfo()	Returns an information structure of the given item.
TREEVIEW_ITEM_GetText()	Returns the item text.
TREEVIEW_ITEM_GetUserData()	Returns the UserData value of the treeview item.
TREEVIEW_ITEM_SetImage()	Sets the images used to draw the individual given item.
TREEVIEW_ITEM_SetText()	Sets the text of the given item.
TREEVIEW_ITEM_SetUserData()	Sets the UserData value of the treeview item.

15.24.5.1 Common functions

TREEVIEW_AttachItem()

Description

Attaches an already existing item to the treeview widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int TREEVIEW_AttachItem(TREEVIEW_Handle hObj,
                        TREEVIEW_ITEM_Handle hItem,
                        TREEVIEW_ITEM_Handle hItemAt, int Position);
```

Parameter	Meaning
hObj	Handle of widget.
hItem	Handle of item to be attached.
hItemAt	Handle of a currently attached item which specifies the position to be used.
Position	(see table below)

Permitted values for parameter Position	
TREEVIEW_INSERT_ABOVE	Attaches the item above the given position at the same indent level as the given position.
TREEVIEW_INSERT_BELOW	Attaches the item below the given position at the same indent level as the given position.
TREEVIEW_INSERT_INDENT	Attaches the item below the given position by indenting it. The given position needs to be a node level.

Return value

0 on success, otherwise 1.

Add. information

The function can be used for attaching a single item as well as for attaching a complete tree. Please note that in case of attaching a tree the root item of the tree needs to be passed as hItem. If attaching the first item to an empty treeview the parameters hItem and Position should be 0.

TREEVIEW_CreateEx()

Description

Creates a new treeview widget.

Prototype

```
TREEVIEW_Handle TREEVIEW_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 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	(see table below)
Id	Window ID of the widget.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter ExFlags	
TREEVIEW_CF_HIDELINES	Joining lines are not displayed.
TREEVIEW_CF_ROWSELECT	Activates row selection mode.
TREEVIEW_CF_AUTOSCROLLBAR_H	Enables the use of an automatic horizontal scrollbar.
TREEVIEW_CF_AUTOSCROLLBAR_V	Enables the use of an automatic vertical scrollbar.

Return value

Handle of the new widget; 0 if the routine fails.

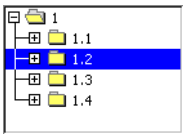
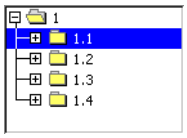
Add. information

The values of parameter ExFlags can be or-combined.

TREEVIEW_CreateIndirect()

Prototype explained at the beginning of the chapter. The Para element of the resource table is not used.

TREEVIEW_DecSel()

Before	After
	

Description

Moves the cursor to the previous visible item of the given treeview.

Prototype

```
void TREEVIEW_DecSel(TREEVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Add. information

If there is no previous visible item the cursor remains on the current position.

TREEVIEW_GetDefaultBkColor()

Description

Returns the default background color used for new treeview widgets.

Prototype

```
GUI_COLOR TREEVIEW_GetDefaultBkColor(int Index);
```

Parameter	Meaning
Index	(see table below)

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Background color of unselected element.
TREEVIEW_CI_SEL	Background color of selected element.
TREEVIEW_CI_DISABLED	Background color of disabled element.

Return value

Default background color used for new treeview widgets.

TREEVIEW_GetDefaultFont()

Description

Returns the default font used to draw the item text of new treeview widgets.

Prototype

```
const GUI_FONT GUI_UNI_PTR * TREEVIEW_GetDefaultFont(void);
```

Return value

Default font used to draw the item text of new treeview widgets.

TREEVIEW_GetDefaultLineColor()

Description

Returns the default color used to draw the joining lines of new treeview widgets.

Prototype

```
GUI_COLOR TREEVIEW_GetDefaultLineColor(int Index);
```

Parameter	Meaning
Index	(see table below)

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Line color of unselected element.
TREEVIEW_CI_SEL	Line color of selected element.
TREEVIEW_CI_DISABLED	Line color of disabled element.

Return value

Default color used to draw the joining lines of new treeview widgets..

TREEVIEW_GetDefaultTextColor()

Description

Returns the default text color used to draw the item text of new treeview widgets.

Prototype

```
GUI_COLOR TREEVIEW_GetDefaultTextColor(int Index);
```

Parameter	Meaning
Index	(see table below)

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Text color of unselected element.
TREEVIEW_CI_SEL	Text color of selected element.
TREEVIEW_CI_DISABLED	Text color of disabled element.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Default text color used to draw the item text of new treeview widgets.

TREEVIEW_GetItem()

Description

Returns the handle of the requested treeview item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_GetItem(TREEVIEW_Handle hObj,
                                      TREEVIEW_ITEM_Handle hItem,
                                      int Flags);
```

Parameter	Meaning
hObj	Handle of widget.
hItem	Handle of treeview item specifying the position to start search from.
Flags	(see table below)

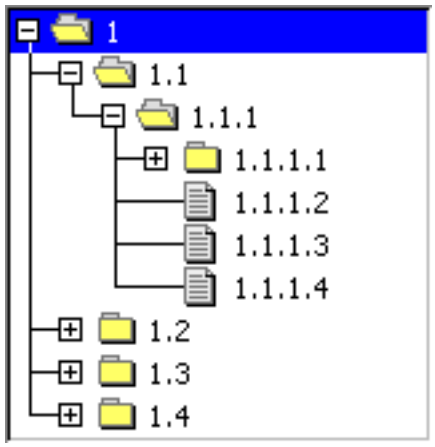
Permitted values for parameter Flags	
TREEVIEW_GET_FIRST	Returns the first item of the treeview widget. Parameter hItem is not required and can be 0.
TREEVIEW_GET_LAST	Returns the last item of the treeview widget. Parameter hItem is not required and can be 0.
TREEVIEW_GET_NEXT_SIBLING	Returns the next child item of the parent node of hItem.
TREEVIEW_GET_PREV_SIBLING	Returns the previous child item of the parent node of hItem.
TREEVIEW_GET_FIRST_CHILD	Returns the first child of the given node.
TREEVIEW_GET_PARENT	Returns the parent node of the given item.

Return value

Handle of the requested treeview item on success, otherwise 0.

Sample

The picture shows a treeview widget with several items. The following shows how parameter Flags can be used for getting treeview items relative to parameter hItem:



- TREEVIEW_GET_NEXT_SIBLING
  The next sibling of '1.1' is '1.2'.
- TREEVIEW_GET_PREV_SIBLING
  The previous sibling of '1.2' is '1.1'.
- TREEVIEW_GET_FIRST_CHILD
  The first child item of '1.1.1' is '1.1.1.1'.
- TREEVIEW_GET_PARENT
  The parent item of '1.1' is '1'.

The use of TREEVIEW_GET_FIRST and TREEVIEW_GET_LAST should be obvious. If the requested item does not exist, the function returns 0.

TREEVIEW_GetSel()

Description

Returns the handle of the currently selected treeview item.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

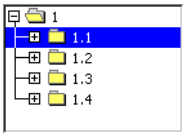
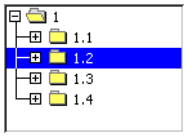
```
TREEVIEW_ITEM_Handle TREEVIEW_GetSel(TREEVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Return value

Handle of the currently selected treeview item. If no item has been selected the return value is 0.

TREEVIEW_IncSel()

Before	After
	

Description

Moves the cursor to the next visible item of the given treeview.

Prototype

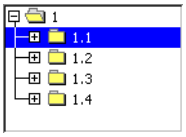
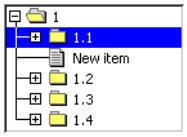
```
void TREEVIEW_IncSel(TREEVIEW_Handle hObj);
```

Parameter	Meaning
hObj	Handle of widget.

Add. information

If there is no next visible item the cursor remains on the current position.

TREEVIEW_InsertItem()

Before	After
	

Description

The function creates and inserts one new treeview item relative to the given item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_InsertItem(TREEVIEW_Handle hObj,
                                         int IsNode,
                                         TREEVIEW_ITEM_Handle hItemPrev,
                                         int Position,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                        const char GUI_UNI_PTR * s);
```

Parameter	Meaning
hObj	Handle of widget.
IsNode	(see table below)
hItemPrev	Handle of treeview item specifying the position of the new item.
Position	(see table below)
s	Text of new treeview item.

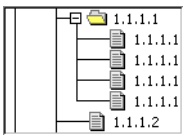
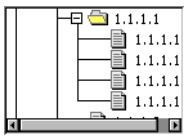
Permitted values for parameter IsNode	
TREEVIEW_ITEM_TYPE_LEAF	New item is a 'leaf'.
TREEVIEW_ITEM_TYPE_NODE	New item is a 'node'.

Permitted values for parameter Position	
TREEVIEW_INSERT_FIRST_CHILD	Should be used for the first item of a treeview node.
TREEVIEW_INSERT_ABOVE	Inserts the item above the given item with the same indent level.
TREEVIEW_INSERT_BELOW	Inserts the item below the given item with the same indent level.

Return value

Handle of the new item on success, otherwise 0.

TREEVIEW_SetAutoScrollH()

Before	After
	

Description

Enables or disables the use of an automatic horizontal scrollbar.

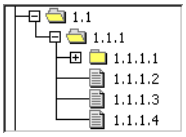
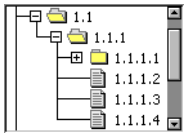
Prototype

```
void TREEVIEW_SetAutoScrollH(TREEVIEW_Handle hObj, int State);
```

Parameter	Meaning
hObj	Handle of widget.
State	1 for enabling an automatic horizontal scrollbar, 0 for disabling.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

TREEVIEW_SetAutoScrollV()

Before	After
	

Description

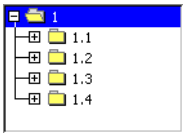
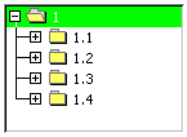
Enables or disables the use of an automatic vertical scrollbar.

Prototype

```
void TREEVIEW_SetAutoScrollV(TREEVIEW_Handle hObj, int State);
```

Parameter	Meaning
hObj	Handle of widget.
State	1 for enabling an automatic vertical scrollbar, 0 for disabling.

TREEVIEW_SetBkColor()

Before	After
	

Description

Sets the background color of the given widget.

Prototype

```
void TREEVIEW_SetBkColor(TREEVIEW_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Index	(see table below)
Color	Color to be used.

Permitted values for parameter Index	
TREEVIEW_CI_UNSEL	Color of unselected item.
TREEVIEW_CI_SEL	Color of selected item.
TREEVIEW_CI_DISABLED	Color of disabled item.

TREEVIEW_SetDefaultBkColor()

Description

Sets the default background color used for new treeview widgets.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void TREEVIEW_SetDefaultBkColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Please refer to the table of TREEVIEW_SetBkColor().
Color	Color to be used.

TREEVIEW_SetDefaultFont()

Description

Sets the default font used for new treeview widgets.

Prototype

```
void TREEVIEW_SetDefaultFont(const GUI_FONT GUI_UNI_PTR * pFont);
```

Parameter	Meaning
pFont	Pointer to GUI_FONT structure to be used.

TREEVIEW_SetDefaultLineColor()

Description

Sets the default line color used for new treeview widgets.

Prototype

```
void TREEVIEW_SetDefaultLineColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	Please refer to the table of TREEVIEW_SetBkColor().
Color	Color to be used.

TREEVIEW_SetDefaultTextColor()

Description

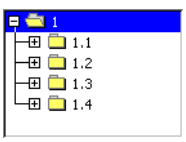
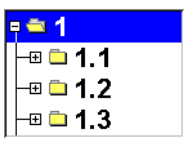
Sets the default text color used for new treeview widgets.

Prototype

```
void TREEVIEW_SetDefaultTextColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
Index	(please refer to the table of TREEVIEW_SetBkColor())
Color	Color to be used.

TREEVIEW_SetFont()

Before	After
	

Description

Sets the font to be used to draw the item text of the given treeview widget.

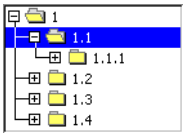
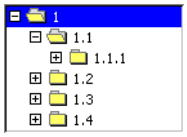
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void TREEVIEW_SetFont(TREEVIEW_Handle hObj,
                      const GUI_FONT GUI_UNI_PTR * pFont);
```

Parameter	Meaning
hObj	Handle of widget.
pFont	Pointer to GUI_FONT structure to be used.

TREEVIEW_SetHasLines()

Before	After
	

Description

Manages the visibility of the joining lines between the treeview items.

Prototype

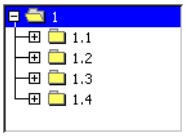
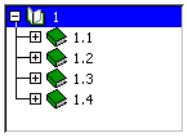
```
void TREEVIEW_SetHasLines(TREEVIEW_Handle hObj, int State);
```

Parameter	Meaning
hObj	Handle of widget.
State	1 for showing the lines, 0 for not showing the lines.

Add. information

Per default the lines are shown.

TREEVIEW_SetImage()

Before	After
	

Description

Sets the images used to draw the treeview items.

Prototype

```
void TREEVIEW_SetImage(TREEVIEW_Handle hObj, int Index,
                       const GUI_BITMAP * pBitmap);
```

Parameter	Meaning
hObj	Handle of widget.
Index	(see table below)
pBitmap	Pointer to bitmap structure to be used.

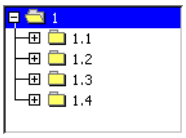
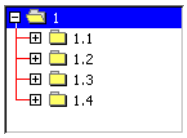
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Index	
TREEVIEW_BI_CLOSED	Image of closed nodes.
TREEVIEW_BI_OPEN	Image of open nodes.
TREEVIEW_BI_LEAF	Image of leaf.
TREEVIEW_BI_PLUS	Plus sign of closed nodes.
TREEVIEW_BI_MINUS	Minus sign of open nodes.

Add. information

The function TREEVIEW_SetItemImage() can be used to set individual images for each item.

TREEVIEW_SetLineColor()

Before	After
	

Description

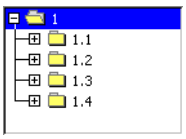
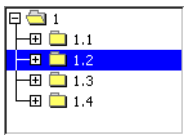
Sets the color used to draw the joining lines between the treeview items.

Prototype

```
void TREEVIEW_SetLineColor(TREEVIEW_Handle hObj, int Index,
                           GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Please refer to the table of TREEVIEW_SetBkColor().
Color	Color to be used.

TREEVIEW_SetSel()

Before	After
	

Description

Sets the currently selected item of the treeview.

Prototype

```
void TREEVIEW_SetSel(TREEVIEW_Handle hObj, TREEVIEW_ITEM_Handle hItem);
```

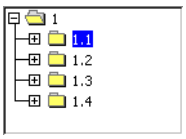
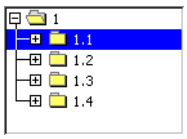
Parameter	Meaning
hObj	Handle of widget.
hItem	Handle of treeview item to be selected.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. information

If the given treeview item is a child of a closed node no selection is visible after calling this function.

TREEVIEW_SetSelMode()

Before	After
	

Description

Sets the selection mode of the treeview widget.

Prototype

```
void TREEVIEW_SetSelMode(TREEVIEW_Handle hObj, int Mode);
```

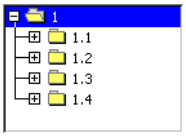
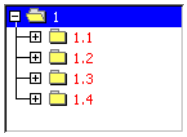
Parameter	Meaning
hObj	Handle of widget.
Mode	(see table below)

Permitted values for parameter Mode	
TREEVIEW_SELMODE_ROW	Activates row selection mode.
TREEVIEW_SELMODE_TEXT	Activates text selection mode.

Add. information

Default selection mode is text selection. If row selection is activated, the complete row can be used to select the item. If text selection is active, only the item text and the item bitmap can be used for selection.

TREEVIEW_SetTextColor()

Before	After
	

Description

Sets the color used to draw the treeview items of the given widget.

Prototype

```
void TREEVIEW_SetTextColor(TREEVIEW_Handle hObj, int Index,
```

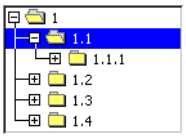
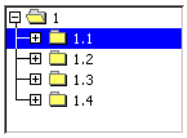
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of widget.
Index	Please refer to the table of TREEVIEW_SetBkColor().
Color	Color to be used.

15.24.5.2Item related routines

TREEVIEW_ITEM_Collapse()

Before	After
	

Description

Collapses the given node and shows the plus sign afterwards.

Prototype

```
void TREEVIEW_ITEM_Collapse(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Meaning
hItem	Handle of item to be collapsed.

Add. information

The given item needs to be a node. Otherwise the function returns immediately.

TREEVIEW_ITEM_Create()

Description

Creates a new treeview item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_CreateItem(int IsNode,
                                         const char GUI_UNI_PTR * s,
                                         U32 UserData);
```

Parameter	Meaning
IsNode	(see table below)
s	Pointer to item text to be shown.
UserData	32 bit value to be used by the application.

Permitted values for parameter IsNode	
TREEVIEW_ITEM_ISNODE	Used to create a node.
TREEVIEW_ITEM_ISLEAF	Used to create a leaf.

Return value

Handle of new item on success, otherwise 0.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. information

After creating a treeview item it contains a copy of the text.

TREEVIEW_ITEM_Delete()

Description

Deletes the given treeview item.

Prototype

```
void TREEVIEW_ITEM_Delete(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Meaning
hItem	Handle of item to be deleted.

Add. information

If the item is currently not attached to any treeview, the parameter hObj should be 0. The function can be used to delete a single item as well as for deleting a complete tree. In case of deleting a tree the root element of the tree should be passed to the function.

TREEVIEW_ITEM_Detach()

Description

Detaches the given treeview item from the treeview widget.

Prototype

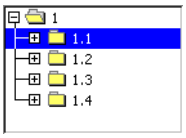
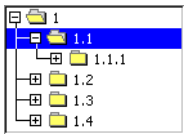
```
void TREEVIEW_ITEM_Detach(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Meaning
hItem	Handle of item to be detached.

Add. information

The function detaches the given item and all of its children from the treeview.

TREEVIEW_ITEM_Expand()

Before	After
	

Description

Expands the given node and shows the minus sign afterwards.

Prototype

```
void TREEVIEW_ITEM_Expand(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Meaning
hItem	Handle of node to be expanded.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. information

The given item needs to be a node. Otherwise the function returns immediately.

TREEVIEW_ITEM_GetInfo()

Description

Returns a structure with information about the given item.

Prototype

```
void TREEVIEW_ITEM_GetInfo(TREEVIEW_ITEM_Handle hItem,
                           TREEVIEW_ITEM_INFO * pInfo);
```

Parameter	Meaning
hItem	Handle of treeview item.
pInfo	Pointer to a TREEVIEW_ITEM_INFO structure to be filled by the function.

Elements of TREEVIEW_ITEM_INFO

Data type	Element	Meaning
int	IsNode	1 if item is a node, 0 if not.
int	IsExpanded	1 if item (node) is open, 0 if closed.
int	HasLines	1 if joining lines are visible, 0 if not.
int	HasRowSelect	1 if row selection is avtive, 0 if not.
int	Level	Indentation level of item.

TREEVIEW_ITEM_GetText()

Description

Returns the item text of the given treeview item.

Prototype

```
void TREEVIEW_ITEM_GetText(TREEVIEW_ITEM_Handle hItem,
                           U8 * pBuffer, int MaxNumBytes);
```

Parameter	Meaning
hItem	Handle of treeview item.
pBuffer	Pointer to buffer filled by the function.
MaxNumBytes	Size of the buffer in bytes.

Add. information

If MaxNumBytes is less than the item text length the buffer is filled with the first MaxNumBytes of the item text.

TREEVIEW_ITEM_GetUserData()

Description

The function return the 32 bit value associated with the given treeview item which can be used by the application program.

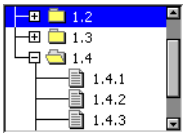
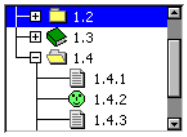
Prototype

```
U32 TREEVIEW_ITEM_GetUserData(TREEVIEW_ITEM_Handle hItem);
```

Parameter	Meaning
hItem	Handle of treeview item.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

TREEVIEW_ITEM_SetImage()

Before	After
	

Description

The function sets images to be used only with the given treeview item.

Prototype

```
void TREEVIEW_ITEM_SetImage(TREEVIEW_ITEM_Handle hItem, int Index,
                            const GUI_BITMAP * pBitmap);
```

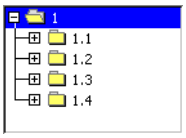
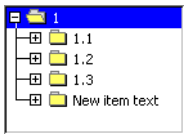
Parameter	Meaning
hItem	Handle if treeview item.
Index	(see table below)
pBitmap	Pointer to bitmap structure to be used.

Permitted values for parameter Index	
TREEVIEW_BI_CLOSED	Image of closed node.
TREEVIEW_BI_OPEN	Image of open node.
TREEVIEW_BI_LEAF	Image of leaf.

Add. information

This function 'overwrites' the default images of the widget. If no individual image is set the default image is used.

TREEVIEW_ITEM_SetText()

Before	After
	

Description

The function sets the text of the given item.

Prototype

```
TREEVIEW_ITEM_Handle TREEVIEW_ITEM_SetText(TREEVIEW_ITEM_Handle hItem,
                                           const char GUI_UNI_PTR * s);
```

Parameter	Meaning
hItem	Handle of treeview item.
s	Pointer to text to be used.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

Handle of the treeview item with the new text.

Add. information

The text will be copied into the treeview item. Please note that using this function changes the handle of the item. After calling this function the new handle needs to be used.

TREEVIEW_ITEM_SetUserData()

Description

The function sets a 32 bit value associated with the given treeview item which can be used by the application program.

Prototype

```
void TREEVIEW_ITEM_SetUserData(TREEVIEW_ITEM_Handle hItem, U32 UserData);
```

Parameter	Meaning
hItem	Handle of treeview item.
UserData	32 bit value to be used by the application program.

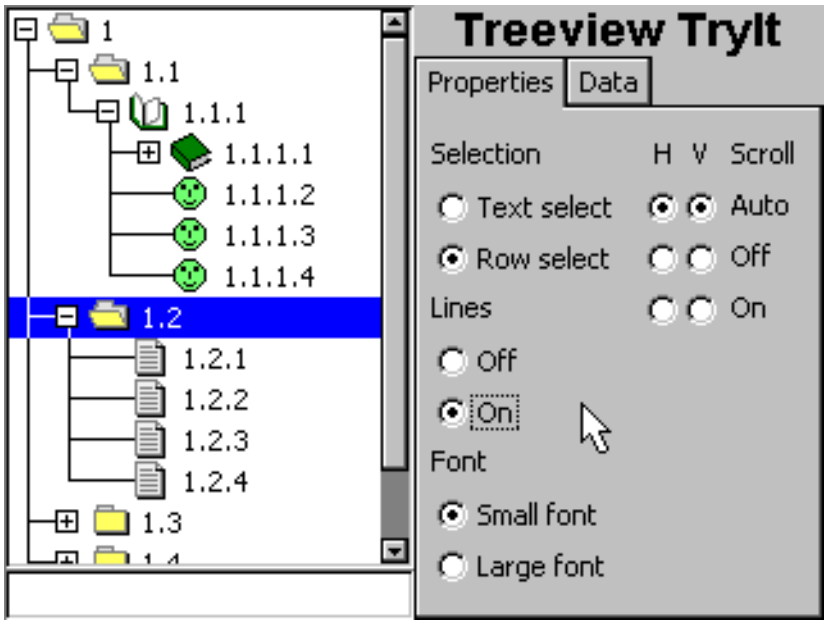
15.24.6 Examples

The sample folder contains the following samples which show how the widget can be used:

- WIDGET_TreeviewTryit.c

Please note that several other samples also make use of this widget and may also be helpful to get familiar with the widget.

Screenshot of WIDGET_TreeviewTryit.c:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

15.25 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.

15.25.1 Configuration options

Type	Macro	Default	Explanation
S	WINDOW_BKCOLOR_DEFAULT	0xC0C0C0	Default background color for new WINDOW widgets

15.25.2 Keyboard reaction

The widget can not gain the input focus and does not react on keyboard input.

15.25.3 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
Color	Color to be used.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.
To move the input focus within a dialog to the next focussable dialog item the key GUI_KEY_TAB can be used. To move backwards GUI_KEY_BACKTAB can be used.

Blocking vs. non-blocking dialogs

Dialog windows can be blocking or non-blocking.
A blocking dialog blocks the thread of execution. It has input focus by default and must be closed by the user before the thread can continue. A blocking dialog does not disable other dialogs shown at the same time. With other words a blocking dialog is not a modal dialog. Blocking means, the used functions (GUI_ExecDialogBox() or GUI_ExecCreatedDialog()) does not return until the dialog is closed.
A non-blocking dialog, on the other hand, does not block the calling thread -- it allows the task to continue while it is visible. The function returns immediately after creating the dialog.

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 sychronization. The events sent by a child depend on its type and are documented seperately 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:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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

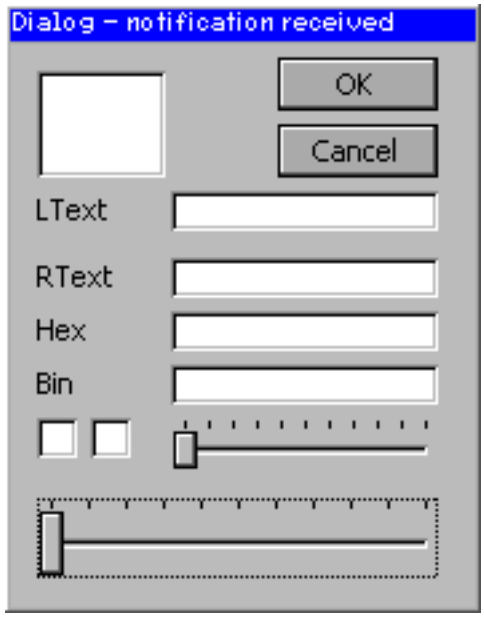
The sample above has been created using the blank dialog procedure shown below. This is the basic template which should be used as a starting point when creating any dialog procedure:

```
/*********************************************************************
*
*       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):



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

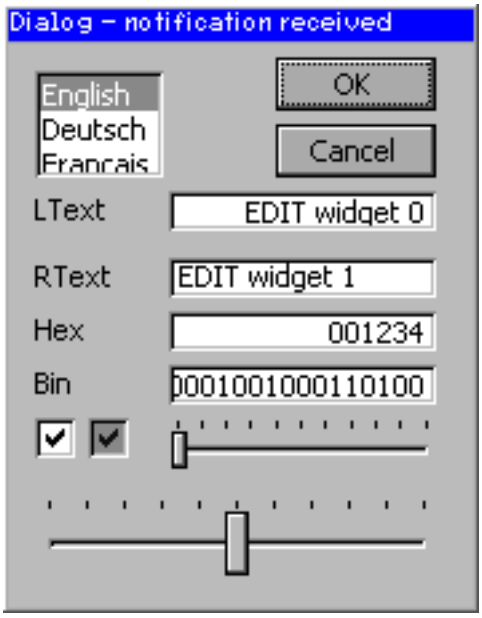
After creation of the dialog box, all widgets included in the resource table will be visible, although as can be seen in the previous screen shot, they will appear "empty". This is because the dialog procedure does not yet contain code that initializes the individual elements. The initial values of the 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:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
paWidget	Pointer to resource table defining the widgets to be included in the dialog.
NumWidgets	Total number of widgets included in the dialog.
cb	Pointer to an application-specific callback function (dialog procedure).
hParent	Handle of parent window (0 = no parent window).
x0	X-position of the dialog relative to parent window.
y0	Y-position of the dialog relative to parent window.

GUI_ExecCreatedDialog()

Description

Executes an already created dialog box.

Prototype

```
int GUI_ExecCreatedDialog(WM_HWIN hDialog);
```

Parameter	Meaning
hDialog	Handle to dialog box.

Return value

Value returned from GUI_EndDialog.

GUI_ExecDialogBox()

Description

Creates and executes a dialog box.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

Add. information

Please note that the handle hDialog is not longer valid after the function has been called. As mentioned above the function ends the dialog which means that it will be deleted from memory. This also applies to all child windows of the given window.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 17

Virtual screen / Virtual pages

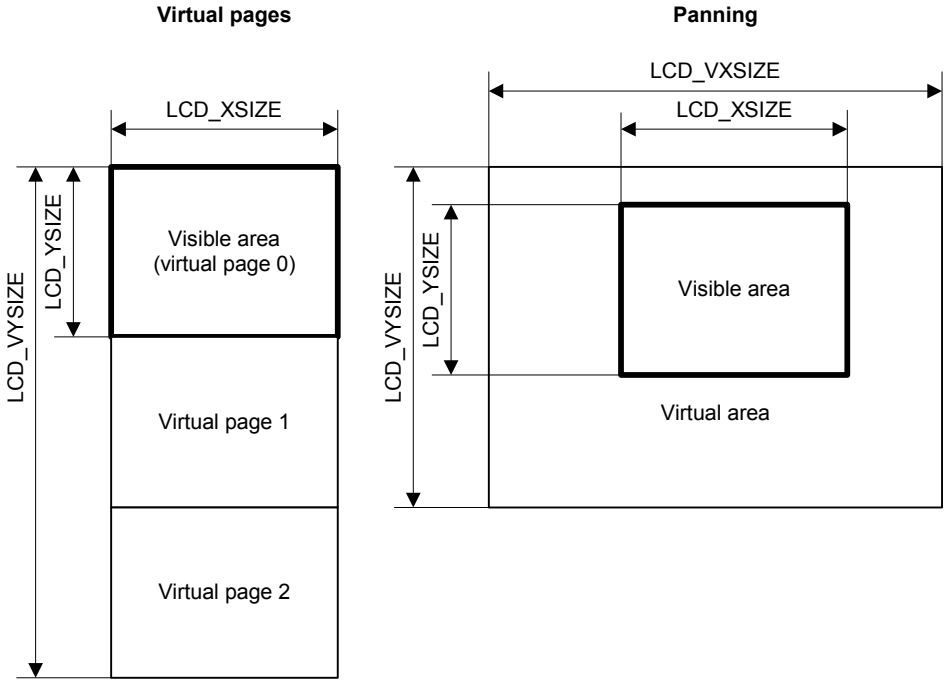
A virtual screen means a display area greater than the physical size of the display. It requires additional video memory and allows instantaneous switching between different screens even on slow CPUs. The following chapter shows

- 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Configurable display start position

The used display controller needs a configurable display start position. This means the display driver even has a register for setting the frame buffer start address or it has a command to set the upper left display start position.

17.3 Configuration

Virtual screens should be configured during the initialization. The function LCD_SetVSizeEx() needs to be used to define the virtual display size. Further it is required to react on the command LCD_X_SETORG in the driver callback routine by setting the right frame buffer start address.

LCD_SetVSizeEx()

Description

Sets the virtual display size.

Prototype

```
int LCD_SetVSizeEx(int LayerIndex, int xSize, int ySize);
```

Parameter	Meaning
LayerIndex	Zero based layer index, typically 0 on single layer systems.
xSize	Horizontal resolution of virtual display.
ySize	Vertical resolution of virtual display.

Return value

0 on success, 1 on error.

```
                                                    display start position */
```

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

```
LCD_SetSizeEx (0, 128,  64);
LCD_SetVSizeEx(0, 128, 192);
```

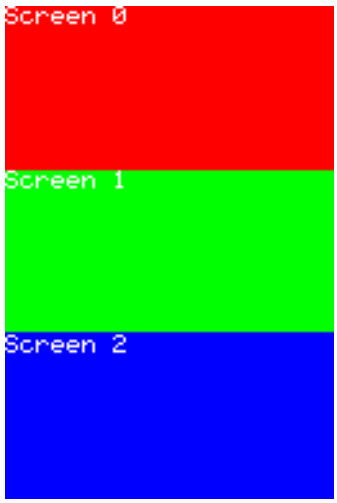
Application

```
  GUI_SetColor(GUI_RED);
  GUI_FillRect(0,   0, 127,  63);
  GUI_SetColor(GUI_GREEN);
  GUI_FillRect(0,  64, 127, 127);
  GUI_SetColor(GUI_BLUE);
  GUI_FillRect(0, 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 */
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Output

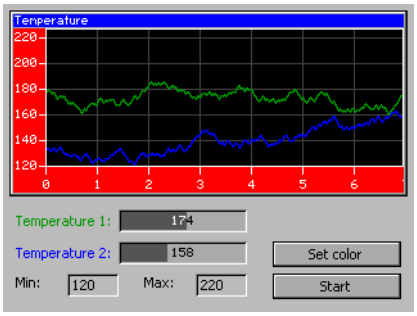
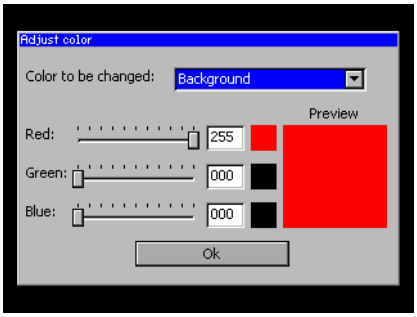
The table below shows the output of the display:

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)`		

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

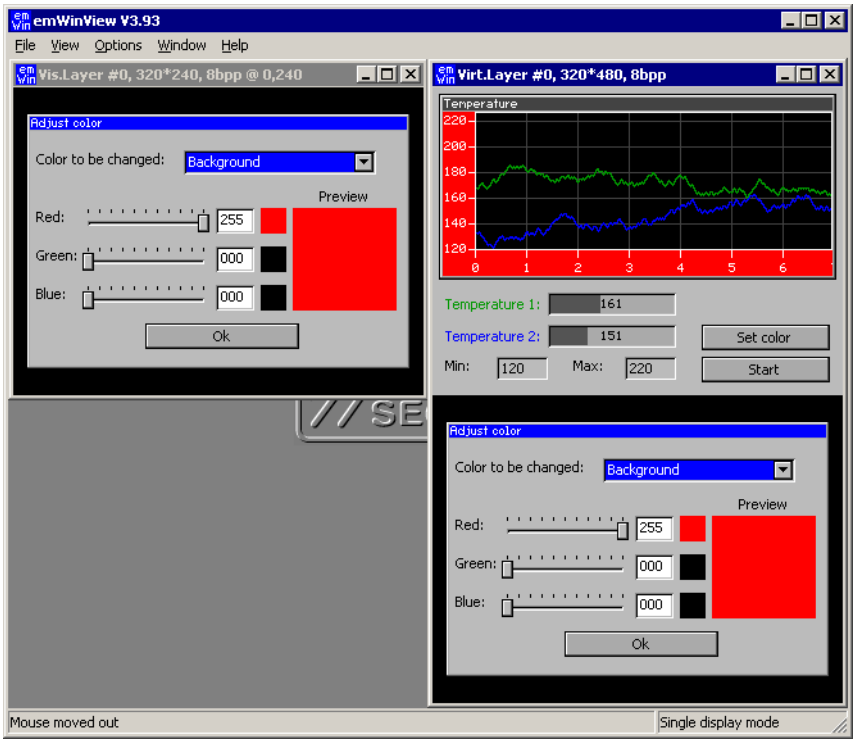
17.4.2 Real time sample using the window manager

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\Tutorial\VSCREEN_RealTime.c`:

Screen 0 / Page 0	Screen 1 / Page 1
	

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

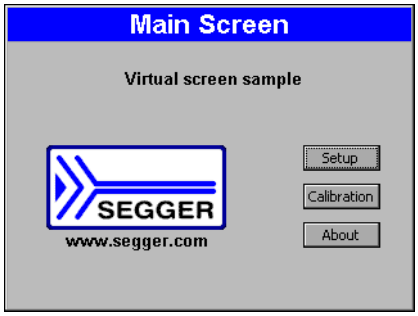
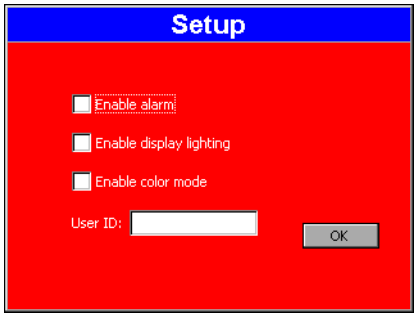
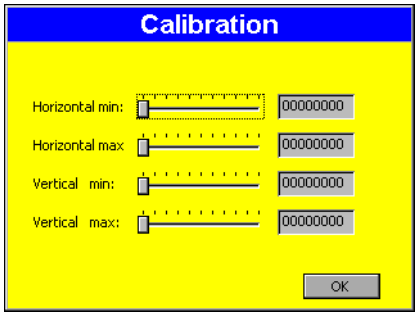
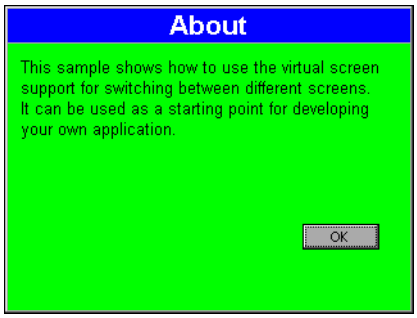


If using the viewer both screens can be shown at the same time. The screenshot above shows the visible display at the left side and the contents of the whole configured virtual display RAM at the right side.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

17.4.3 Dialog sample using the window manager

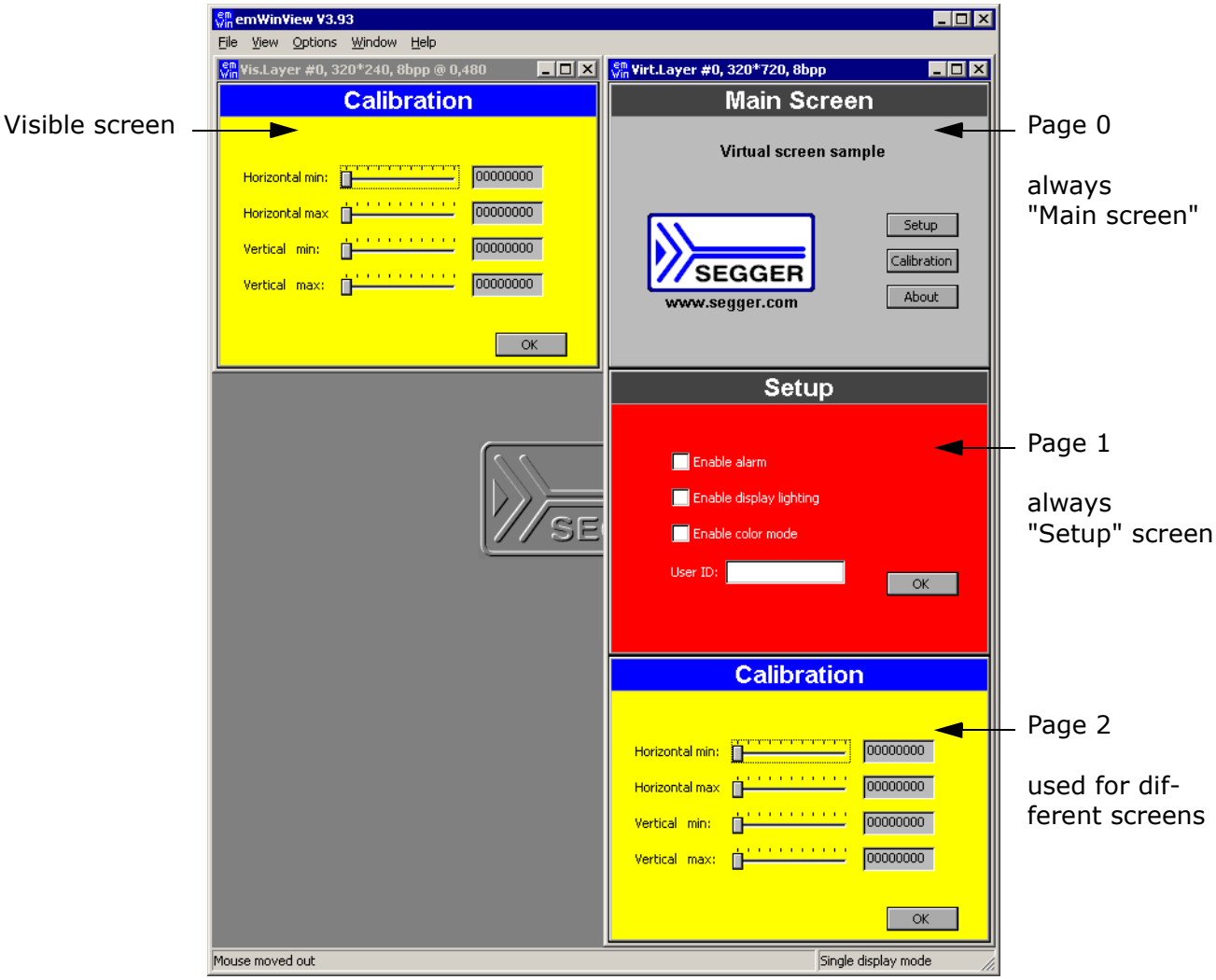
The second advanced sample is available in the folder `Sample\GUI\VSCREEN_MultiPage`. It uses the virtual screen to show 4 screens on 3 different video pages. The application consists of the following screens:

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 18

Multi layer / multi display support

If more than 1 display should be accessed or the display controller supports more than 1 layer (and more than one layer should be used) multi layer support of emWin is required.
Multi layer support and multi display support work the same way. Each layer / display can be accessed with its own color settings, its own size and its own display driver. Initialization of more than one layer is quite simple: The maximum number of available layers `GUI_NUM_LAYERS` should be defined in `GUIConf.h` and each layer needs a display driver device which should be created during the initialization in the configuration routine `LCD_X_Config()`. There is no limitation regarding the maximum number of available layers.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 Selecting a layer for drawing operations

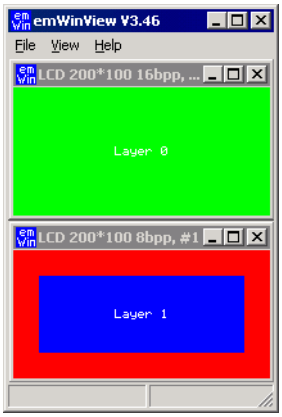
When drawing directly, per default layer 0 is used. Other layers can be selected by using the function `GUI_SelectLayer()`.

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



18.1.2 Selecting a layer for a window

The window manager automatically keeps track of which window is located in which layer. This is done in a fairly easy way:
If the window manager is used, every layer has a top level (desktop) window.
Any other window in this layer is visible only if it is a descendent (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.

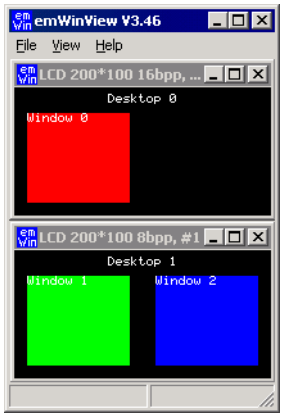
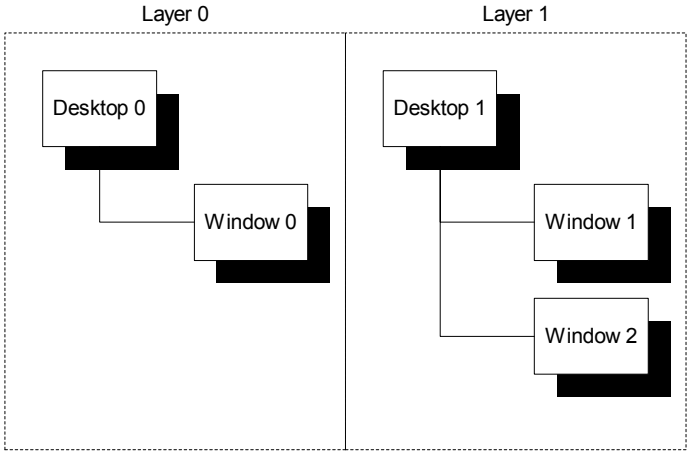
Sample

The following sample shows how to create 3 windows on 2 different desktop windows:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
/* Create 1 child window on destop 0 */
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0);
/* Create 2 child windows on destop 1 */
hWin1 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin1, 0);
hWin2 = WM_CreateWindowAsChild(110, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin2, 0);
```

The following table shows the screenshot and the window hierarchy of the above sample:

Screenshot	Window hierarchy
	

18.1.2.1 Moving a window from one layer to an other

This can sometime be very desirable and can easily be accomplished: If a window is detached from its parent (The desktop window of one layer or any descendent of this desktop window) and attached to a window which lies in an other layer, this window actually moves from one layer to an other layer.

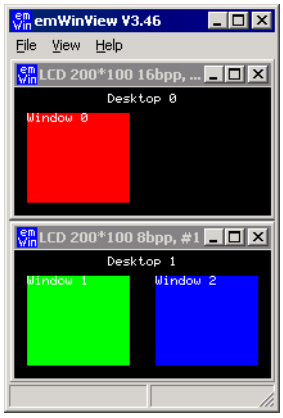
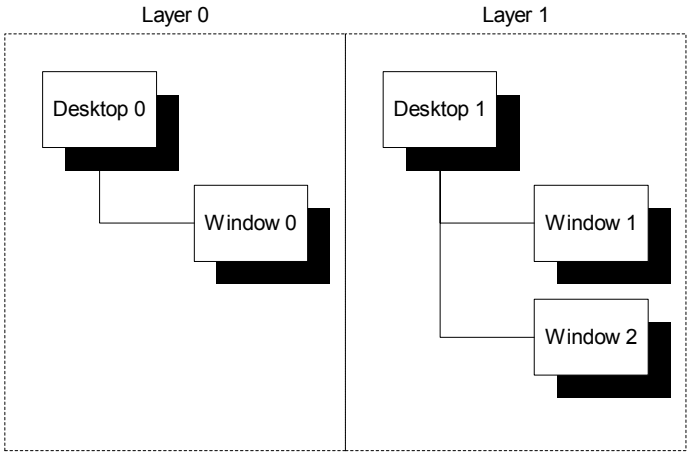
Sample

The following sample shows how to attach a window to a new parent window:

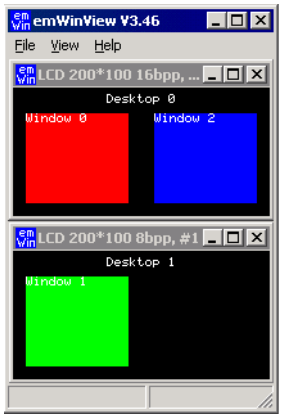
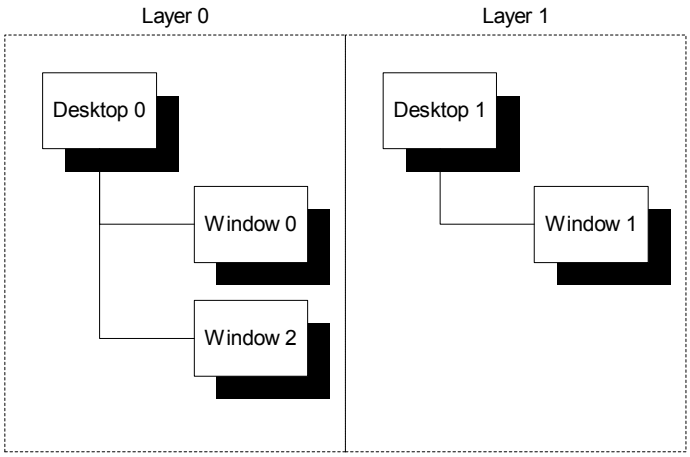
```
/* Create 1 child window on destop 0 */
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0);
/* Create 2 child windows on destop 1 */
hWin1 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin1, 0);
hWin2 = WM_CreateWindowAsChild(110, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin2, 0);
GUI_Delay(1000);
/* Detach window 2 from desktop 1 and attach it to desktop 0 */
WM_AttachWindow(hWin2, WM_GetDesktopWindowEx(0));
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
	

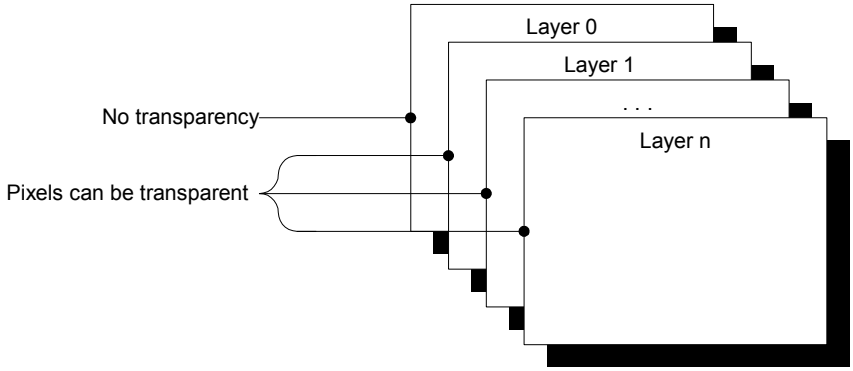
The next table shows the screenshot and the window hierarchy of the above sample after attaching the window to the new parent:

Screenshot	Window hierarchy
	

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 all layers; layers with higher index are on top of layers with lower index and can have transparent pixels:



18.2.1 Transparency

Transparency means that at the position of pixels with color index 0 in a layer > 0, the color of the background layer is visible. Since for all but layer 0 Index 0 means transparency, Index 0 can not be used to display colors. This also means that the color conversion should never yield 0 as best match for a color, since this would result in a transparent pixel. This means that only some fixed palette modes or a custom palette mode should be used and that you need to be careful when defining you own palette. You need to make sure that the color conversion (24 bit RGB -> Index) never yields 0 as result.

Fixed palette modes

86661 is currently the only available fixed palette mode for transparency support. 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:

```
static const LCD_COLOR _aColors_16[] = {
  GUI_TRANSPARENT, 0x000000, 0x222222, 0x333333,
  0x444444, 0x555555, 0x666666, 0x777777,
  0x888888, 0x999999, 0xAAAAAA, 0xBBBBBB,
  0xCCCCCC, 0xDDDDDD, 0xEEEEEE, 0xFFFFFF
};

static const LCD_PHYSPALETTE _aPalette_16 = {
  16, _aColors_16

};
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  .
  .
  .
  //
  // Set user palette data (only required if no fixed palette is used)
  //
  LCD_SetLUTEx(1, _aPalette_16);
}
```

Sample

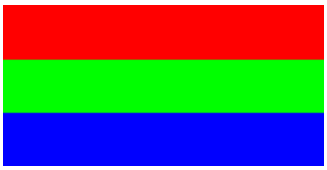
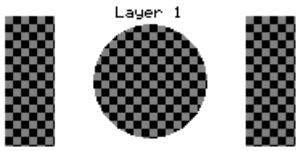
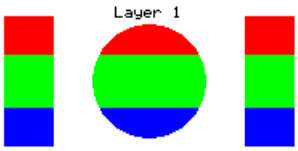
The following sample shows how to use transparency. It draws 3 color bars in layer 0. Layer 1 is filled with white and 3 transparent items are drawn.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
GUI_SelectLayer(0);
GUI_SetColor(GUI_RED);
GUI_FillRect(0, 0, 199, 33);
GUI_SetColor(GUI_GREEN);
GUI_FillRect(0, 34, 199, 66);
GUI_SetColor(GUI_BLUE);
GUI_FillRect(0, 67, 199, 99);
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_WHITE);
GUI_Clear();
GUI_SetColor(GUI_BLACK);
GUI_DispStringHCenterAt("Layer 1", 100, 4);
GUI_SetColor(GUI_TRANSPARENT);
GUI_FillCircle(100, 50, 35);
GUI_FillRect(10, 10, 40, 90);
GUI_FillRect(160, 10, 190, 90);
```

Screenshots of the above sample

The table below shows the contents of the separate layers and the composite view, as the result appears on the display:

Layer 0	Layer 1	Display
		

18.2.2 Alpha blending

Alpha blending is a method of combining two colors allowing for transparency effects. Assumed 2 colors C_0 and C_1 should be combined with alpha blending A (a value between 0 and 1 where 0 means invisible and 1 means 100% visible) the resulting color C_r can be calculated as follows:

$$C_r = C_0 * (1 - A) + C_1 * A$$

Logical colors are handled internally as 32 bit values. The lower 24 bits are used for the color information and the alpha blending is managed in the upper 8 bits. An alpha value of 0x00 means opaque and 0xFF means completely transparent (invisible).

Different methods

There are 3 different methods of managing the alpha information:

- Layer alpha blending: On systems with layer alpha blending the alpha value is fixed to the layer and can be set with the function `LCD_SetAlphaEx()`.
- Lookup table (LUT) alpha blending: This kind of alpha blending uses the LUT for managing the alpha information.
- Pixel alpha blending: Each pixel of the layer to be combined with the background consists of alpha blending information.

Fixed palette modes

For LUT alpha blending the fixed palette modes 822216 and 84444 can be used. Pixel alpha blending is supported only in 32 bpp mode using the fixed palette mode 8888. For details about the fixed palette modes please refer to the chapter 'Colors'.

Sample

The following sample shows how to use pixel alpha blending. It draws a circle in layer 0 and a yellow triangle build of horizontal lines with a vertical gradient of alpha values:

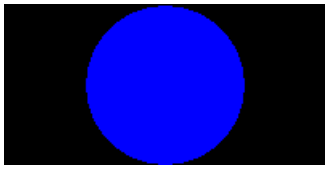
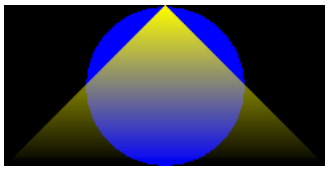
```
GUI_SetColor(GUI_BLUE);
GUI_FillCircle(100, 50, 49);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_TRANSPARENT);
GUI_Clear();
for (i = 0; i < 100; i++) {
  U32 Alpha;
  Alpha = (i * 255 / 100) << 24;
  GUI_SetColor(GUI_YELLOW | Alpha);
  GUI_DrawHLine(i, 100 - i, 100 + i);
}
```

Screenshots of the above sample

The table below shows the contents of the separate layers and the composite view, as the result appears on the display:

Layer 0	Layer 1	Display
		

18.2.3 Hardware cursors

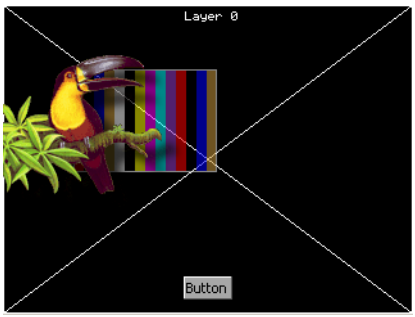
The term 'Hardware cursor' means the use of cursor images in a separate layer with a transparent background. If a hardware supports of multiple layers and the ability of layer positioning emWin can be configured to use a separate layer for managing the cursor. The main advantages of this kind of cursor support are a better performance because only a few registers need to be changed on a movement and the ability of custom drawings in the cursor layer. For details about how to use please refer to the function `GUI_AssignCursorLayer()` described later in this chapter.

18.2.4 Multi layer sample

Chapter 3 "Simulator and viewer" contains a multi layer sample. For details please take a look to this chapter. Further The sample folder contains the following sample which show how to use multiple layer support:

- `MULTILAYER_AlphaChromaMove.c`

Screenshot of above sample



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 6 different displays.

18.3.1 Enabling multi display support

To enable the multi display support you have to define the maximum number of layers in `GUIConf.h`:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
#define GUI_NUM_LAYERS 2 /* Enables support for 2 displays/layers */
```

Further you have to create and configure a display driver device for each layer.

18.3.2 Runtime screen rotation

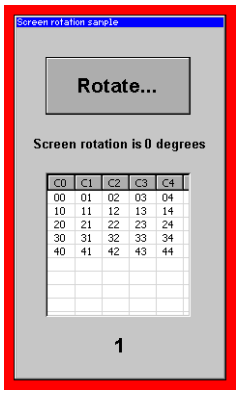
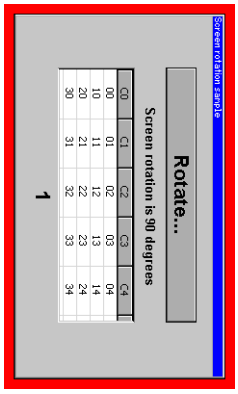
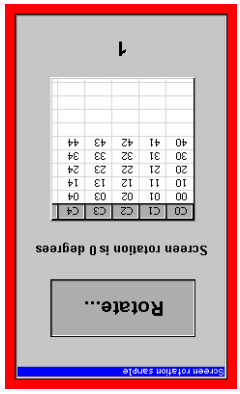
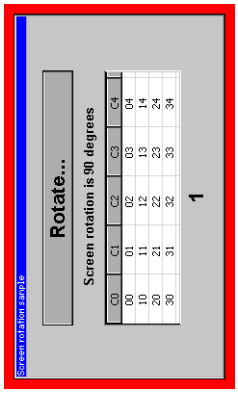
In some cases it may be necessary to change the display orientation at runtime. The multi display support allows to do this. In this case the file `LCDConf.c` should contain a display configuration for each required display orientation. Switching the display orientation then works as follows:

- Select the configuration with the required display orientation with `GUI_SelectLayer()`.
- If the rotation requires a reinitialization of the display controller the right driver function for reinitializing should be called. This is `LCD_L0_Init()` for layer 0 and `LCD_L0_x_Init()` for higher layers, where 'x' means the zero based index of the configuration.

emWin comes with 2 samples including the configuration files. They can be found under `Sample\Tutorial\MULTILAYER_ScreenRotation...`

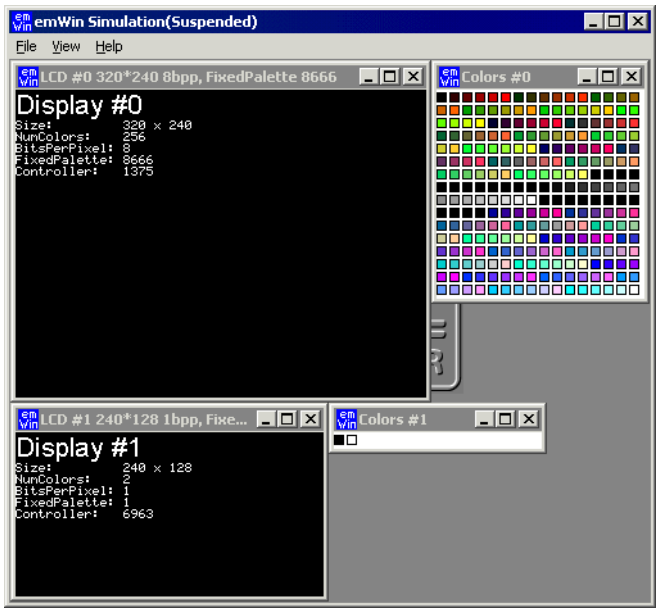
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screenshots of the above sample

Layer 0	Layer 1	Layer 2	Layer 3
			

18.3.3 Multi display sample

The sample below shows a screenshot of the simulation with 2 displays. The fist display is a 8bpp color display with a size of 320 x 240 pixel. The driver is LCD13XX.c configured for an Epson S1D13705 LCD-controller. The second display is a 1bpp bw-display with a size of 240 x 128 pixels. The driver is LCDSlin.c configured for a Toshiba T6963 LCD-controller:



18.4 Configuring multi layer support

LCD Configuration of the above multi layer sample

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for first layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, // Display driver
                           GUICC_655,     // Color conversion
                           0, 0);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (0, 400,  234);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0xc00000); // Video RAM start address
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  //
  // Set display driver and color conversion for second layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,  // Display driver
                           GUICC_86661,   // Color conversion
                           0, 1);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (1, 400,  234);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(1, (void *)0xc00000); // Video RAM start address
}
```

18.5 Configuring multi display support

Configuration of the above multi display sample

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for first layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,  // Display driver
                           GUICC_8666,    // Color conversion
                           0, 0);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (0, 320,  240);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0xc00000); // Video RAM start address
  //
  // Set display driver and color conversion for second layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_1,  // Display driver
                           GUICC_1,       // Color conversion
                           0, 1);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (1, 240,  128);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(1, (void *)0x800000); // Video RAM start address
}
```

18.6 Multi layer API

The table below lists the available multi layer related routines in alphabetical order. Detailed descriptions follow:

Routine	Explanation
GUI_AssignCursorLayer()	Assigns a layer to be used to manage a hardware cursor.
GUI_SelectLayer()	Selects a layer/display for output operations.
GUI_SetLUTColorEx()	Sets a color of a color index in the given layer.
GUI_SetLayerAlphaEx()	Sets the layer alpha blending.
GUI_SetLayerPosEx()	Sets the position of the given layer.
GUI_SetLayerSizeEx()	Sets the size of the given layer.
GUI_SetLayerVisEx()	Sets the visibility of the given layer.
LCD_GetNumLayers()	Returns the number of layers.

GUI_AssignCursorLayer()

Description

The function assigns a layer to be used as cursor layer.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void GUI_AssignCursorLayer(unsigned Index, unsigned CursorLayer);
```

Parameter	Meaning
Index	Layer index.
CursorLayer	Layer to be used to manage the cursor.

Add. Information

Using a hardware cursor means a layer is used as cursor layer. Contrary to the default cursor handling, where the cursor is drawn in the same video memory area as all other items, a hardware cursor is drawn in a separate layer. In this case emWin makes sure the background color of the hardware cursor layer is set to transparency and the selected cursor will be drawn into the layer.
Whereas the default cursor management requires more or less calculation time to draw the cursor and to manage the background, moving a hardware cursor requires only the modification of a few registers.
Please note that using this function requires that the display driver supports layer positioning.

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_SetLayerAlphaEx()

Description

Sets the alpha blending of the given layer.

Prototype

```
int GUI_SetLayerAlphaEx(unsigned Index, int Alpha);
```

Parameter	Meaning
Index	Layer index.
Alpha	Alpha blending value of the given layer.

Add. Information

To be able to use this function the hardware and the used display driver need to support layer alpha blending. If the driver does not support this feature the function returns immediately.
The usable range of alpha values depends on the hardware. In many cases the range of alpha values is limited, for example 0 - 0x3f. emWin does not know something about limitations and passes the given value to the driver. It is the responsibility of the application to make sure that the given value is in a legal range.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

GUI_GetLayerPosEx()

Description

Sets the X- and Y-position of the given layer.

Prototype

```
void GUI_GetLayerPosEx(unsigned Index, GUI_POINT * pPos);
```

Parameter	Meaning
Index	Layer index.
pPos	Pointer to a GUI_POINT structure containing the position to be set.

Add. Information

To be able to use this function the hardware and the used display driver need to support layer positioning. If the driver does not support this feature the function returns immediately.

GUI_SetLayerSizeEx()

Description

Sets the X- and Y-size of the given layer.

Prototype

```
int GUI_SetLayerSizeEx(unsigned Index, int xSize, int ySize);
```

Parameter	Meaning
Index	Layer index.
xSize	New horizontal size in pixels of the given layer.
ySize	New vertical size in pixels of the given layer.

Add. Information

To be able to use this function the hardware and the used display driver need to support layer sizing. If the driver does not support this feature the function returns immediately.

GUI_SetLayerVisEx()

Description

Sets the visibility of the given layer.

Prototype

```
int GUI_SetLayerVisEx(unsigned Index, int OnOff);
```

Parameter	Meaning
Index	Layer index.
OnOff	1 if layer should be visible, 0 for invisible.

Add. Information

To be able to use this function the hardware and the used display driver need to support this feature. If the driver does not support this feature the function returns immediately.

GUI_SetLUTColorEx()

Description

Modifies a single entry in the color table and the LUT of the given layer.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_GetNumLayers()

Description

Returns the number of layers configured in your configuration.

Prototype

```
int LCD_GetNumLayers(void);
```

Return value

Number of layers configured in your configuration.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
GUI_PID_GetState()	Return the current state of the PID.
GUI_PID_StoreState()	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
pState	Pointer to a structure of type GUI_PID_STATE 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
pState	Pointer to a structure of type GUI_PID_STATE.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Add. information

This function can be used from an interrupt service routine.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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) {
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  State.Pressed |= 2;      /* Set bit 1 if right button is pressed */
}
GUI_MOUSE_StoreState(&State);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
GUI_MOUSE_DRIVER_PS2_Init()	Initialise the mouse driver.
GUI_MOUSE_DRIVER_PS2_OnRx()	Called form 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
Data	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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. Your 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
GUI_TOUCH_GetState()	Return the current state of the touch-screen.
GUI_TOUCH_StoreState()	Store the current state of the touch-screen using X- and Y-coordinates.
GUI_TOUCH_StoreStateEx()	Store the current state of the touch-screen.

GUI_TOUCH_GetState()

Description

Returns the current state of the touch-screen.

Prototype

```
int GUI_TOUCH_GetState(GUI_PID_STATE *pState);
```

Parameter	Meaning
pState	Pointer to a structure of type GUI_PID_STATE.

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
x	X-position.
y	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) {
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
    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

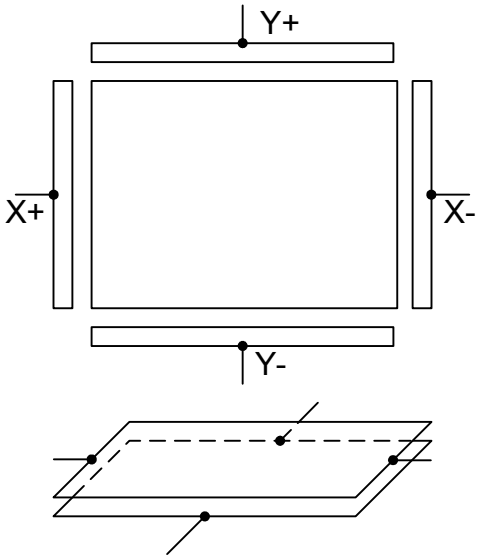
```
GUI_PID_STATE State;
State.x = _TouchPositionX;
State.y = _TouchPositionY;
if (_TouchIsPressed) {
  State.Pressed = 1;
} else {
  State.Pressed = 0;
}
GUI_TOUCH_StoreStateEx(&State);
```

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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(
```

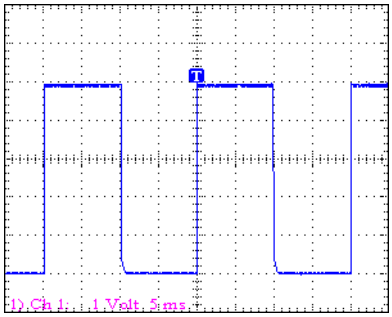
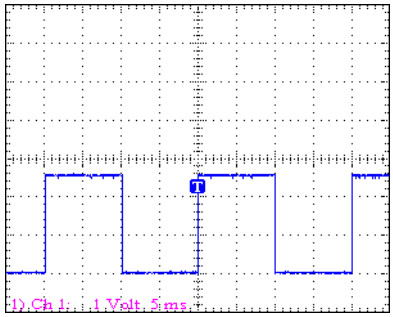
A modul `GUI_TOUCH_X.c` containing the empty routines is located in the folder `Sample\GUI_X`. You can use this module as a starting point.
The activate routines should prepare the measurement by switching on the measurement voltage. `GUI_TOUCH_X_ActivateX()` for example should prepare the measurement in Y by switching on the measurement voltage in X. Further it should switch of the voltage in Y and disable the measurement in X.
The measurement routines should return the measurement result of a A/D converter. Later in this chapter you will find 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
	

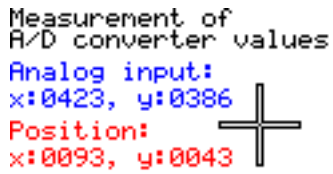
Use sample to determine values for GUITouchConf.h

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
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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\Tutorial 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:



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

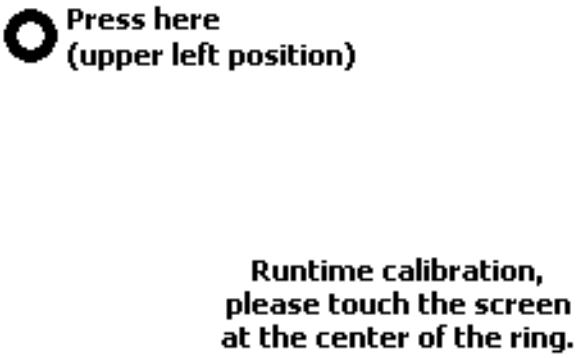
The last step is to modify the touch configuration file GUITouchConf.h. Insert the values into the file like the following sample:

```
/*********************************************************************
*                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:

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG



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
GUI_TOUCH_X_ActivateX()	Prepares measurement for Y-axis.
GUI_TOUCH_X_ActivateY()	Prepares measurement for X-axis.
GUI_TOUCH_X_MeasureX()	Returns the X-result of the A/D converter.
GUI_TOUCH_X_MeasureY()	Returns the Y-result of the A/D converter.

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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_SetOrientationLog()	Sets the logical display orientation.

GUI_TOUCH_Calibrate()

Description

Changes the calibration at run-time.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_TOUCH_Calibrate(int Coord, int Log0, int Log1, int Phys0, int
                        Phys1);
```

Parameter	Meaning
Coord	GUI_COORD_X for X-axis, GUI_COORD_Y for Y-axis.
Log0	Logical value 0 in pixels.
Log1	Logical value 1 in pixels.
Phys0	A/D converter value for Log0.
Phys1	A/D converter value for Log1.

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.

GUI_TOUCH_SetOrientation()

Description

The function configures the touch screen orientation. If the touch screen for example already has been configured to work with the default orientation and the display now needs to be turned or mirrored, this function can be used to configure the touch driver to use the same orientation as the display without changing anything at the hardware routines.

Prototype

```
void GUI_TOUCH_SetOrientation(unsigned Orientation);
```

Parameter	Meaning
Orientation	One or more "OR" comined values of the table below.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Permitted values for parameter Orientation	
GUI_MIRROR_X	Mirroring the X-axis
GUI_MIRROR_Y	Mirroring the Y-axis
GUI_SWAP_XY	Swapping X- and Y-axis

19.4.2.5 Configuring the analog touch-screen driver

The touch screen driver is completely runtime configurable. GUI_TOUCH_Calibrate() should be used to specify the physical values returned by the A/D converter for 2 positions per axis. If the display needs to be turned or mirrored, GUI_TOUCH_SetOrientation() can be used to set a new orientation without changing anything at the hardware routines.
Configuring the touch screen should be done before emWin manages any touch input.

Sample

```
#define TOUCH_AD_LEFT   0x3c0
#define TOUCH_AD_RIGHT  0x034
#define TOUCH_AD_TOP    0x3b0
#define TOUCH_AD_BOTTOM 0x034

Orientation = (GUI_MIRROR_X * LCD_GetMirrorXEx(0)) |
              (GUI_MIRROR_Y * LCD_GetMirrorYEx(0)) |
              (GUI_SWAP_XY  * LCD_GetSwapXYEx (0));
GUI_TOUCH_SetOrientation(Orientation);
GUI_TOUCH_Calibrate(GUI_COORD_X, 0, 239, TOUCH_AD_LEFT, TOUCH_AD_RIGHT);
GUI_TOUCH_Calibrate(GUI_COORD_Y, 0, 319, TOUCH_AD_TOP,  TOUCH_AD_BOTTOM);
```

Compatibility to older versions of emWin

As explained in the chapter 'High-Level Configuration' the macro GUI_SUPPORT_TOUCH can be used for activating a 'compatibility mode' to older versions of emWin. 'Compatibility Mode' at this point means that the preprocessor is used to set the default values located in GUITouchConf.h, which is no longer required.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
  }
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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
GUI_KEY_BACKSPACE	Backspace key.
GUI_KEY_TAB	Tab key.
GUI_KEY_ENTER	Enter/return key.
GUI_KEY_LEFT	Left arrow key.
GUI_KEY_UP	Up arrow key.
GUI_KEY_RIGHT	Right arrow key.
GUI_KEY_DOWN	Down arrow key.
GUI_KEY_HOME	Home key (move to beginning of current line).
GUI_KEY_END	End key (move to end of current line).
GUI_KEY_SHIFT	Shift key.
GUI_KEY_CONTROL	Control key.
GUI_KEY_ESCAPE	Escape key.
GUI_KEY_INSERT	Insert key.
GUI_KEY_DELETE	Delete key.

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
GUI_StoreKeyMsg()	Store a message in a specified key.
GUI_SendKeyMsg()	Send a message to a specified key.

GUI_StoreKeyMsg()

Description

Stores the message data (Key, PressedCnt) into the keybord buffer.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 GUI_StoreKeyMsg()).

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
GUI_ClearKeyBuffer()	Clear the key buffer.
GUI_GetKey()	Return the contents of the key buffer.
GUI_StoreKey()	Store a key in the buffer.
GUI_WaitKey()	Wait for a key to be pressed.

GUI_ClearKeyBuffer()

Description

Clears the key buffer.

Prototype

```
void GUI_ClearKeyBuffer(void);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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

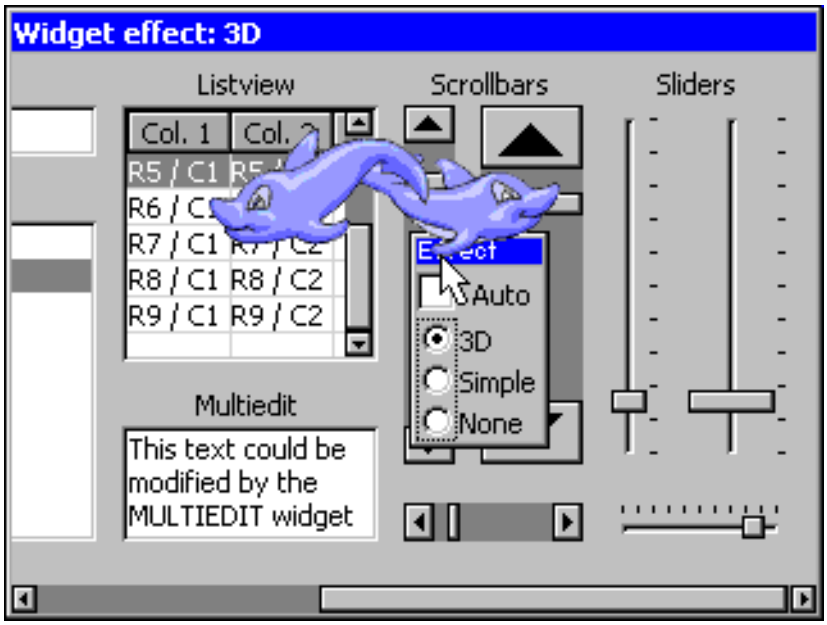
The application is "blocked", meaning it will not return until a key is pressed.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 21

Sprites

A ’sprite’ is an image which can be shown above all other graphics on the screen. A sprite preserves the screen area it covers. It can be moved or removed at any time, fully restoring the screen content. Animation by use of multiple images is possible.
Sprites are completely independent from all other drawing operations as well as window operations: Sprites do not affect drawing or window operations; drawing or window operations do not affect sprites.
Sprites can be seen as objects which are sitting "on top" of the screen, similar to cursors.



© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

21.1 Introducton

emWin sprites are implemented as a pure software solution. No additional hardware is required to use emWin sprites. They can be shown, moved and deleted without effect on the currently visible graphic items.

Memory requirements

Each sprite needs a memory area for saving the display data 'behind' the sprite to be able to restore the background on moving operations or on removing the sprite. Further a memory area for a color cache is required. The size of the color cache depends on the number of colors used in the sprite image. So the complete number of bytes required for a sprite can be calculated as follows:

```
SizeOfSpriteObject (~30 bytes) +
(XSize * YSize + NumberOfBitmapColors) * REQUIRED_BYTES_PER_PIXEL
```

Maximum number of sprites

The number of simultaneous visible sprites is not limited by emWin. It depends only on the available memory.

Performance

Please note that drawing a sprite is more computer-bound than drawing a simple bitmap, because it has to manage the background data and intersections with other sprites.

Z-order

Z-order is an ordering of overlapping two-dimensional objects, in this case the sprites. When two sprites overlap, their Z-order determines which one appears on top of the other. The sprite created at last is the topmost sprite.

21.2 Sprite API

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

Routine	Explanation
GUI_SPRITE_Create()	Creates a sprite.
GUI_SPRITE_CreateEx()	Creates a sprite in the given layer.
GUI_SPRITE_Delete()	Deletes a sprite.
GUI_SPRITE_GetState()	Returrn if the sprite is visible or not.
GUI_SPRITE_Hide()	Hides a sprite.
GUI_SPRITE_SetBitmap()	Sets a new bitmap of a sprite.
GUI_SPRITE_SetBitmapAndPosition()	Sets a new bitmap and the position of a sprite.
GUI_SPRITE_SetPosition()	Sets the position of a sprite.
GUI_SPRITE_Show()	Shows the given sprite.

GUI_SPRITE_Create()

Description

Creates a sprite at the given position in the current layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_Create(const GUI_BITMAP GUI_UNI_PTR * pBM,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    int x, int y);
```

Parameter	Meaning
pBM	Pointer to a bitmap structure to be used for drawing the sprite.
x	X-position of the sprite in screen coordinates.
y	Y-position of the sprite in screen coordinates.

Return value

Handle of the new sprite, 0 on failure.

Add. information

The bitmap addressed by the parameter pBM needs to agree with the following requirements:

- It should not be compressed.
- It needs to be transparent.
- It needs to be a palette based bitmap with 1, 2, 4 or 8bpp.

Other bitmaps or insufficient memory cause the function to fail.

GUI_SPRITE_CreateEx()

Description

Creates a sprite at the given position in the desired layer.

Prototype

```
GUI_HSPRITE GUI_SPRITE_CreateEx(const GUI_BITMAP GUI_UNI_PTR * pBM,
                                int x, int y, int Layer);;
```

Parameter	Meaning
pBM	Pointer to a bitmap structure to be used for drawing the sprite.
x	X-position of the sprite in screen coordinates.
y	Y-position of the sprite in screen coordinates.
Layer	Layer of sprite.

Return value

Handle of the new sprite, 0 on failure.

GUI_SPRITE_Delete()

Description

Deletes the given sprite.

Prototype

```
void GUI_SPRITE_Delete(GUI_HSPRITE hSprite);
```

Parameter	Meaning
hSprite	Handle of sprite to be deleted.

Add. information

The function deletes the sprite from the memory and restores its background automatically.

GUI_SPRITE_GetState()

Description

Returns if the given Sprite is visible or not.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_SPRITE_GetState(GUI_HSPRITE hSprite);
```

Parameter	Meaning
hSprite	Handle of sprite.

Return value

1 if it is visible, 0 if not.

GUI_SPRITE_Hide()

Description

Hides the given sprite.

Prototype

```
void GUI_SPRITE_Hide(GUI_HSPRITE hSprite);
```

Parameter	Meaning
hSprite	Handle of sprite to hide.

Add. information

The function removes the given sprite from the list of visible sprites.

GUI_SPRITE_SetBitmap()

Description

Sets a new image for drawing the sprite.

Prototype

```
int GUI_SPRITE_SetBitmap(GUI_HSPRITE hSprite,
                         const GUI_BITMAP GUI_UNI_PTR * pBM);
```

Parameter	Meaning
hSprite	Handle of sprite.
pBM	Pointer to a bitmap structure to be used for drawing the sprite.

Return value

0 on success, 1 if the routine fails.

Add. information

The new bitmap must have exact the same size as the previous one. Passing a pointer to a bitmap of a different size causes the function to fail.
The function immediately replaces the visible sprite image on the screen. No further operation is required for showing the new image.

GUI_SPRITE_SetBitmapAndPosition()

Description

Sets the position and the image at once.

Prototype

```
int GUI_SPRITE_SetBitmapAndPosition(GUI_HSPRITE hSprite,
                                    const GUI_BITMAP GUI_UNI_PTR * pBM,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    int x, int y);
```

Parameter	Meaning
hSprite	Handle of sprite.
pBM	Pointer to the new bitmap structure to be used to draw the sprite.
x	New X-position in screen coordinates.
y	New Y-position in screen coordinates.

Add. information

It makes a difference on using the functions GUI_SPRITE_SetBitmap() and GUI_SPRITE_SetPosition() one after another or using this function. Whereas the image on the screen will be rendered twice on calling GUI_SPRITE_SetBitmap() and GUI_SPRITE_SetPosition() it is rendered only once on using this function, which can be used very well in animations.

GUI_SPRITE_SetPosition()

Description

Moves the sprite to the new position.

Prototype

```
void GUI_SPRITE_SetPosition(GUI_HSPRITE hSprite, int x, int y);
```

Parameter	Meaning
hSprite	Handle of sprite.
x	New X-position in screen coordinates.
y	New Y-position in screen coordinates.

Add. information

The function moves the given sprite to the new position.

GUI_SPRITE_Show()

Description

Shows the given sprite.

Prototype

```
void GUI_SPRITE_Show(GUI_HSPRITE hSprite);
```

Parameter	Meaning
hSprite	Handle of sprite.

Add. information

The function adds the given sprite to the list of visible sprites.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 22

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

22.1 Available cursors

The following cursor styles are currently available. If a call to GUI_CURSOR_Show() is made and no style is specified with GUI_CURSOR_Select(), the default cursor will be a medium arrow.

Arrow cursors		Cross cursors	
GUI_CursorArrowS Small arrow		GUI_CursorCrossS Small cross	
GUI_CursorArrowM Medium arrow (default cursor)		GUI_CursorCrossM Medium cross	
GUI_CursorArrowL Large arrow		GUI_CursorCrossL Large cross	
Inverted arrow cursors		**Inverted cross cursors**	
GUI_CursorArrowSI Small inverted arrow		GUI_CursorCrossSI Small inverted cross	
GUI_CursorArrowMI Medium inverted arrow		GUI_CursorCrossMI Medium inverted cross	
GUI_CursorArrowLI Large inverted arrow		GUI_CursorCrossLI Large inverted cross	

22.2 Cursor API

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

Routine	Explanation
GUI_CURSOR_GetState()	Returns if the cursor is visible or not.
GUI_CURSOR_Hide()	Hide the cursor.
GUI_CURSOR_Select()	Set a specified cursor.
GUI_CURSOR_SetPosition()	Set the cursor position.
GUI_CURSOR_Show()	Show the cursor.

GUI_CURSOR_GetState()

Description

Returns if the cursor is currently visible or not.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int GUI_CURSOR_GetState(void);
```

Return value

1 if the cursor is visible and 0 if not.

GUI_CURSOR_Hide()

Description

Hides the cursor.

Prototype

```
void GUI_CURSOR_Hide(void);
```

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
pCursor	Pointer to the cursor to be selected.

Permitted values for parameter pCursor	
GUI_CursorArrowS	Small arrow.
GUI_CursorArrowM	Medium arrow.
GUI_CursorArrowL	Large arrow.
GUI_CursorArrowSI	Small inverted arrow.
GUI_CursorArrowMI	Medium inverted arrow.
GUI_CursorArrowLI	Large inverted arrow.
GUI_CursorCrossS	Small cross.
GUI_CursorCrossM	Medium cross.
GUI_CursorCrossL	Large cross.
GUI_CursorCrossSI	Small inverted cross.
GUI_CursorCrossMI	Medium inverted cross.
GUI_CursorCrossLI	Large inverted cross.

Add. information

If this function is not called, the default cursor is a medium arrow.

GUI_CURSOR_SetPosition()

Description

Sets the cursor position.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

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

Parameter	Meaning
x	X-position of the cursor.
y	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 23

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 seperate software item and is not included in the emWin basic package. The software for antialiasing is located in the subdirectory `GUI\AntiAlias`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

23.1 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).

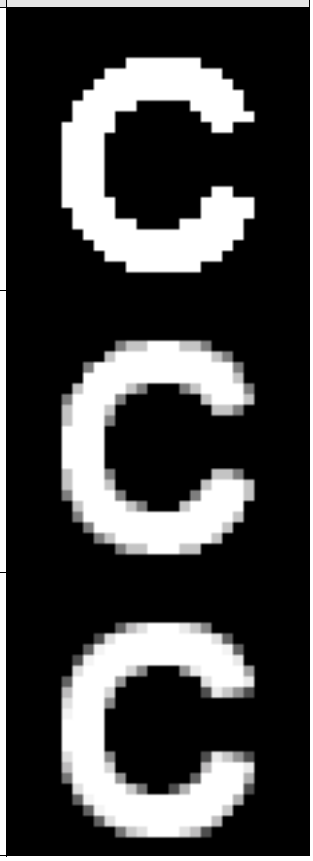
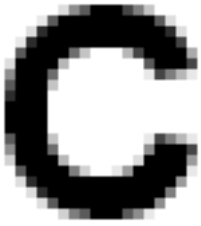
23.1.1 Quality of antialiasing



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 x 2 = 4. The next line is drawn with an antialiasing factor of 3, so there are 3 x 3 = 9 shades, and so on. Factor 4 should be sufficient for most applications. Increasing the antialiasing factor further does not improve the result dramatically, but increases the calculation time.

23.1.2 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		

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

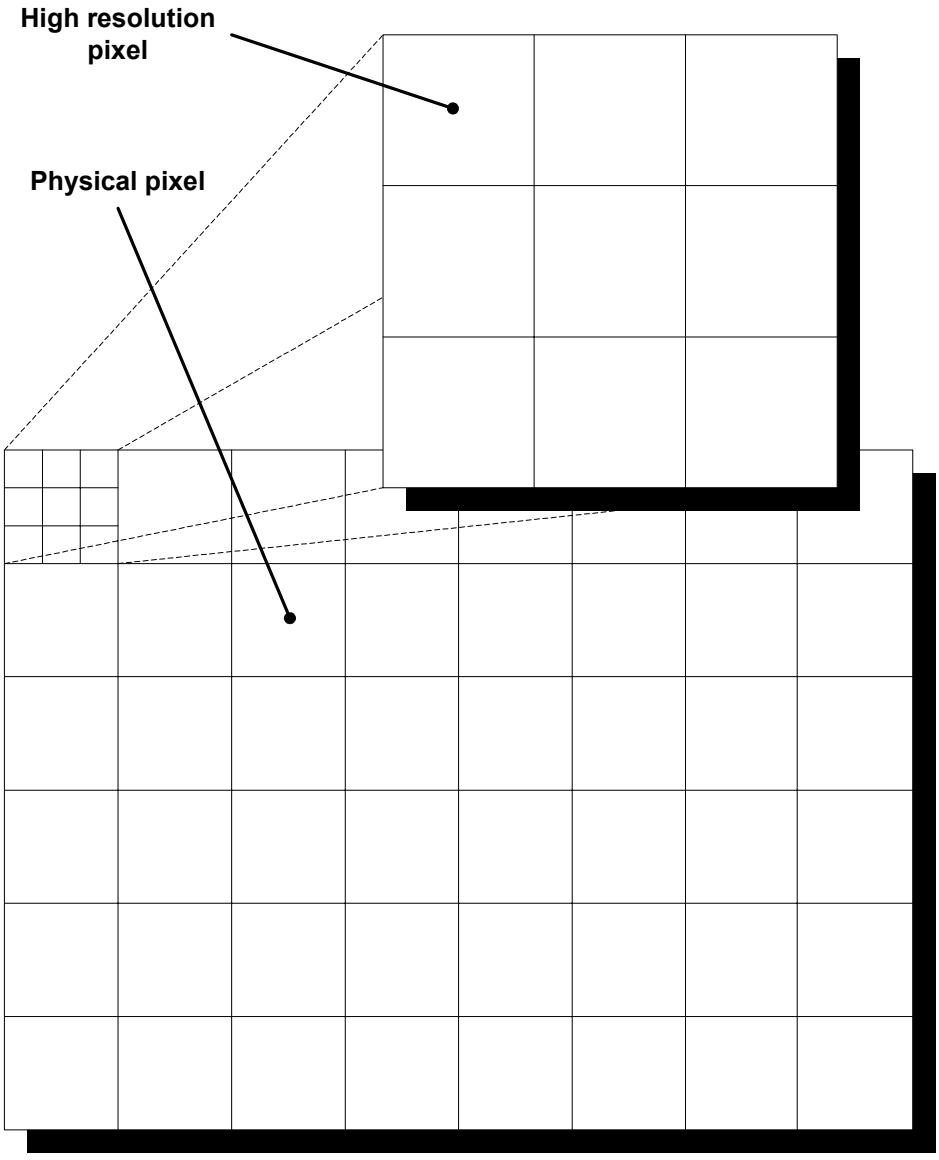
Antialiased fonts can be created with the emWin font converter. The general purpose of using antialiased fonts is to improve the appearance of text. While the effect of using high-quality antialiasing will be more visually pleasing than low-quality, computation time and memory consumption will increase proportionally. Low-quality (2bpp) fonts require twice the memory of non-antialiased (1bpp) fonts; high-quality (4bpp) fonts require four times the memory.

23.1.3 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:



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

23.2 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	
GUI_AA_DisableHiRes()	Disable high-resoluton coordinates.
GUI_AA_EnableHiRes()	Enable high-resolution coordinates.
GUI_AA_GetFactor()	Return the current antialiasing factor.
GUI_AA_SetFactor()	Set the current antialiasing factor.
Drawing functions	
GUI_AA_DrawArc()	Draw an antialiased arc.
GUI_AA_DrawLine()	Draw an antialiased line.
GUI_AA_DrawPolyOutline()	Draw the outline of an antialiased polygon of max. 10 points.
GUI_AA_DrawPolyOutlineEx()	Draw the outline of an antialiased polygon.
GUI_AA_FillCircle()	Draw an antialiased circle.
GUI_AA_FillPolygon()	Draw a filled and antialiased polygon.

23.3 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);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

23.4 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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) {
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  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 >= 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

23.5 Examples

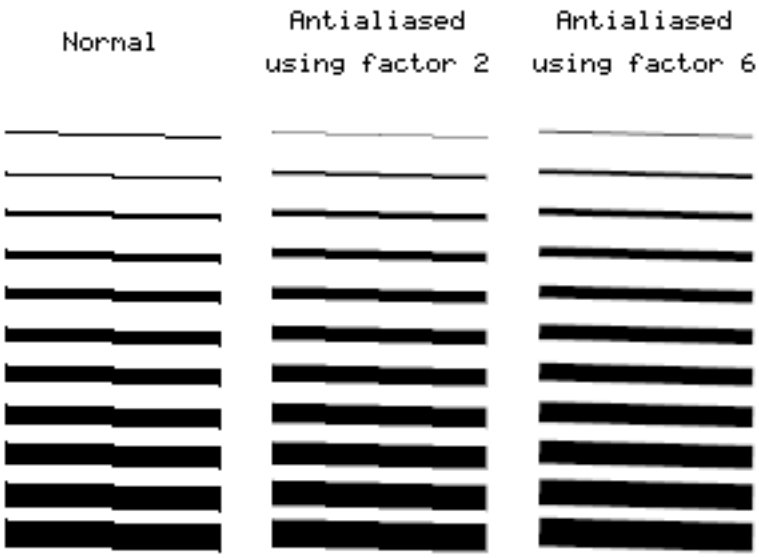
Different antialiasing factors

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\n\nusing factor 2", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
  x1 = 210; x2 = 290;
  /* Draw lines with antialiasing quality faktor 6 */
  GUI_AA_SetFactor(6);
  GUI_DispStringHCenterAt("Antialiased\n\nusing factor 6", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
}

/*********************************************************************
*
*                 main
*
**********************************************************************
*/

void main(void) {
  GUI_Init();
  DemoAntialiasing();
  while(1)
    GUI_Delay(100);
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screen shot of above example



Lines placed on high-resolution coordinates

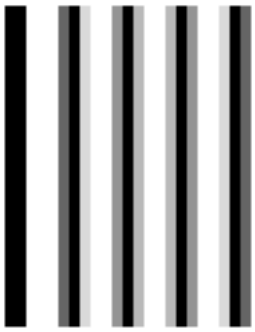
This example shows antialiased lines placed on high-resolution coordinates. It is available as AA_HiResPixels.c.

```
/*********************************************************************
*                 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);
  }
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
/*********************************************************************
*
*                 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 the pointer appears to make short "jumps", whereas in high-resolution mode there is no apparent jumping.
The example can be found as AA_HiResAntialiasing.c.

```
/*********************************************************************
*                 SEGGER MICROCONTROLLER 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)];
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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
*
**********************************************************************
*/
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
void main(void) {
  GUI_Init();
  ShowHiresAntialiasing();
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 24

Foreign Language Support

Text written in a foreign language like Arabic, Thai or Chinese contains characters, which are normally not part of the fonts shipped with emWin.
This chapter explains the basics like the Unicode standard, which defines all available characters worldwide and the UTF-8 encoding scheme, which is used by emWin to decode text with Unicode characters.
It also explains how to enable Arabic language support and how to render text with Shift-JIS (Japanese Industry Standard) encoding.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

24.1 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.

24.1.1 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");
```

24.1.2 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

24.1.3 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.

24.1.3.1 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\xa8\xe3\x83\xb3\xe3\x82\xb3\xe3\x83\xbc"
      "\xe3\x83\x87\xe3\x82\xa3\xe3\x83\xb3\xe3\x82\xb0",
  "2 - \xe3\x83\x86\xe3\x82\xad\xe3\x82\xb9\xe3\x83\x88",
  "3 - \xe3\x82\xb5\xe3\x83\x9d\xe3\x83\xbc\xe3\x83\x88",
  "English:",
  "1 - encoding",
  "2 - text",
  "3 - support"
};

void MainTask(void) {
  int i;
  GUI_Init();
  GUI_SetFont(&GUI_Font16_1HK);
  GUI_UC_SetEncodeUTF8();
  for (i = 0; i < GUI_COUNTOF(_apStrings); i++) {
    GUI_DispString(_apStrings[i]);
    GUI_DispNextLine();
  }
  while(1) {
    GUI_Delay(500);
  }
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

24.1.4 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.

24.1.4.1 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 reccomended 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,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                U16 * pBuffer, int BufferSize);
```

Parameter	Meaning
s	Pointer to UFT-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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 behviour 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.

24.1.4.2 Double byte functions

GUI_UC_DispString()

Description

This function displays the given double byte string.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void GUI_UC_DispString(const U16 GUI_FAR *s);
```

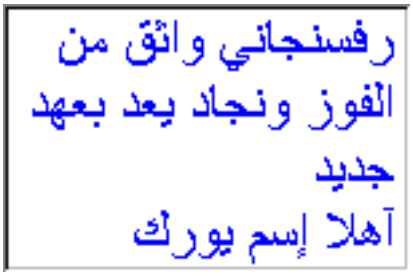
Parameter	Meaning
s	Pointer to double byte string.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

24.2 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.

24.2.1 Notation forms

The Arabic base character set is defined in the Unicode standard within the range from 0x0600 to 0x06FF. Unfortunately these character codes can not directly be used to get the character of the font for drawing it, because the notation form depends on the character position in the text. One character can have up to 4 different notation forms:

- One, if it is at the beginning of a word (initial)
- One, if it is at the end of a word (final)
- One, if it is in the middle of a word (medial)
- One, if the character stands alone (isolated)

But not each character is allowed to be joined to the left and to the right (double-joined). The character 'Hamza' for example always needs to be separated and 'Alef' is only allowed at the end or separated. Character combinations of the letters 'Lam' and 'Alef' should be transformed to a 'Ligature'. This means one character substitionally for the combination of 'Lam' and 'Alef'.
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	0xFEA0	Jeem
0x062D	0xFEA1	0xFEA2	0xFEA3	0xFEA4	Hah
0x062E	0xFEA5	0xFEA6	0xFEA7	0xFEA8	Khah
0x062F	0xFEA9	0xFEAA	-	-	Dal
0x0630	0xFEAB	0xFEAC	-	-	Thal
0x0631	0xFEAD	0xFEAE	-	-	Reh

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Base	Isolated	Final	Initial	Medial	Character
0x0632	0xFEAF	0xFEB0	-	-	Zain
0x0633	0xFEB1	0xFEB2	0xFEB3	0xFEB4	Seen
0x0634	0xFEB5	0xFEB6	0xFEB7	0xFEB8	Sheen
0x0635	0xFEB9	0xFEBA	0xFEBB	0xFEBC	Sad
0x0636	0xFEBD	0xFEBE	0xFEBF	0xFEC0	Dad
0x0637	0xFEC1	0xFEC2	0xFEC3	0xFEC4	Tah
0x0638	0xFEC5	0xFEC6	0xFEC7	0xFEC8	Zah
0x0639	0xFEC9	0xFECA	0xFECB	0xFECC	Ain
0x063A	0xFECD	0xFECE	0xFECF	0xFED0	Ghain
0x0641	0xFED1	0xFED2	0xFED3	0xFED4	Feh
0x0642	0xFED5	0xFED6	0xFED7	0xFED8	Qaf
0x0643	0xFED9	0xFEDA	0xFEDB	0xFEDC	Kaf
0x0644	0xFEDD	0xFEDE	0xFEDF	0xFEE0	Lam
0x0645	0xFEE1	0xFEE2	0xFEE3	0xFEE4	Meem
0x0646	0xFEE5	0xFEE6	0xFEE7	0xFEE8	Noon
0x0647	0xFEE9	0xFEEA	0xFEEB	0xFEEC	Heh
0x0648	0xFEED	0xFEEE	-	-	Waw
0x0649	0xFEEF	0xFEF0	-	-	Alef Maksura
0x064A	0xFEF1	0xFEF2	0xFEF3	0xFEF4	Yeh
0x067E	0xFB56	0xFB57	0xFB58	0xFB59	Peh
0x0686	0xFB7A	0xFB7B	0xFB7C	0xFB7D	Tcheh
0x0698	0xFB8A	0xFB8B	-	-	Jeh
0x06A9	0xFB8E	0xFB8F	0xFB90	0xFB91	Keheh
0x06AF	0xFB92	0xFB93	0xFB94	0xFB95	Gaf
0x06CC	0xFBFC	0xFBFD	0xFBFE	0xFBFF	Farsi Yeh

24.2.2 Ligatures

Character combinations of 'Lam' and 'Alef' needs to be transformed to ligatures. The following table shows how emWin transforms these combinations into ligatures, if the first letter is a 'Lam' (code 0x0644):

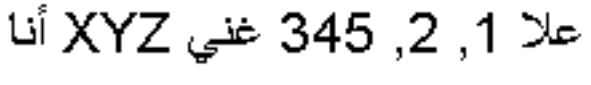
Second letter	Ligature (final)	Ligatur (elsewhere)
0x622, Alef with Madda above	0xFEF6	0xFEF5
0x623, Alef with Hamza above	0xFEF8	0xFEF7
0x625, Alef with Hamza below	0xFEFA	0xFEF9
0x627, Alef	0xFEFC	0xFEFB

24.2.3 Bidirectional text alignment

As mentioned above Arabic is written from the right to the left (RTL). But if for example the Arabic text contains numbers build of more than one digit these numbers should be written from left to right. And if Arabic text is mixed with European text a couple of further rules need to be followed to get the right visual alignment of the text.
The Unicode consortium has defined these rules in the Unicode standard. If bidirectional text support is enabled, emWin follows up most of these rules to get the right visual order before drawing the text.
emWin also supports mirroring of neutral characters in RTL aligned text. This is important if for example Arabic text contains parenthesis. The mirroring is done by replacing the code of the character to be mirrored with the code of a mirror partner whose image fits to the mirrored image. This is done by a fast way using a table containing all characters with existing mirror partners. Please note that support for mirroring further characters is not supported.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

The following sample shows how bidirectional text is rendered by emWin:

UTF-8 text	Rendering
\xd8\xb9\xd9\x84\xd8\xa7 1, 2, 345 \xd8\xba\xd9\x86\xd9\x8a XYZ \xd8\xa3\xd9\x86\xd8\xa7	

24.2.4 Requirements

Arabic language support is part of the basic package. Please note, that the standard fonts of emWin does not contain font files with Arabic characters. To create a Arabic font file the font converter is required

Memory

The bidirectional text alignment and Arabic character transformation uses app. 60 KB of ROM and app. 800 bytes of additional stack.

24.2.5 How to enable Arabic support

Per default emWin writes text always from the left to the right and there will be no Arabic character transformation as described above. To enable support for bidirectional text and Arabic character transformation please add the following line to the file `GUIConf.h`:

```
#define GUI_SUPPORT_BIDI 1
```

If enabled, emWin follows the rules of the bidirectional algorithm, described by the Unicode consortium, to get the right visual order before drawing text.

24.2.6 Sample

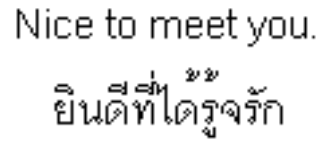
The sample folder contains the sample `FONT_Arabic`, which shows how to draw Arabic text. It contains an emWin font with Arabic characters and some small Arabic text samples.

24.2.7 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

24.3 Thai language support



The Thai alphabet uses 44 consonants and 15 basic vowel characters. These are horizontally placed, left to right, with no intervening space, to form syllables, words, and sentences. Vowels are written above, below, before, or after the consonant they modify, although the consonant always sounds first when the syllable is spoken. The vowel characters (and a few consonants) can be combined in various ways to produce numerous compound vowels (dipthongs and tripthongs).

24.3.1 Requirements

As explained above the Thai language makes an extensive usage of compound characters. To be able to draw compound characters in emWin, a new font type is needed, which contains all reqiured character information like the image size, image position and cursor incrementation value. From version 4.00 emWin supports a new font type with this information. This also means that older font types can not be used to draw Thai text.
Please note, that the standard fonts of emWin does not contain font files with Thai characters. To create a Thai font file the font converter of version 3.04 or newer is required.

Memory

The Thai language support needs no additional ROM or RAM.

24.3.2 How to enable Thai support

Thai support needs not explicitly enabled by a configuration switch. The only thing required to draw Thai text is a font file of type ’Extended’ created with the font converter from version 3.04 or newer.

24.3.3 Sample

The sample folder contains the sample `FONT_ThaiText.c`, which shows how to draw Thai text. It contains an emWin font with Thai characters and some small Thai text samples.

24.3.4 Font files used with Thai text

Font files used to render Thai text need to include at least all characters defined in the ’Thai’ range 0xE00-0xE7F.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

24.4 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.

24.4.1 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 seperate FontConverter documentation describes all you need for an efficient way of implementing Shift JIS in your emWin-projects.

24.4.2 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___,
```

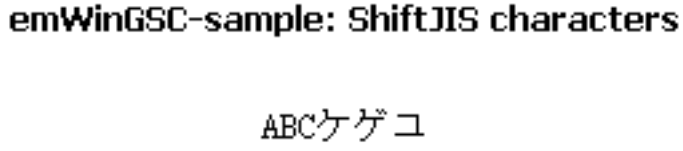
 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  _X___X__,
  _X___X__,
  _X___X__,
  _XXXX___,
  _X___X__,
  _X___X__,
  _X___X__,
  XXXXX___,
  ________,
  ________};

/* LATIN CAPITAL LETTER C */
static const unsigned char acFontSJIS13_0043[ 13] = { /* code 0043 */
  ________,
  ________,
  __XX_X__,
  _X__XX__,
  X____X__,
  X________,
  X________,
  X________,
  X________,
  _X___X__,
  __XXX___,
  ________,
  ________};

/* KATAKANA LETTER KE */
static const unsigned char acFontSJIS13_8350[ 26] = { /* code 8350 */
  __XX____,________,
  ___X____,________,
  ___X____,________,
  ___XXXXX,XXXX____,
  __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_,________,
  _______X_,________,
  _____X__,________,
  ____X___,________,
  __XX____,________,
  ________,________};

/* KATAKANA LETTER KO */
static const unsigned char acFontSJIS13_8352[ 26] = { /* code 8352 */
  ________,________,
  ________,________,
  __XXXXXX,XX______,
  ________,_X______,
  ________,_X______,
  ________,_X______,
  ________,_X______,
  ________,_X______,
  ________,_X______,
  _XXXXXXX,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 */
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
  ,{  14,  14,  2, (void *)&acFontSJIS13_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



 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 25

Display drivers

A display driver supports a particular family of display controllers and all displays which are connected to one or more of these controllers. The driver is essentially generic, meaning it can be configured by modifying the configuration files. The driver itself does not need to be modified. The configuration files contain 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 display 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 display controllers.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.1 Available drivers and supported display controllers

The following table lists the available drivers and which display controllers are supported by each:

Driver	Display Controller	Supported bits/pixel
GUIDRV_Fujitsu	Fujitsu MB87J2020 (Jasmine) Fujitsu MB87J2120 (Lavender)	1, 2, 4, 8, 16
GUIDRV_Lin	Any display controller with linear video memory in 1, 2, 4, 8 and 16 bits/pixel mode such as: Epson S1D13505 Epson S1D13700 (direct interface) Epson S1D13706 Epson S1D13715 (direct interface) Epson S1D13717 (direct interface) Epson S1D13719 (direct interface) Epson S1D13711 Epson S2D13705 Epson S2D13A05 Solomon SSD1905 Fujitsu MB86276 (Lime) Fujitsu MB86290A (Cremson) Fujitsu MB86291 (Scarlet) Fujitsu MB86292 (Orchid) Fujitsu MB86293 (Coral Q) Fujitsu MB86294 (Coral B) Fujitsu MB86295 (Coral P) Toshiba Capricorn 2 Epson S1D13A03, S1D13A04, S1D13A05 Epson S1D13513 (direct interface) Epson SED1352, S1D13502 Epson SED1353, S1D13503 Epson SED1354, S1D13504 Epson SED1356, S1D13506 Epson SED1374, S1D13704 Epson SED1375, S1D13705 Epson SED1376, S1D13706 Epson SED1386, S1D13806 ... and any other display controller with linear video memory (ARM or MIPS CPUs such as Sharp LH754xx, LH79520, Motorola Dragonball or NEC VR4181A) which should be accessed only in 32 bit mode. It can also be used without any display controller if only the contents of the of video memory should be managed.	1, 2, 4, 8, 16, 24, 32
GUIDRV_Mem	No controller, writes into main memory Requires ISR or special hardware to refresh LCD (monochrome displays)	1, 2
GUIDRV_MemC	No controller, writes into main memory Requires ISR or special hardware to refresh LCD (color displays)	3, 6
GUIDRV_Noritake	Noritake display GU256X128C-3900	1

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Driver	Display Controller	Supported bits/pixel
GUIDRV_Page1bpp	Samsung KS0713, Samsung S6B0713 Samsung KS0724, Samsung S6B0724 UltraChip UC1601, UC1606 Samsung KS0108B, Samsung S6B0108B Hitachi HD61202 Philips PCF8810, PCF8811 Philips PCF8535 New Japan Radio Company NJU6679 Philips PCD8544 Epson S1D15710 Solomon SSD1303 OLED controller Samsung S6B0719 Sitronix ST7522 Solomon SSD1805 UltraChip UC1608 New Japan Radio Company NJU6676 Epson S1D15721 Epson S1D15E05, S1D15E06 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
GUIDRV_Page4bpp	Sitronix ST7528	4
GUIDRV_SLin	Epson SED1330 Epson SED1335 Epson S1D13700 (indirect interface) RAIO 8835 RAIO 8822/8803 Toshiba T6963	1
GUIDRV_Vesa	Any VESA compatible hardware	8, 16
GUIDRV_Xylon	FPGA based display controller from Xylon	8, 16, 32
GUIDRV_0323	Solomon SSD0323 OLED controller	4
GUIDRV_07X1	Solomon SSD1854 Novatek NT7506 STMicroelectronics STE2010 Samsung KS0711, Samsung S6B0711 Samsung KS0741, Samsung S6B0741 Sitronix ST7541, ST7571 Tomato TL0350A	2
GUIDRV_1200	Toppoly C0C0 Toppoly C0E0	16
GUIDRV_13700	Epson S1D13700 Ultrachip UC1617 Solomon SSD1848	2
GUIDRV_13701	Epson S1D13701 OLED controller	9, 12
GUIDRV_13748	Epson S1D13748 Controller	16
GUIDRV_159A	Epson SED159A Sitronix ST7632	8
GUIDRV_1611	Epson S1D15E05, S1D15E06 Epson S1D15719 UltraChip UC1611 UltraChip UC1610	2 2 4 2
GUIDRV_161620	NEC µPD161620	12
GUIDRV_1781	Solomon SSD1768 Solomon SSD1781 Solomon SSD1783 Solomon SSD1797	16

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Driver	Display Controller	Supported bits/pixel
GUIDRV_6331	Samsung S6B33B0X, Samsung S6B33B1X, Samsung S6B33B2X	16
GUIDRV_6642X	Hitachi HD66420 Hitachi HD66421	2
GUIDRV_CompactColor_16	Himax HX8301, HX8312A, HX8347 Hitachi HD66766, HD66772, HD66789 Ilitek ILI9161, ILI9220, ILI9320, ILI9325 LG Electronics LGDP4531 MagnaChip D54E4PA7551 Novatek NT39122, NT7573 OriseTech SPFD5408, SPFD5420A Renesas R63401, R61509, R61516, R61505 Samsung S6D0110A, S6D0117, S6D0129 Sharp LR38825, LCY-A06003 Sitronix ST7712, ST7628, ST7637 Solomon SSD1289 Toshiba JBT6K71 Epson S1D13743	16
GUIDRV_66750	Hitachi HD66750 Hitachi HD66753	2
GUIDRV_7529	Sitronix ST7529	1, 4, 5
GUIDRV_7920	Sitronix ST7920	1
GUIDRV_8822	Raio RA8822	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	LCD Controller	Supported bits/pixel
GUIDRV_Template	Driver template. Can be used as a starting point for writing a new driver. Part of the basic package	-
GUIDRV_Null	Empty driver. (Performs no output) Can be used for measurement purpose. Part of the basic package.	-

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.2 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

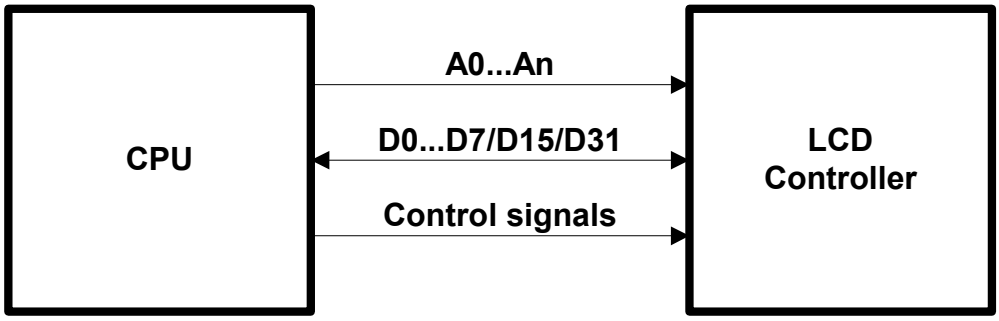
The following explains these interfaces and how to configure them. Note that not all configuration macros are always required. For details about which macros are required please refer to the detailed driver descriptions later in this chapter.

25.2.1 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

25.2.1.1 Typical block diagram

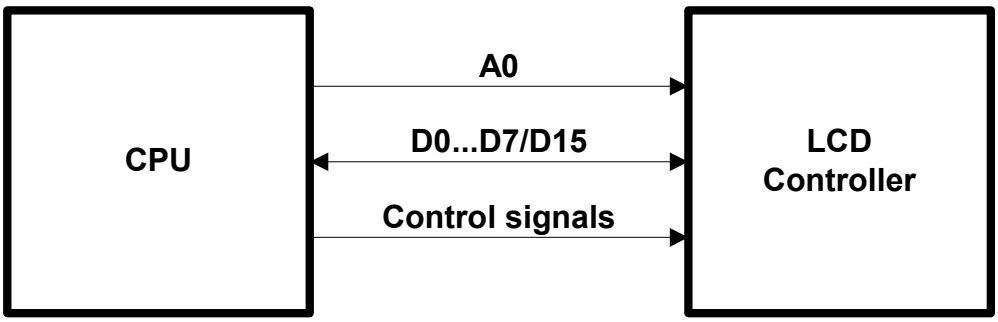


25.2.2 Simple bus interface

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.2.2.1 Typical block diagram



8 (16) data bits, one address bit and 2 or 3 control lines are used to connect the CPU and one LCD controller. Four macros inform the LCD driver how to access each controller used. If the LCD controller(s) is connected directly to the address bus of the CPU, configuration is simple and usually consists of no more than one line per macro. If the LCD controller(s) is connected to I/O pins, the bus interface must be simulated, which takes about 5-10 lines of program per macro (or a function call to a routine which simulates the bus interface). The signal **A0** is also called **C/D** (Command/Data), **D/I** (Data/Instruction) or **RS** (Register select), depending on the display controller.

25.2.2.2 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 the display controller with A0 - line low.
F	LCD_WRITE_A1	Writes a byte to the display controller with A0 - line high.
F	LCD_WRITEM_A1	Writes several bytes to the LCD controller with A0 - line high.

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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 the display 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 the display 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.

25.2.2.3 Sample routines for connection to I/O pins

Examples can be found in the folder `Sample\LCD_X`:

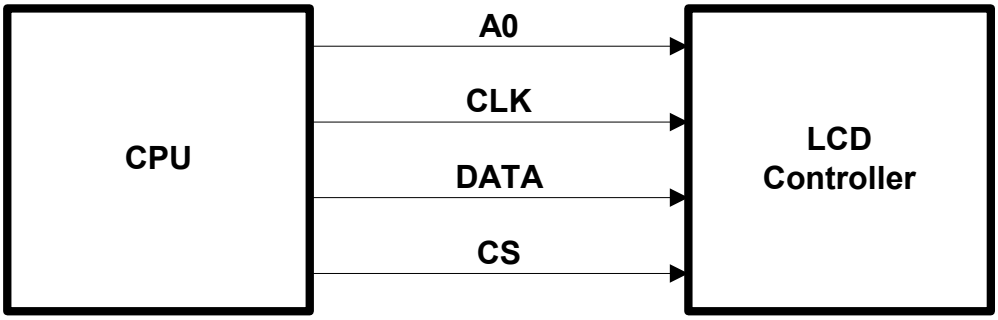
- `LCD_X_6800.c`, port routines for the 6800 interface.
- `LCD_X_8080.c`, port routines for the 8080 interface.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.2.3 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.

25.2.3.1 Typical block diagram



25.2.3.2 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 the display controller with A0 (C/D) - line low.
F	LCD_WRITE_A1	Writes a byte to the display controller with A0 (C/D) - line high.
F	LCD_WRITEM_A1	Writes several bytes to the LCD controller with A0 (C/D) - line high.

LCD_WRITE_A0

Description

Writes a byte to the display 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 the display controller with A0 - line high.

Type

Function replacement

Prototype

```
#define LCD_WRITE_A1(Byte)
```

Parameter	Meaning
Byte	Byte to write.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

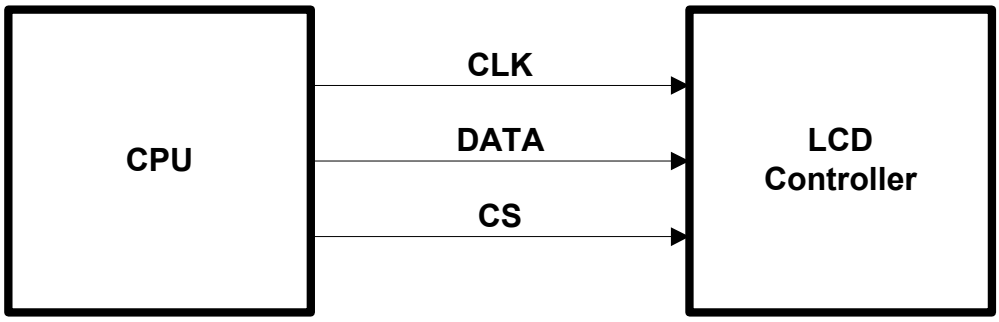
25.2.3.3 Sample routines for connection to I/O pins

An example can be found in the folder Sample\LCD_X:

- LCD_X_SERIAL.c, port routines for a serial interface

25.2.4 3 pin SPI interface

25.2.4.1 Typical block diagram



25.2.4.2 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 the display controller.
F	LCD_WRITEM	Writes several bytes to the LCD controller.

LCD_WRITE

Description

Writes a byte to the LCD controller.

Type

Function replacement

Prototype

```
#define LCD_WRITE(Byte)
```

Parameter	Meaning
Byte	Byte to write.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

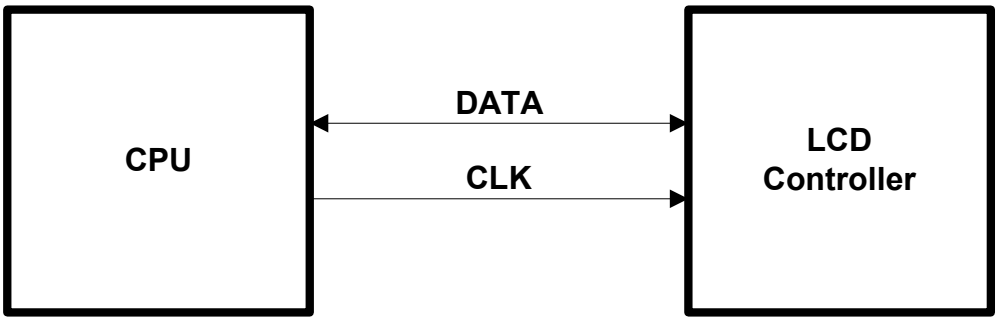
25.2.4.3 Sample routines for connection to I/O pins

An example can be found in the folder Sample\LCD_X:

- LCD_X_Serial_3Pin.c, port routines for a 3 pin serial interface

25.2.5 I2C bus interface

25.2.5.1 Typical block diagram



25.2.5.2 Macros used by a I2C bus 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 the display controller.
F	LCD_WRITE_A1	Writes a data byte to the display controller.
F	LCD_WRITEM_A1	Writes several data bytes to the LCD controller.

The purpose of these macros are exactly the same as explained under ’simple bus interface’.

25.2.5.3 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 I2C bus interface

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.2.6 Non readable displays

Some display controllers with a simple bus interface do not support reading back display data. Especially displays which are connected via SPI interface often have this limitation. In this case we recommend using a display data cache. For details how to enable a display data cache please refer to the detailed driver descriptions later in this chapter.
On systems with a very small RAM it is sometimes not possible to use a display data cache. If a display is not readable and a display data cache can not be used some features of emWin will not work. The list below shows these features:

- Cursors and Sprites
- XOR-operations, required for text cursors in EDIT and MULTIEDIT widgets
- Alpha blending
- Antialiasing

This is valid for all drivers where one data unit (8 or 16 bit) represents one pixel. Display drivers, where one data unit represents more than one pixel, can not be used if no display data cache is available and the display is not readable. An example is the LCDPage1bpp driver where one byte represents 8 pixels.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.3 Display orientation

The way how to configure the display orientation depends on the display driver to be used. Wheras the display orientation of the most common drivers is runtime configurable some drivers need to be configured at compile time.
Wheras the display orientation of the GUIDRV_Lin is determined by selecting the desired driver file the configuration of drivers with a simple bus interface like the configuration of the display orientation of GUIDRV_CompactColor_16 needs to be done in the configuration file of the driver.

25.3.1 Runtime configuration

The display orientation of the most common driver GUIDRV_Lin is determined by selecting the desired driver file.

25.3.2 Compile time configuration

The display orientation of some drivers with simple bus interface like GUIDRV_CompactColor_16 needs to be configured at compile time in the configuration file of the driver.

25.3.3 Display orientations

There are 8 possible display orientations; the display can be turned 0°, 90°, 180° or 270° and can also be viewed from top or from bottom. The default orientation is 0° and top view. These 4 *2 = 8 different display orientations can also be expressed as a combination of 3 binary switches: X-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-done 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.

Display	Orientation macros in driver configuration file
	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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Display	Orientation macros in driver configuration file
	Use #define LCD_SWAP_XY 1
	Use #define LCD_SWAP_XY 1 #define LCD_MIRROR_X 1
	Use #define LCD_SWAP_XY 1 #define LCD_MIRROR_X 1 #define LCD_MIRROR_Y 1
	Use #define LCD_SWAP_XY 1 define LCD_MIRROR_Y 1

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.4 Display driver callback function

A display driver requires a callback function. It is called by the driver for several tasks. One task is putting the display driver into operation which is also explained in the chapter 'Configuration'. It is also called for other tasks which require hardware related operations like switching the display on and off or setting a lookup table entry.

LCD_X_DisplayDriver()

Description

This is the callback function of the display driver. It is called by the display driver for several jobs. It passes a command and a pointer to a data structure to the callback routine. The command tells the callback function what should be done. If the command requires parameters they are passed through the data pointer `pData`. It points to a structure whose format depends on the command.

Prototype

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd,
                        void * pData);
```

Parameter	Meaning
`LayerIndex`	Zero based layer index.
`Cmd`	Command to be executed. Detailed descriptions below.
`pData`	Pointer to a data structure.

Return value

The routine should return -2 if an error orrurs, -1 if the command is not handled by the function and 0 if the command has been successfully executed.

25.4.1 Commands passed to the callback function

The following explains the common commands passed to the callback function. For details about display driver specific commands please refer to the detailed display driver description later in this chapter. They are described under the topic 'Additional callback commands'.

LCD_X_INITCONTROLLER

As mentioned above the application should initialize the display controller and put it into operation if the callback routine receives this command. No parameters are passed on this command. Typically an initialization routine which initializes the registers of the display controller should be called in reaction of this command.

Parameters

none

LCD_X_SETVRAMADDR

This command is passed by the driver to tell the callback routine the start address of the video RAM. The typicall reaction should be writing the address to the frame buffer start address register.

Parameters

`pData` points to a data structure of type `LCD_X_SETVRAMADDR_INFO`:

Data type	Element	Meaning
`void *`	pVRAM	Points to the start address of the video RAM. This address is typically written to the video RAM base address register of the display controller.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LCD_X_ON

The display should be switched on.

Parameters

none

LCD_X_OFF

The display should be switched off.

Parameters

none

LCD_X_SETLUTENTRY

A lookup table entry should be set. The typicall reaction should be writing an entry into the lookup table of the display controller.

Parameters

`pData` points to a data structure of type `LCD_X_SETLUTENTRY_INFO`:

Data type	Element	Meaning
`LCD_COLOR`	`Color`	RGB value of the color to be written to the LUT. Please note that the format required by the hardware could be different to the RGB format.
`U8`	`Pos`	Zero based index of the LUT entry to be set.

LCD_X_SETORG

The function is used in relation with virtual screens. It is called if the origin of the display should be set. A typicall reaction can be modifying the frame buffer start address.

Parameters

`pData` points to a data structure of type `LCD_X_SETORG_INFO`:

Data type	Element	Meaning
`int`	`xPos`	New X-position of the physical display position within the virtual screen.
`int`	`yPos`	New Y-position of the physical display position within the virtual screen.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.5 Detailed display driver descriptions

This chapter currently does not contain all drivers listed in the table shown at the beginning of the chapter. They will be added on demand.

25.5.1 GUIDRV_Lin

This driver supports all display controllers with linear video memory accessible via full bus interface described later in this chapter. It can be used with and without a display controller. The driver does only manage the contents of the video memory. It does not send any commands to the display controller or assumes any specific registers. So it is independent of the register interface of the display controller and can be used for managing each linear mapped video memory.

Color depth and display orientation

The driver consists of several files. They are named GUIDRV_Lin_[0]_BPP.c. where the optional '0' stands for the desired display orientation and 'BPP' for the the color depth. The following table shows the driver files and the configuration macros which should be used to create and link the driver during the initialization:

Display driver	Identifier	Color depth and orientation
GUIDRV_Lin_1.c	GUIDRV_LIN_1	1bpp, default orientation
GUIDRV_Lin_2.c	GUIDRV_LIN_2	2bpp, default orientation
GUIDRV_Lin_4.c	GUIDRV_LIN_4	4bpp, default orientation
GUIDRV_Lin_8.c	GUIDRV_LIN_8	8bpp, default orientation
GUIDRV_Lin_16.c	GUIDRV_LIN_16	16bpp, default orientation
GUIDRV_Lin_24.c	GUIDRV_LIN_24	24bpp, default orientation
GUIDRV_Lin_32.c	GUIDRV_LIN_32	32bpp, default orientation
GUIDRV_Lin_OX_16.c	GUIDRV_LIN_OX_16	16bpp, X axis mirrored
GUIDRV_Lin_OX_32.c	GUIDRV_LIN_OX_32	32bpp, X axis mirrored
GUIDRV_Lin_OY_32.c	GUIDRV_LIN_OY_16	16bpp, Y axis mirrored
GUIDRV_Lin_OSY_16.c	GUIDRV_LIN_OSY_16	16bpp, Y axis mirrored, X and Y swapped

The table above does not show each possible combination of orientation and color depth. But it should make clear how the right driver can be selected. Each combination of orientation and color depth is possible. Please note that currently not all combinations are shipped with the driver. If the required combination is not available please contact support.

Supported hardware

Controllers

The driver supports all systems with linear mapped video memory.

Bits per pixel

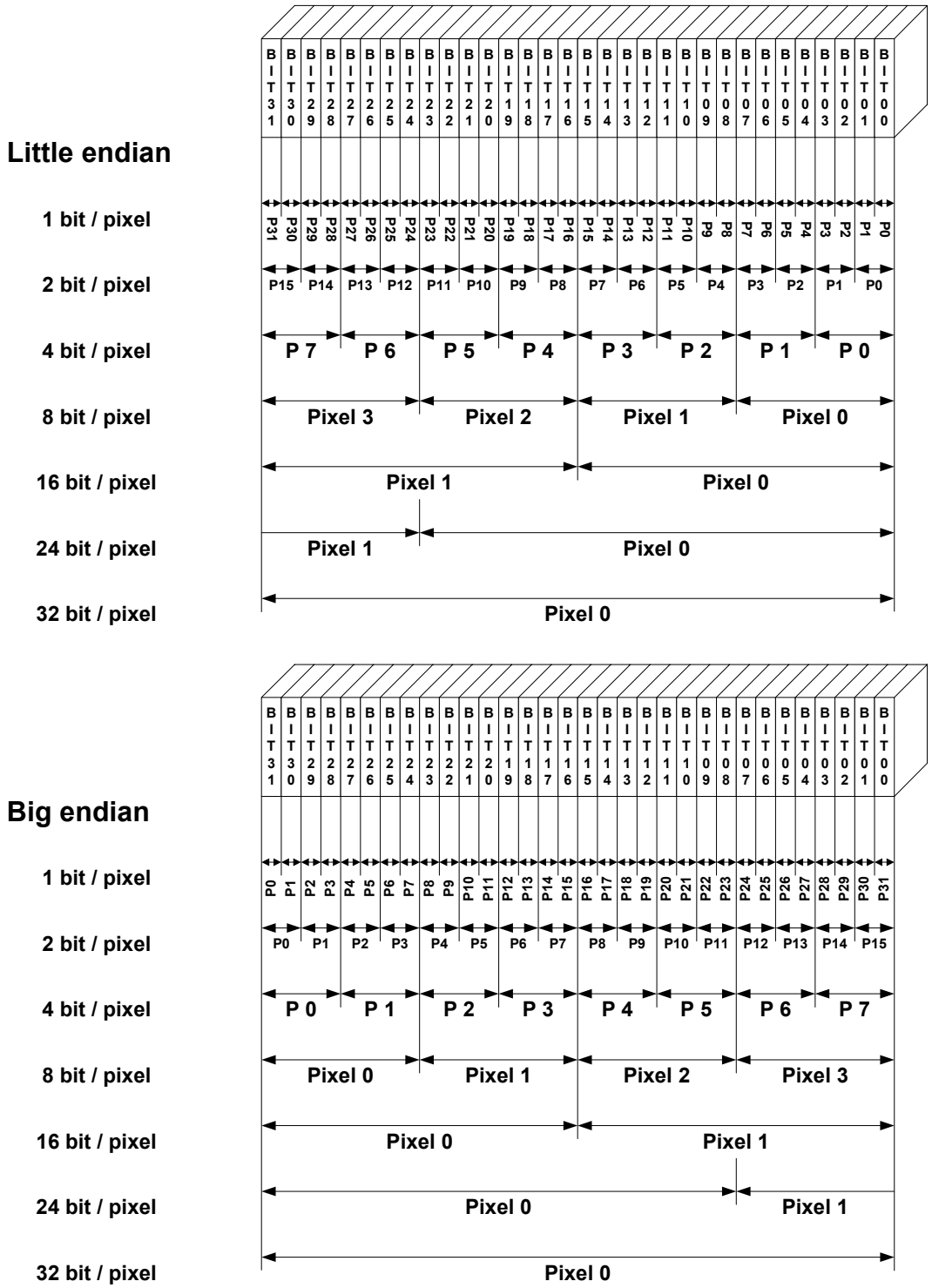
Supported color depths are 1, 2, 4, 8, 16, 24 and 32 bits per pixel.

Interfaces

The driver supports a full bus interface from the CPU to the video memory. The video memory needs to be accessible 8, 16 and 32 bit wise.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Display data RAM organization



The picture above shows the relation between the display memory and the pixels of the LCD in terms of the color depth and the endian mode.

Little endian video mode

Least significant bits are used and output first. The least significant bits are for the first (left-most) pixel.

Big endian video mode

Most significant bits are used and output first. The most significant bits are for the first (left-most) pixel.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

RAM requirements of the driver

None.

Additional callback commands

None.

Available configuration macros (compile time 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 (default) for little endian mode.

Available configuraion routines (runtime configuration)

The following table lists the available runtime configuration routines:

Routine	Explanation
LCD_SetSizeEx()	Changes the size of the visible area.
LCD_SetVRAMAddrEx()	Changes the video RAM start address.
LCD_SetVSizeEx()	Changes the size of the virtual display area.

Configuration sample

The following shows how to create a display driver device with this driver and how to configure it:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,    // Display driver
                           GUICC_8666,      // Color conversion
                           0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx    (0, 320,  240);          // Physical display size in pixels
  LCD_SetVSizeEx   (0, 320,  480);          // Virtual  display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0x20000000); // Video RAM start address
}
```

Additional driver functions

None

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.5.2 GUIDRV_CompactColor_16

Supported hardware

Controllers

This driver works with the following display controllers:

- Epson S1D13743
- Himax HX8301, HX8312, HX8347
- Hitachi HD66766, HD66789, HD66772
- Ilitek ILI9320, ILI9325, ILI9220, ILI9161
- LG Electronics LGDP4531
- MagnaChip D54E4PA7551
- Novatek NT39122, NT7573
- OriseTech SPFD5408, SPFD5420A
- Renesas R63401, R61509, R61516, R61505
- Samsung S6D0129, S6D0110A, S6D0117
- Sitronix ST7628, ST7712, ST7637
- Sharp LR38825, LCY-A06003
- Solomon SSD1289, SSD2119
- Toshiba JBT6K71

Bits per pixel

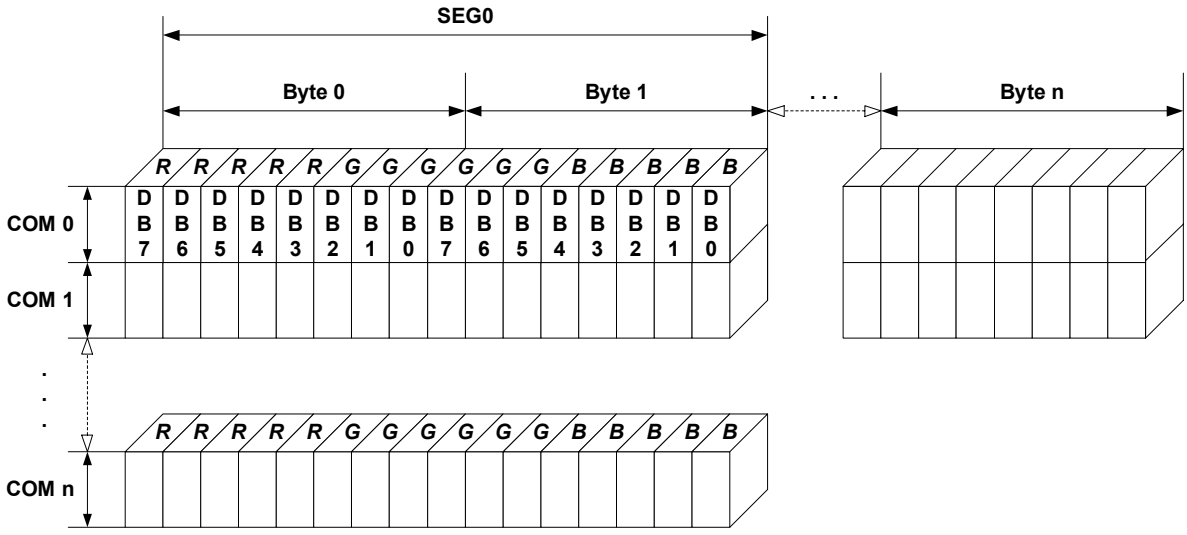
Supported color depth is 16 bpp.

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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_WRITEM_A1 to transfer the data to the display controller. The default buffer size is 500 bytes.

Additional driver functions

None.

Driver selection and configuration

To be able to use this driver the following macro definition needs to be added to the configuration file `LCDConf.h`:

```
#define LCD_USE_COMPACT_COLOR_16
```

After this define has been added the display driver assumes the driver specific configuration file `LCDConf_CompactColor_16.h` in the configuration folder. All further compile time configuration macros should be defined in this file.

Available configuration macros (compile time configuration)

Controller selection

The following table shows the available macro:

Macro	Explanation
LCD_CONTROLLER	Selects the display controller to be used: 66700 - Sharp LR38825 66701 - Renesas R63401 66701 - Renesas R61509 66701 - OriseTech SPFD5420A 66702 - Solomon SSD1289 66702 - SSD2119 66703 - Toshiba JBT6K71 66704 - Sharp LCY-A06003 66705 - Samsung S6D0129 66705 - Renesas R61505 66706 - MagnaChip D54E4PA7551 66707 - Himax HX8312 66708 - Ilitek ILI9320 66708 - Ilitek ILI9325 66708 - OriseTech SPFD5408 66708 - LG Electronics LGDP4531 66709 - Novatek NT39122 66709 - Sitronix ST7628 66709 - Sitronix ST7637 66709 - Renesas R61516 66710 - Novatek NT7573 66711 - Epson S1D13743 66712 - Himax HX8347 66766 - Hitachi HD66766 66766 - Samsung S6D0110A 66766 - Ilitec ILI9161 66772 - Hitachi HD66772 66772 - Samsung S6D0117 66772 - Sitronix ST7712 66772 - Himax HX8301 66772 - Ilitec ILI9220 66789 - Hitachi HD66789

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Display configuration

The following table shows the available configuration macros:

Macro	Explanation
LCD_MIRROR_X	Activate to mirror X-axis.
LCD_MIRROR_Y	Activate to mirror Y-axis.
LCD_SWAP_XY	Activate to swap X- and Y-axis.

For details please refer to 'Display orientation' explained earlier in this chapter.

Hardware access

The following table shows the available configuration macros which can be defined in this file for configuring the hardware access:

Macro	Explanation
LCD_NUM_DUMMY_READS	Number of required dummy reads if a read operation should be executed. The default value is 2. If using a serial interface the display controllers HD66766 and HD66772 need 5 dummy reads. Sharp LR38825 needs 3 dummy reads with a 8-bit bus.
LCD_REG01	This macro is only required if a Himax HX8312A is used. Unfortunately the register 0x01 (Control register 1) contains orientation specific settings as well as common settings. So this macro should contain the contents of this register.
LCD_SERIAL_ID	With a serial interface this macro defines the ID signal of the device ID code. It should be 0 (default) or 1.
LCD_USE_SERIAL_3PIN	Should be set to 1 if the serial interface is used. Default is 0.
LCD_USE_PARALLEL_16	Should be set to 1 if the 16 bit parallel interface is used. Default is 0.
LCD_WRITE_BUFFER_SIZE	Defines the size of the write buffer. Using a write buffer increases the performance of the driver. If multiple pixels should be written with the same color, the driver first fills the buffer and then writes the contents of the buffer with one execution of the macro LCD_WRITEM_A1, instead of multiple macro executions. The default buffer size is 500 bytes.
LCD_WRITE_A0	Write a byte to display controller with RS-line low.
LCD_WRITE_A1	Write a byte to display controller with RS-line high.
LCD_READM_A1	Read multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) from display controller with RS-line high.
LCD_WRITEM_A1	Write multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) to display controller with RS-line high.
LCD_WRITEM_A0	Write multiple bytes (8 bit parallel interface) or multiple words (16 bit parallel interface) to display controller with RS-line low.

The driver initializes the 'Driver Output Mode' and 'Entry Mode' register itself. The user does not need to initialize this registers in `LCD_X_InitController()`.

Available configuraion routines (runtime configuration)

The following table lists the available runtime configuration routines:

Routine	Explanation
LCD_SetSizeEx()	Changes the size of the visible area.

Configuration sample

The following shows how to select the driver and how it can be configured:

LCDConf.h

As explained above it should include the following for selecting the driver:

```
#define LCD_USE_COMPACT_COLOR_16
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LCDConf_CompactColor_16.h

This file contains the display driver specific configuration and could look as the following:

```
//
// General configuration of LCD
//
#define LCD_CONTROLLER      66709 // Renesas R61516
#define LCD_BITSPERPIXEL       16
#define LCD_SWAP_RB             1
#define LCD_USE_PARALLEL_16     1
#define LCD_MIRROR_Y            1
//
// Simple bus configuration
//
void LCD_X_Write01_16(unsigned short c);
void LCD_X_Write00_16(unsigned short c);
void LCD_X_WriteM01_16(unsigned short * pData, int NumWords);
void LCD_X_WriteM00_16(unsigned short * pData, int NumWords);
void LCD_X_ReadM01_16 (unsigned short * pData, int NumWords);
#define LCD_WRITE_A1(Word) LCD_X_Write01_16(Word)
#define LCD_WRITE_A0(Word) LCD_X_Write00_16(Word)
#define LCD_WRITEM_A1(Word, NumWords) LCD_X_WriteM01_16(Word, NumWords)
#define LCD_WRITEM_A0(Word, NumWords) LCD_X_WriteM00_16(Word, NumWords)
#define LCD_READM_A1(Word, NumWords)  LCD_X_ReadM01_16(Word, NumWords)
```

Creating a display driver device

The following shows how to create a display driver device with this driver and how to configure it:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_COMPACT_COLOR_16, // Display driver
                           GUICC_M565,              // Color conversion
                           0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx(0, 240, 320);                    // Physical display size in pixels
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.5.3 LCDTemplate driver

This driver is part of the basic package and can be easily adapted to each display controller. It contains the complete functionality needed for a display driver.

Adapting the template driver

To adapt the driver to a currently not supported display controller you only have to adapt the routines `_SetPixelIndex()` and `_GetPixelIndex()`. The upper layers calling this routines makes sure that the given coordinates are in range, so that no check on the parameters needs to be performed.
If a display is not readable the function `_GetPixelIndex()` won't be able to read back the contents of the display data RAM. In this case a display data cache should be implemented in the driver, so that the contents of each pixel is known by the driver. If no data cache is available in this case some functions of emWin will not work right. These are all functions which need to invert pixes. Especially the XOR draw mode and the drawing of text cursors (which also uses the XOR draw mode) will not work right. A simple application which do not use the XOR draw mode will also work without adapting the function `_SetPixelIndex()`.
In a second step it should be optimized to improve drawing speed.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.5.4 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

25.6 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.

25.6.1 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.

User defined routines

Routine	Explanation
LCD_X_InitController()	Called by the display driver to get the display controller initialized.

LCD: LCD layer routines

Routine	Explanation
"Get" group	
LCD_GetBitsPerPixel()	Return the number of bits per pixel.
LCD_GetBitsPerPixelEx()	Returns the number of bits per pixel of given layer/display.
LCD_GetFixedPalette()	Return the fixed palette mode.
LCD_GetFixedPaletteEx()	Returns the fixed palette mode of given layer/display.
LCD_GetNumColors()	Return the number of available colors.
LCD_GetNumColorsEx()	Returns the number of available colors of given layer/display.
LCD_GetVXSize()	Return virtual X-size of LCD in pixels.
LCD_GetVXSizeEx()	Returns virtual X-size of given layer/display in pixels.
LCD_GetVYSize()	Return virtual Y-size of LCD in pixels.
LCD_GetVYSizeEx()	Returns virtual Y-size of given layer/display in pixels.
LCD_GetXMag()	Returns the magnification factor in x.
LCD_GetXMagEx()	Returns the magnification factor of given layer/display in x.
LCD_GetXSize()	Return physical X-size of LCD in pixels.
LCD_GetXSizeEx()	Returns physical X-size of given layer/display in pixels.
LCD_GetYMag()	Returns the magnification factor in y.
LCD_GetYMagEx()	Returns the magnification factor of given layer/display in y.
LCD_GetYSize()	Return physical Y-size of LCD in pixels.
LCD_GetYSizeEx()	Returns physical Y-size of given layer/display in pixels.
Configuration group	
LCD_SetSizeEx()	Sets the physical size in pixels of the given layer.
LCD_SetVRAMAddrEx()	Sets the address of the video RAM of the given layer.
LCD_SetVSizeEx()	Sets the size of the virtual display area in pixels of the given layer.

25.6.2 Callback routines

LCD_X_InitController()

Description

The function is called by the display controller. The job of the routine is to get the display controller registers initialized right.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void LCD_X_InitController(unsigned LayerIndex);
```

Parameter	Meaning
LayerIndex	Index of layer to be initialized.

Add. information

The sample folder contains several samples for a large number of display controllers. This function replaces the macro LCD_INIT_CONTROLLER which was used in older versions.

25.6.3 LCD layer routines

25.6.3.1 "Get" group

LCD_GetBitsPerPixel()

Description

Returns the number of bits per pixel.

Prototype

```
int LCD_GetBitsPerPixel(void);
```

Return value

Number of bits per pixel.

LCD_GetBitsPerPixelEx()

Description

Returns the number of bits per pixel.

Prototype

```
int LCD_GetBitsPerPixelEx(int Index);
```

Parameter	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 pixesl. 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 pixesl. In most cases, the virtual size is equal to the physical size.

Prototype

```
int LCD_GetVXSizeEx(int Index);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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.

LCD_GetXSize(), LCD_GetYSize()

Description

Returns the physical X- or Y-size, respectively, of the LCD in pixels.

Prototypes

```
int LCD_GetXSize(void)
int LCD_GetYSize(void)
```

Return value

Physical X/Y-size of the display.

LCD_GetXSizeEx(), LCD_GetYSizeEx()

Description

Returns the physical X- or Y-size, respectively, of the LCD in pixels.

Prototype

```
int LCD_GetXSizeEx(int Index);
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
int LCD_GetYSizeEx(int Index);
```

Parameter	Meaning
Index	Layer index.

Return value

Physical X/Y-size of the display.

25.6.3.2 Configuration group

LCD_SetSizeEx()

Description

Sets the physical size of the visible area of the given display/layer.

Prototype

```
int LCD_SetSizeEx(int LayerIndex, int xSize, int ySize);
```

Parameter	Meaning
LayerIndex	Layer index.
xSize	X-Size in pixels of the visible area of the given layer.
ySize	Y-Size in pixels of the visible area of the given layer.

Return value

0 on success, 1 on error.

Add. information

The function requires a display driver which is able to manage dynamically changes of the display size. If the display driver does not support this feature the function fails.

LCD_SetVRAMAddrEx()

Description

Sets the address of the video RAM.

Prototype

```
int LCD_SetVRAMAddrEx(int LayerIndex, U32 Addr);
```

Parameter	Meaning
LayerIndex	Layer index.
Addr	Start address of the video RAM.

Return value

0 on success, 1 on error.

Add. information

The function requires a display driver which is able to manage dynamically changes of the video RAM address. If the display driver does not support this feature the function fails.

LCD_SetVSizeEx()

Description

Sets the size of the virtual display area.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
int LCD_SetVSizeEx(int LayerIndex, int xSize, int ySize);
```

Parameter	Meaning
LayerIndex	Layer index.
xSize	X-Size in pixels of the virtual area of the given layer.
ySize	Y-Size in pixels of the virtual area of the given layer.

Return value

0 on success, 1 on error.

Add. information

The function requires a display driver which is able to manage dynamically changes of the virtual display size. If the display driver does not support this feature the function fails.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 26

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 'desk-top' 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

26.1 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.

26.1.1 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.

26.1.2 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

26.2 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

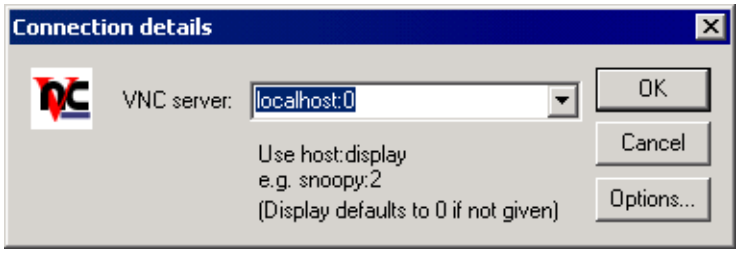
The VNC implementation of emWin has been tested with version 3.3.3R2 and should work with this version or later versions.

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.

26.2.1 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.

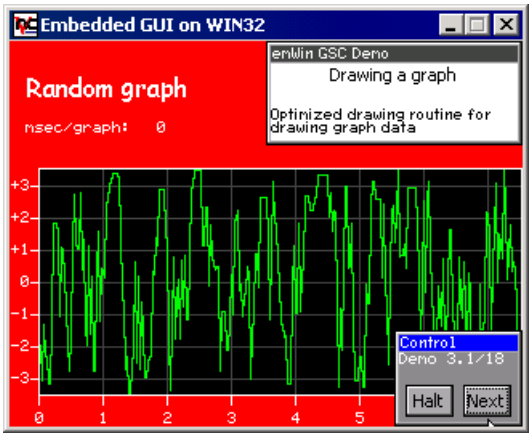
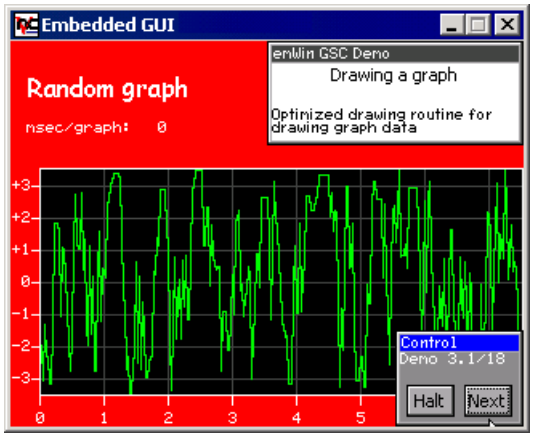
Connecting to a VNC server running on a different PC or the target

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'.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Screenshot

The following screenshots shows the viewer:

Connected to the simulation	Connected to the target
	

26.3 emWin VNC server

26.3.1 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.

26.3.2 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 incomming connection is detected and then calls `GUI_VNC_Process()`, which is the implementation of the actual server.

26.3.3 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

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

26.4 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.

26.5 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 resut in only minor accellerations. 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).

26.6 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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);
  ...
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 27

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

27.1 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
GUI_Delay()	Delay for a specified period of time.
GUI_Exec()	Execute callback functions (all jobs).
GUI_Exec1()	Execute one callback function (one job only).
GUI_GetTime()	Return the current system time.

GUI_Delay()

Description

Delays for a specified period of time.

Prototype

```
void GUI_Delay(int Period);
```

Parameter	Explanation
Period	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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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()`.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Chapter 28

Configuration

Before emWin can be used on a target system, the software needs to be configured. Configuring means modifying the configuration files which usually reside in the (sub)directory `Config`. We try to keep the configuration as simple as possible, but there are some configuration routines which need to be modified in order for the system to work properly.
The following items need to be configured:

- Memory area to be used by emWin
- Display driver to be used for drawing operations
- Color conversion routines to be used
- Display controller initialization

The following chapter explains the configuration of emWin in detail.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

28.1 What needs to be configured?

The configuration is basically divided into two parts: GUI-configuration and LCD-configuration. GUI-configuration means configuration of available features, default colors and -fonts and the configuration of available memory. The LCD-configuration is more hardware dependent and has to define the physical size of the display, the display driver and the color conversion routines to be used. For details about 'color conversion routines' please refer to the chapter 'Colors'.
A further part is configuring the simulation. But this is not required for the target hardware and not part of this chapter. For details about configuring the simulation please refer to the chapter 'Simulator'.

28.2 Runtime- and compile time configuration

There are 'C' and 'H' files to be configured. The configuration in the header files is fixed at compile time and can not be changed whereas the configuration done in the 'C' files can be changed at runtime. This makes it possible to create a library which is largely configuration independent and can be used with any display and any driver. This requires that the configuration routines described in this chapter are not part of the library but of the application.

28.3 Initialization process of emWin

The illustration shows process of initialization. From the application only `GUI_Init()` needs to be called. The configuration routines explained below are called during the internal initialization process.

```
GUI_Init()
  ├─ GUI_X_Config()
  │    ├─ GUI_ALLOC_AssignMemory()    /* Required */
  │    └─ GUI_ALLOC_SetAvBlockSize()  /* Required */
  ├─ LCD_X_Config()
  │    ├─ GUI_DEVICE_CreateAndLink()  /* Required */
  │    ├─ LCD_SetVSizeEx()            /* Required */
  │    ├─ LCD_SetSizeEx()             /* Required */
  │    ├─ LCD_SetVRAMAddrEx()         /* Optional */
  │    ├─ GUI_TOUCH_Calibrate()       /* Optional */
  │    └─ GUI_TOUCH_SetOrientation()  /* Optional */
  └─ LCD_X_DisplayDriver()
```

GUI_X_Config()

It is called at the very first beginning of the initialization process to make sure that memory is assigned to emWin. Within this routine `GUI_ALLOC_AssignMemory()` and `GUI_ALLOC_SetAvBlockSize()` must be called to assign a memory block to emWin and set the average memory block size. The functions are explained later in this chapter.

LCD_X_Config()

This function is called immediately after `GUI_X_Config()`. The main purpose of this routine is creating a display driver device and selecting the color conversion routines. Further it is responsible for setting the display size. If a touch screen is used it should also be configured here.

LCD_X_DisplayDriver()

At a later point of the initialization process the function `LCD_X_DisplayDriver()` is called. It is called directly by the display driver. During the initialization process the task of this routine is putting the display controller into operation. A detailed explanation of the routine follows later in this chapter.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

28.4 Runtime configuration

The following table shows the available runtime configuration files located in the sub-folder Config:

Configuraion file	Purpose
GUIConf.c	Configuration of available memory.
LCDConf.c	Configuration of the display size, the display driver and the color conversion routines.
SIMConf.c	Configuration of the simulation (not part of this chapter).
GUI_X.c	Configuration of timing routines.

28.4.1 Customizing GUIConf.c

The purpose of this modul is to provide emWin with the function GUI_X_Config() which is responsible for assigning a memory block to the memory management system. This requires knowledge about the memory requirement of the used components. The separate chapter 'Memory requirements' contains a detailed description of the memory requirements (RAM and ROM) of the individual emWin modules.
Per default GUIConf.c is located in the (sub)directory Config and contains the routine GUI_X_Config() which is responsible to assign a memory block to emWin. It is not cogently required to leave it in the file GUIConf.c. The routine GUI_X_Config() can be located anywhere in the application.

GUI_X_Config()

Description

Calling this function is the very first thing done by the initialization process. It is responsible to assign a memory block to emWin. This block is managed by the internal memory management system. The memory block needs to be accessible 8, 16 and 32 bit wise.

Prototype

```
void GUI_X_Config(void);
```

Add. information

Please note that not the complete memory block can be used by the application, because a small overhead of the memory is used by the management system itself. Each memory block requires app. 12 bytes for management purpose.

28.4.1.1 API functions to be used in GUI_X_Config()

The following table shows the API functions which must be called within GUI_X_Config():

Routine	Explanation
GUI_ALLOC_AssignMemory()	Assigns a memory block for the memory mamagement system.
GUI_ALLOC_SetAvBlockSize()	Sets the average size of the memory blocks. The bigger the block size, the less number of memory blocks are available.

GUI_ALLOC_AssignMemory()

Description

The function assigns the one and only memory block to emWin which is used by the internal memory management system. This function should be called typically from GUI_X_Config().

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void GUI_ALLOC_AssignMemory(void * p, U32 NumBytes);
```

Parameter	Meaning
p	Pointer to the memory block which should be used by emWin.
NumBytes	Size of the memory block in bytes.

Add. information

It is required to set the value of GUI_ALLOC_SIZE defined in GUIConf.h to 0 to enable runtime memory configuration. Please note that not the complete memory block can be used by the application, because a small overhead of the memory is used by the management system itself.

GUI_ALLOC_SetAvBlockSize()

Description

Sets the average block size of the memory blocks allocated by the memory management system. The block size affects the number of maximum available blocks. If for example an application uses some listviews with a large number of entries it makes sense to set the average block size to a small value. On the other hand if an application uses the memory management primarily for a few memory devices or image decompression the average size should be set to a bigger value. The recommended range is between 32 and 1024. The value depends on the application.
This function should be called typically from GUI_X_Config().

Prototype

```
void GUI_ALLOC_SetAvBlockSize(U32 BlockSize);
```

Parameter	Meaning
BlockSize	Average block

Add. information

The average block size is used to calculate the maximum number of available memory blocks:

Max. # of blocks = Size of memory in bytes / (BlockSize + sizeof(BLOCK_STRUCT))

BlockStruct means an internal structure whose size depends on GUI_DEBUG_LEVEL. If it is >0 the size will be 12 bytes, otherwise 8 bytes. Please note that the structure size also depends on the used compiler.

28.4.2 Customizing LCDConf.c

The purpose of this module is to provide emWin with the required display configuration routine and the callback function for the display driver. These are the following functions:

Routine	Explanation
LCD_X_Config()	Configuration routine for creating the display driver device, setting the color conversion routines and the display size.
LCD_X_DisplayDriver()	Callback routine called by the display driver for putting the display controller into operation.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

LCD_X_Config()

Description

As described in the table above this routine is responsible to create a display driver device, set the right color conversion routines and for configuring the physical display size.

Prototype

```
void LCD_X_Config(void);
```

Add. information

Depending on the used display driver it could also be required to set the video RAM address, initialize a custom palette or some else. Please refer to the display driver documentation to check if there are any additional requirements. The functions available for configuration purpose in this routine are listed and explained later in this chapter.

Sample

The following shows a typical sample implementation:

```
//
// Set display driver and color conversion for 1st layer
//
GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, GUICC_565, 0, 0);
//
// Display driver configuration, required for Lin-driver
//
LCD_SetSizeEx    (0, 320, 240);
LCD_SetVSizeEx   (0, 320, 240);
LCD_SetVRAMAddrEx(0, (void *)0x200000);
```

LCD_X_DisplayDriver()

Description

This is the callback function of the display driver. It is called for several purposes. During the process of initialization only a few are of interest, actually the display controller initialization and the setting of the video RAM address.

Prototype

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd,
                        LCD_X_DATA * pData);
```

Parameter	Meaning
LayerIndex	Zero based layer index.
Cmd	(see table below)
pData	Pointer to a data structure of type LCD_X_DATA.

Permitted values for element Cmd during the initialization process	
LCD_X_INITCONTROLLER	The display controller should become initialized. Typically an initialization routine which initializes the registers of the display controller should be called in reaction of this command.
LCD_X_SETVRAMADDR	The element pVRAM of the data structure pointed by pData points to the start address of the video RAM. This address is typically written to the video RAM base address register of the display controller.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Used elements of LCD_X_DATA:

Data type	Element	Meaning
void *	pVRAM	Pointer to the start address of the video RAM.

Return value

The routine should return -2 if an error orrurs, -1 if the command is not handled by the function and 0 if the command has been successfully executed.

Add. information

Please refer to the display driver documentation for more information about the commands passed to the routine by the display driver.

Samples

The folder Sample\LCDConf\ contains a lot of sample implementations of this routine which can be used as starting point.

28.4.2.1 API functions to be used in LCD_X_Config()

The following table shows the API functions which are available for configuration purpose within LCD_X_Config():

Routine	Explanation
GUI_DEVICE_CreateAndLink()	Creates a display driver device and associates the color conversion routines to be used.
GUI_TOUCH_SetOrientation()	Sets the orientation of the touch screen. This routine is only required if a touch screen is used which does not operates in its default orientation.
GUI_TOUCH_Calibrate()	Calibrates the touch screen.
LCD_SetLUTEx()	Initializes the lookup table with the given palette. This function is only required if a custom palette should be used.
LCD_SetSizeEx()	Required to set the physical size of the display.
LCD_SetVRAMAddrEx()	Sets the address of the video RAM. It is only required if a display driver with linear mapped video RAM is used.
LCD_SetVSizeEx()	Required only if the virtual display size is different to the physical size.

For information about LCD_... please refer to chapter '???'.
For information about GUI_TOUCH_... please refer to chapter '???'.

GUI_DEVICE_CreateAndLink()

Description

This routine creates the display driver device, sets the color conversion routines to be used for accessing the display and it links the driver device into the device list of the given layer. LCD_X_Config() is called immediately after GUI_X_Config(). This makes sure that the memory configuration already has been done and the driver is able to allocate memory.
The required memory for a display driver device is app. 50 bytes + the driver specific memory. For details about the memory requirements of the individual display drivers please refer to the chapter 'Display driver'.

Prototype

```
GUI_DEVICE * GUI_DEVICE_CreateAndLink(const GUI_DEVICE_API * pDeviceAPI,
                                      const LCD_API_COLOR_CONV * pColorConvAPI,
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
                                    U16 Flags, int LayerIndex);
```

Parameter	Meaning
pDeviceAPI	Pointer to the display driver to be used. The chapter 'Display drivers' contains a table of the available display drivers.
pColorConvAPI	Pointer to the color conversion routines to be used. The chapter 'Colors' contains a table with the available color conversion routines.
Flags	Should be zero.
LayerIndex	Layer which should be managed by the driver.

Return value

On success the function returns a pointer to the created device object, otherwise it returns NULL.

Add. information

Please note that the used driver also determines the display orientation in some cases. This differs from driver to driver. For details about the display orientation please refer to the chapter 'Display driver'.

28.4.3 Customizing GUI_X.c

This file is the location of the timing routines, the debugging routines and the kernel interface routines:

28.4.3.1 Timing routines

GUI_X_Delay()

Description

Returns after a specified time period in milliseconds.

Prototype

```
void GUI_X_Delay(int Period)
```

Parameter	Meaning
Period	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)
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Return value

The current system time in milliseconds, of type integer.

28.4.3.2 Debugging routines

GUI_X_ErrorOut(), GUI_X_Warn(), GUI_X_Log()

Description

These routines are called by emWin with debug information in higher debug levels in case a problem (Error) or potential problem is discovered. The routines can be blank; they are not required for the functionality of emWin. In a target system, they are typically not required in a release (production) build, since a production build typically uses a lower debug level.
Fatal errors are output using GUI_X_ErrorOut() if (GUI_DEBUG_LEVEL >= 3)
Warnings are output using GUI_X_Warn() if (GUI_DEBUG_LEVEL >= 4)
Messages are output using GUI_X_Log() if (GUI_DEBUG_LEVEL >= 5)

Prototypes

```
void GUI_X_ErrorOut(const char * s);
void GUI_X_Warn(const char * s);
void GUI_X_Log(const char * s);
```

Parameter	Meaning
s	Pointer to the string to be sent.

Add. information

This routine is called by emWin to tramsmit 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.

28.4.3.3 Kernel interface routines

Detailed descriptions for these routines may be found in ’Execution Model: Single Task/Multitask’.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

28.5 Compile time configuration

The following table shows the available compile time configuration files located in the subfolder Config:

Configuraion file	Purpose
GUIConf.h	Configuration of available features, number of layers, default fonts and default colors.
LCDConf.h	Configuration of the used display driver(s).

28.5.1 Customizing GUIConf.h

As described above the file should contain the configuration of available features and the configuration of the default font. Each emWin shipment comes with a GUIConf.h file which includes a basic configuration which can be used as a starting point.

28.5.1.1 Configuring the available features of emWin

The following table shows the available configuration macros:

Type	Macro	Default	Explanation
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_BIDI	0	Activates the bidirectional language support required for drawing Arabic and Hebrew text.
B	GUI_SUPPORT_CURSOR	(see expl.)	Per default cursers are enabled if either GUI_SUPPORT_TOUCH or GUI_SUPPORT_MOUSE has been enabled. If cursors should be shown without enabling one of these options it should be set to 1.
B	GUI_SUPPORT_MEMDEV	0	Enables optional memory device support.
B	GUI_SUPPORT_MOUSE	0	Enables the optional mouse support.
B	GUI_SUPPORT_ROTATION	1	Enables text rotation support.
B	GUI_SUPPORT_SPRITE	1	In case of using cursors, but no sprites, the macro can be set to 0 to reduce the ROM requirement.
B	GUI_SUPPORT_TOUCH	0	Enables optional touch-screen support.
B	GUI_WINSUPPORT	0	Enables optional window manager support.

28.5.1.2 Default font and default color configuration

The following table shows the available configuration macros:

Type	Macro	Default	Explanation
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.

The default colors and fonts of the widgets which are part of the optional window manager can also be configured. For details please refer to the widget chapter.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

28.5.1.3 Advanced GUI configuration options

The following table shows the available configuration macros:

Type	Macro	Default	Explanation
S	GUI_DEBUG_LEVEL	1 (target) 4 (simulation)	Defines the debug level, which determines how many checks (assertions) are performed by emWin and if debug errors, warnings and messages are output. Higher debug levels generate bigger code.
N	GUI_MAXTASK	4	Define the maximum number of tasks from which emWin is called to access the display when multitasking support is enabled (see Chapter 13: "Execution Model: Single Task/ Multitask").
F	GUI_MEMCPY	---	This macro allows replacement of the memcpy function.
F	GUI_MEMSET	---	Replacement of the memset function of the GUI.
N	GUI_NUM_LAYERS	1	Defines the maximum of available layers/ displays.
B	GUI_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_UNICODE	1	Enables support for Unicode characters embedded in 8-bit strings.
B	GUI_TRIAL_VERSION	0	Marks the compiler output as evaluation version.
B	GUI_WINSUPPORT	0	Enables optional window manager support.

GUI_MEMCPY

This macro allows replacement of the memcpy function of the GUI. On a lot of systems, memcpy takes up a considerable amount of time because it is not optimized by the compiler manufacturer. emWin contains an alternative memcpy routine, which has been optimized for 32 bit CPUs. On a lot of systems this routine should generate faster code than the default memcpy routine. However, this is still a generic "C"-routine, which in a lot of systems can be replaced by faster code, typically using either a different "C" routine, which is better optimized for the particular CPU or by writing a routine in Assembly language.
To use the optimized emWin routine add the following define to the file GUIConf.h:

```
#define GUI_MEMCPY(pSrc, pDest, NumBytes) GUI__memcpy(pSrc, pDest, NumBytes)
```

GUI_MEMSET

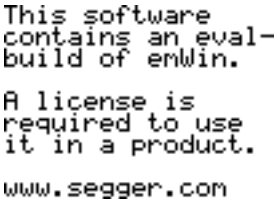
This macro allows replacement of the memset function of the GUI. On a lot of systems, memset takes up a considerable amount of time because it is not optimized by the compiler manufacturer. We have tried to address this by using our own memset() Routine GUI__memset. However, this is still a generic "C"-routine, which in a lot of systems can be replaced by faster code, typically using either a different "C" routine, which is better optimized for the particular CPU, by writing a routine in Assembly language or using the DMA.
If you want to use your own memset replacement routine, add the define to the GUIConf.h file.

GUI_TRIAL_VERSION

This macro can be used to mark the compiler output as an evaluation build. It should be defined if the software is given to a third party for evaluation purpose (typically with evaluation boards).
Note that a special license is required to do this; the most common licenses do not permit redistribution of emWin in source or object code (relinkable) form. Please contact sales@segger.com if you would like to do this.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

If GUI_TRIAL_VERSION is defined, the following message is shown when calling GUI_Init():



This message is always shown in the upper left corner of the display and is normally visible for 1 second. The timing is implemented by a call GUI_X_Delay(1000). The functionality of emWin is in no way limited if this switch is active.

Sample

```
#define GUI_TRIAL_VERSION 1
```

28.5.2 Customizing LCDConf.h

This file contains general configuration options required for compiling the display driver(s) which need not to be changed at runtime. The available configuration options depend on the used display driver. For details about the available configuration options please refer to the chapter 'Display drivers'. The detailed driver description shows the available configuration options for each display driver.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

29.1 Performance

The following chapter shows driver benchmarks on different targets and performance values of image drawing operations.

29.1.1 Driver benchmark

We use a benchmark test to measure the speed of the display drivers on available targets. This benchmark is in no way complete, but it gives an approximation of the length of time required for common operations on various targets.

Configuration and performance table

CPU	LCD Controller (Driver)	bpp	Bench1 Filling	Bench2 Small fonts	Bench3 Big fonts	Bench4 Bitmap 1bpp	Bench5 Bitmap 2bpp	Bench6 Bitmap 4bpp	Bench7 Bitmap 8bpp	Bench8 DDP bitmap
V850SB1 (20MHz)	S1D13806 (1300)	8	16.7M	339K	1.59M	1.52M	240K	459K	83K	1.25M
V850SB1 (20MHz)	S1D13806 (1300)	16	8.33M	326K	1.45M	1.49M	391K	388K	214K	806K
ARM720T (50MHz)	(internal) (3200)	16	7.14M	581K	1.85M	1.96M	694K	645K	410K	2.94M
ARM926EJ-S (200MHz)	(internal) (3200)	16	123M	3.79M	5.21M	7.59M	2.27M	2.21M	1.77M	15.2M

Bench1: Filling

Bench the speed of filling. An area of 64*64 pixels is filled with different colors.

Bench2: Small fonts

Bench the speed of small character output. An area of 60*64 pixels is filled with small-character text.

Bench3: Big fonts

Bench the speed of big character output. An area of 65*48 pixels is filled with big-character text.

Bench4: Bitmap 1bpp

Bench the speed of 1bbp bitmaps. An area of 58*8 pixels is filled with a 1bpp bitmap.

Bench 5: Bitmap 2bpp

Bench the speed of 2bbp bitmaps. An area of 32*11 pixels is filled with a 2bpp bitmap.

Bench6: Bitmap 4bpp

Bench the speed of 4bbp bitmaps. An area of 32*11 pixels is filled with a 4bpp bitmap.

Bench7: Bitmap 8bpp

Bench the speed of 8bbp bitmaps. An area of 32*11 pixels is filled with a 8bpp bitmap.

Bench8: Device-dependent bitmap, 8 or 16 bpp

Bench the speed of bitmaps 8 or 16 bits per pixel. An area of 64*8 pixels is filled with a bitmap. The color depth of the tested bitmap depends on the configuration. For configurations <= 8bpp, a bitmap with 8 bpp is used; 16bpp configurations use a 16-bpp bitmap.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

29.1.2 Image drawing performance

The purpose of the following table is to show the drawing performance of the various image formats supported by emWin. The measurement for the following table has been done on an ARM922T CPU (Sharp LH7A404) running with 200MHz and with 15 bpp display color depth (fixed palette = 555) using the LCDLin driver:

Image format	Megapixels / second
Internal bitmap format: 1bpp 'C' file	17.186
Internal bitmap format: 4bpp 'C' file	3.897
Internal bitmap format: 8bpp 'C' file	4.017
Internal bitmap format: 8bpp 'C' file, without palette	4.478
Internal bitmap format: 16bpp 'C' file, high color 555	13.363
Internal bitmap format: 16bpp 'C' file, high color 565	1.336
Internal bitmap format: 24bpp 'C' file, true color 888	1.671
Internal bitmap format: RLE4 'C' file	6.144
Internal bitmap format: RLE8 'C' file	6.806
Internal bitmap format: RLE16 'C' file	3.740
BMP file 8bpp	4.115
BMP file 16bpp	1.134
BMP file 24bpp	1.544
BMP file 32bpp	1.525
BMP file RLE4	6.998
BMP file RLE8	6.345
GIF file	1.285
JPEG file, gray	0.516
JPEG file, gray, progressive	0.438
JPEG file, H1V1	0.402
JPEG file, H1V1, progressive	0.280
JPEG file, H2V2	0.602
JPEG file, H2V2, progressive	0.431

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

29.2 Memory requirements

The operation area of emWin varies widely, depending primarily on the application and features used. In the following sections, memory requirements of different modules are listed as well as memory requirement of sample applications. The memory requirements of the GUI components have been measured on a system as follows:

ARM7, IAR Embedded workbench V4.42A, Thumb mode, Size optimization

29.2.1 Memory requirements of the GUI components

The following table shows the memory requirements of the main components of emWin. These values depend a lot on the compiler options, the compiler version and the used CPU. Please note that the listed values are the requirements of the basic functions of each module and that there are several additional functions available which have not been considered in the table:

Component	ROM	RAM	Explanation
Window manager	+ 6.2 kB	+ 2.5 kB	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	+ 4.7 kB	+ 7 kB	Additional memory requirements of a 'Hello world' application when using memory devices. The configured GUI_ALLOC_SIZE for the measurement was 8000.
Antialiasing	+ 4.5 kB	+ 2 * LCD_XSIZE	Additional memory requirements for the antialiasing software item.
Driver	+ 2 - 8 kB	20	The memory requirements of the driver depend on the configured driver and if a data cache is used or not. With a data cache the driver requires more RAM. For details please refer to the driver documentation.
Multilayer	+ 2 - 8 kB	-	If working with a multi layer or a multi display configuration additional memory for each additional layer is required, because each layer requires its own driver.
Core	5.2 kB	80 Byte	Memory requirements of a typicall 'Hello world' application without using additional software items.
Core / JPEG	12 kB	38 kB	Basic routines for drawing JPEG files.
Core / GIF	3.3 kB	17 kB	Basic routines for drawing GIF files.
Core / Sprites	4.7 kB	16 Byte	Routines for drawing sprites and cursors.
Core / Fonts	(see explanation)	-	Please refer to the font subchapter for details about the ROM requirements of the standard fonts shipped with emWin.
Widgets	4.5 kB	-	This is the approximately basic ROM requirement for the widgets depending on the individual core functions used by the widgets.
Widget / BUTTON	1 kB	40 Bytes	*1
Widget / CHECKBOX	1 kB	52 Bytes	*1
Widget / DROPDOWN	1.8 kB	52 Bytes	*1
Widget / EDIT	2.2 kB	28 Bytes	*1
Widget / FRAMEWIN	2.2 kB	12 Bytes	*1
Widget / GRAPH	2.9 kB	48 Bytes	*1
Widget / GRAPH_DATA_XY	0.7 kB	-	*1
Widget / GRAPH_DATA_YT	0.6 kB	-	*1
Widget / HEADER	2.8 kB	32 Bytes	*1
Widget / LISTBOX	3.7 kB	56 Bytes	*1
Widget / LISTVIEW	3.6 kB	44 Bytes	*1
Widget / MENU	5.7 kB	52 Bytes	*1
Widget / MULTIEDIT	7.1 kB	16 Bytes	*1

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Component	ROM	RAM	Explanation
Widget / MULTIPAGE	3.9 kB	32 Bytes	*1
Widget / PROGBAR	1.3 kB	20 Bytes	*1
Widget / RADIOBUTTON	1.4 kB	32 Bytes	*1
Widget / SCROLLBAR	2 kB	14 Bytes	*1
Widget / SLIDER	1.3 kB	16 Bytes	*1
Widget / TEXT	0.4 kB	16 Bytes	*1

*1. The listed memory requirements of the widgets contain the basic routines required for creating and drawing the widget. Depending of the specific widget there are several additional functions available which are not listed in the table.

29.2.2 Stack requirements

The basic stack requirement is app. 600 bytes. If using the window manager additional 600 bytes should be calculated. For memory devices further additional 200 bytes are recommended.

29.3 Memory requirements of sample applications

This section shows the requirements of some sample applications. The following table contains the summary of the memory requirements. The values are in bytes unless specified other:

Sample	GUI core	Fonts	Application	Startup code	Library	Total	GUI core	Application	Stack	Total
	ROM						RAM			
Hello world	5.9 kB	1.8 kB	38	0.3 kB	0.1 kB	**8.1 kB**	62	-	272	**334**
Window application	43 kB	12.5 kB	2.7 kB	0.3 kB	1.5 kB	**60 kB**	5.2 kB	40	1.4 kB	**6.6 kB**

For details about the samples please refer to the following sections.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

29.3.1 "Hello world" project

The following shows the memory requirement of a "Hello world" project. It 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)

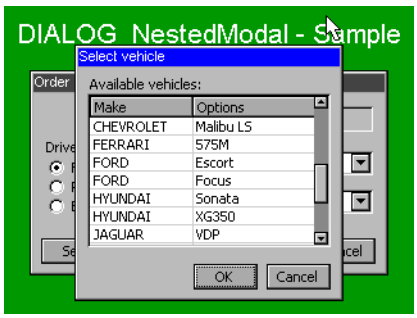
29.3.1.3 Memory requirements

The following table shows the memory requirement of the project:

Description	ROM	RAM
GUI code and const data	5.9 kB	62 Bytes
GUI data (fonts)	1.8 kB	--
Application	38 Bytes	--
Stack	--	272 Bytes
Startup code	0.3 kB	--
Library	71 Bytes	--
Total		
	8.1 kB	**334 Bytes**

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

29.3.2 Window application



This project is a "typical" application using the window manager and different widgets. It 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 DROPDOWN_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	RAM
GUI code and const data	43 kB	5.2 kB
GUI data (fonts)	12.5 kB	--
Application	2.7 kB	40
Stack	--	1.4 kB
Startup code	0.3 kB	--
Library	1.5 kB	--
Total		
	60 kB	**6.6 kB**

© 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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 op-timization"
"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 at lease one of the following standards:

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

Limited number of arguments in a function pointer call

But some compilers are not 100% ANSI 'C' compatible and have for example a limitation regarding the number of arguments in a function pointer call:

```
typedef int tFunc(int a, int b, int c, int d, int e,
                  int f, int g, int h, int i, int j);

static int _Func(int a, int b, int c, int d, int e,
                 int f, int g, int h, int i, int j) {
  return a + b + c + d + e + f + g + h;
}
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
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 is an executable with a lot of unused code. In this case the use of a library would be very helpful. For detailes 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 configuraton and with your compiler. If you wand 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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

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\Tutorial\ProblemReport.c`:

```
/*********************************************************************
*                 SEGGER MICROCONTROLLER SYSTEME GmbH                *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                        emWin problem report                        *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File                : ProblemReport.c
CPU                 :
Compiler/Tool chain :
Problem description :
----------------------------------------------------------------------
```

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

```
*/

#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.

 © 1997 - 2009 SEGGER Microcontroller GmbH & Co. KG

Index

A

Access addresses, defining 28
Access routines, defining 28
Active window 246
Additional software 27
Alias macro 28
Aliasing 653
ANSI 17
Antialiasing 653–665
  of fonts 653–655
  qualities 653–654
  software for 653
  with high-resolution coordinates 653, 655–656
API reference
  antialising 656
  BMP file support 114
  BUTTON widget 310
  CHECKBOX widget 324
  colors 207
  cursors 650
  device simulator 39, 44
  dialogs 598
  EDIT widget 338, 351, 416, 454
  fonts 142
  FRAMEWIN widget 369
  generic touch-screen 630
  GIF file support 124, 132
  GRAPH widget 393
  graphics 86
  JPEG file API 120
  kernel interface routines 240
  LCD layer 705
  LISTBOX widget 435
  LISTWHEEL widget 477
  memory devices 215
  MENU widget 493
  MESSAGEBOX widget 507
  Multi layer 618
  MULTIEDIT widget 510
  MULTIPAGE widget 522
  PROGBAR widget 534
  RADIO widget 541
  routines common to all widgets 304
  SCROLLBAR widget 552
  SLIDER widget 559
  Sprites 644
  text 62
  TEXT widget 566
  timing and execution 720
  TREEVIEW widget 575
  values 76
  VNC 715
  window manager 259
  WINDOW widget 592
  WM routines used for widgets 304
Application program interface (API) 20, 705
Arcs, drawing 107–108
ASCII 62, 137, 153, 155
Auto device object 231–233

B

Background window 248
Banding memory device 229–231
Best palette option 185, 188, 190
Binary switch macro 27
Binary values, displaying 82–83
Bitmap converter 19, 179–192
  command line usage 188–190
  supported input formats 180
  using for color conversion 185–186
Bitmap file support 114
Bitmaps 179–192
  color conversion of 185–186
  device-dependent (DDB) 181
  device-independent (DIB) 181
  drawing 94–135
  full-color mode 185
  generating "C" files from 179–185
  manipulating 180
  RLE compressed 181, 186, 191
Blocking dialog 594
BmpCvt.exe 188–190
Bottom window 247
Button widget 300, 309–322
BUTTON_3D_MOVE_X 309

© 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

BUTTON_3D_MOVE_Y 309
BUTTON_ALIGN_DEFAULT 309
BUTTON_BI_DISABLED 312, 316
BUTTON_BI_PRESSED 312, 316
BUTTON_BI_UNPRESSED 312, 316
BUTTON_BKCOLOR0_DEFAULT 309
BUTTON_BKCOLOR1_DEFAULT 309
BUTTON_CI_DISABLED 313–319
BUTTON_CI_PRESSED 313–319
BUTTON_CI_UNPRESSED 313–319
BUTTON_Create 311
BUTTON_CreateEx 312
BUTTON_CreateIndirect 312
BUTTON_FOCUSCOLOR_DEFAULT 309
BUTTON_FONT_DEFAULT 309
BUTTON_GetBitmap 312
BUTTON_GetBkColor 313
BUTTON_GetDefaultBkColor 313
BUTTON_GetDefaultFont 313
BUTTON_GetDefaultTextAlign 314
BUTTON_GetDefaultTextColor 314
BUTTON_GetFont 314
BUTTON_GetText 314
BUTTON_GetTextColor 315
BUTTON_IsPressed 315
BUTTON_REACT_ON_LEVEL 309
BUTTON_SetBitmap 315
BUTTON_SetBitmapEx 316
BUTTON_SetBkColor 316
BUTTON_SetBMP 317
BUTTON_SetBMPEx 317
BUTTON_SetDefaultBkColor 317
BUTTON_SetDefaultFocusColor 318
BUTTON_SetDefaultFont 318
BUTTON_SetDefaultTextAlign 318
BUTTON_SetDefaultTextColor 319
BUTTON_SetFocusColor 319
BUTTON_SetFocussable 319
BUTTON_SetFont 320
BUTTON_SetPressed 320
BUTTON_SetStreamedBitmap 320
BUTTON_SetStreamedBitmapEx 320
BUTTON_SetText 321
BUTTON_SetTextAlign 321
BUTTON_SetTextColor 321
BUTTON_TEXTCOLOR0_DEFAULT 309
BUTTON_TEXTCOLOR1_DEFAULT 309

C

"C" compiler 31, 181
"C" files
  converting bitmaps into 179–185
  converting fonts into 156
  inclusion of in emWin 27
"C" programming language 17
Callback mechanism 19, 247–257
Callback routines 43, 246
  using 248–257
Character sets 153–155
Check box widget 300, 323–336
CHECKBOX_ALIGN_DEFAULT 323
CHECKBOX_BKCOLOR_DEFAULT 323
CHECKBOX_BKCOLOR0_DEFAULT 323
CHECKBOX_BKCOLOR1_DEFAULT 323
CHECKBOX_Check 325
CHECKBOX_Create 325
CHECKBOX_CreateEx 325
CHECKBOX_CreateIndirect 325
CHECKBOX_FGCOLOR0_DEFAULT 323
CHECKBOX_FGCOLOR1_DEFAULT 323
CHECKBOX_FOCUSCOLOR_DEFAULT 323
CHECKBOX_FONT_DEFAULT 323
CHECKBOX_GetDefaultBkColor 326
CHECKBOX_GetDefaultFont 326
CHECKBOX_GetDefaultSpacing 327
CHECKBOX_GetDefaultTextAlign 327
CHECKBOX_GetDefaultTextColor 327
CHECKBOX_GetState 327
CHECKBOX_GetText 328
CHECKBOX_IMAGE0_DEFAULT 323
CHECKBOX_IMAGE1_DEFAULT 323
CHECKBOX_IsChecked 328
CHECKBOX_SetBkColor 328
CHECKBOX_SetBoxBkColor 329
CHECKBOX_SetDefaultBkColor 329
CHECKBOX_SetDefaultFocusColor 330
CHECKBOX_SetDefaultFont 330
CHECKBOX_SetDefaultImage 330
CHECKBOX_SetDefaultSpacing 331
CHECKBOX_SetDefaultTextAlign 331
CHECKBOX_SetDefaultTextColor 331
CHECKBOX_SetFocusColor 332
CHECKBOX_SetImage 332
CHECKBOX_SetNumStates 333
CHECKBOX_SetSpacing 333
CHECKBOX_SetState 333
CHECKBOX_SetText 334
CHECKBOX_SetTextAlign 334
CHECKBOX_SetTextColor 335
CHECKBOX_SPACING_DEFAULT 323
CHECKBOX_TEXTCOLOR_DEFAULT 323
CHECKBOX_Uncheck 335
Child window 246, 265, 594
Circles, drawing 104–106
Client area, of windows 246
Clip area, of windows 246
Clipping 85, 246
Color bar test routine 194–195
Color conversion, of bitmaps . 180, 185–186
Color palettes
  best palette option 185, 188, 190
  custom 186–187, 206
  fixed 185, 195–204
Colors 193
  converting 193
  logical 193
  physical 193
  predefined 194
Command line usage
  of bitmap converter 188–190
Compile time switches 19
Compiling, with simulator
  demo program 33–34
  for your application 34
  samples 33
Config folder 27, 34, 723
Configuration, of emWin 27
Control characters 62, 137
Controls (see Widgets)
Coordinates 20, 246, 655–656
  high-resolution 653, 655–656
Cursors 649–652
  available styles 650

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

Custom palettes
defining for hardware 206
file formats, for color conversion 187
for color conversion 186–187

D

Data types 21
Decimal values, displaying 76–79
Demos 19
Depth coordinate 247
Desktop coordinates 246
Desktop window 246
Device simulation ??– 42
Device.bmp 37, 43
Device1.bmp 37, 43
Device-dependent bitmap (DDB) 181
Device-independent bitmap (DIB) 181
Dialog messages 594
Dialog procedure 594–597
Dialogs 593–599
blocking 594
creating 594–597
defining behavior of 597
initialization of 596–597
messages sent to 594
non-blocking 594
Directories, inclusion of 24
Directory structure
for emWin 24
for simulator 34
for Visual C++ workspace 34
Display driver 681–710
Displaying bitmap files 113–135
Drawing modes 87–88
Dropdown widget 300, 337–349
DROPDOWN_AddString 339
DROPDOWN_ALIGN_DEFAULT 337
DROPDOWN_BKCOLOR0_DEFAULT 337
DROPDOWN_BKCOLOR1_DEFAULT 337
DROPDOWN_BKCOLOR2_DEFAULT 337
DROPDOWN_CF_AUTOSCROLLBAR 340
DROPDOWN_CF_UP 340
DROPDOWN_Collapse 339
DROPDOWN_Create 339
DROPDOWN_CreateEx 340
DROPDOWN_DecSel 340
DROPDOWN_DecSelExp 341
DROPDOWN_Expand 341
DROPDOWN_FONT_DEFAULT 337
DROPDOWN_GetItemDisabled 341
DROPDOWN_GetListbox 342
DROPDOWN_GetNumItems 342
DROPDOWN_GetSel 342
DROPDOWN_GetSelExp 342
DROPDOWN_IncSel 343
DROPDOWN_IncSelExp 343
DROPDOWN_InsertString 343
DROPDOWN_KEY_EXPAND 337
DROPDOWN_KEY_SELECT 337
DROPDOWN_SetAutoScroll 343
DROPDOWN_SetBkColor 344, 346, 448
DROPDOWN_SetColor 344
DROPDOWN_SetDefaultColor 345
DROPDOWN_SetDefaultFont 345
DROPDOWN_SetDefaultScrollbarColor 345
DROPDOWN_SetFont 345
DROPDOWN_SetItemDisabled 346
DROPDOWN_SetItemSpacing 348
DROPDOWN_SetScrollbarWidth 347
DROPDOWN_SetSel 347
DROPDOWN_SetSelExp 347
DROPDOWN_SetTextAlign 348
DROPDOWN_SetTextColor 348
DROPDOWN_SetTextHeight 349
DROPDOWN_TEXTCOLOR0_DEFAULT 337
DROPDOWN_TEXTCOLOR1_DEFAULT 337
DROPDOWN_TEXTCOLOR2_DEFAULT 337
DROPROWN_DeleteItem 341

E

Edit widget 300, 350–366
EDIT_AddKey 352, 514–519
EDIT_ALIGN_DEFAULT 350
EDIT_BKCOLOR0_DEFAULT 350
EDIT_BKCOLOR1_DEFAULT 350
EDIT_BORDER_DEFAULT 350
EDIT_Create 352
EDIT_CreateAsChild 353
EDIT_CreateEx 353
EDIT_CreateIndirect 354
EDIT_EnableBlink 354
EDIT_FONT_DEFAULT 350
EDIT_GetCursorCharPos 354
EDIT_GetCursorPixelPos 354
EDIT_GetDefaultBkColor 355
EDIT_GetDefaultFont 355
EDIT_GetDefaultTextAlign 355
EDIT_GetDefaultTextColor 356
EDIT_GetFloatValue 356
EDIT_GetNumChars 356
EDIT_GetText 356
EDIT_GetValue 357
EDIT_SetBinMode 357
EDIT_SetBkColor 357
EDIT_SetCursorAtChar 358
EDIT_SetCursorAtPixel 358
EDIT_SetDecMode 358
EDIT_SetDefaultBkColor 359
EDIT_SetDefaultFont 359
EDIT_SetDefaultTextAlign 359
EDIT_SetDefaultTextColor 359
EDIT_SetFloatMode 359
EDIT_SetFloatValue 360
EDIT_SetFont 360
EDIT_SetHexMode 360
EDIT_SetInsertMode 361
EDIT_SetMaxLen 361
EDIT_SetpfAddKeyEx 361
EDIT_SetSel 362
EDIT_SetText 362
EDIT_SetTextAlign 363
EDIT_SetTextColor 363
EDIT_SetTextMode 363
EDIT_SetUlongMode 363
EDIT_SetValue 364
EDIT_TEXTCOLOR0_DEFAULT 350
EDIT_TEXTCOLOR1_DEFAULT 350
EDIT_XOFF 350
Ellipses, drawing 106–107
embOS 235
kernel interface routines for 242
emWin

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

as trial version 32–33
configuration of 27
data types used (see Data types)
directory structure for 24
Driver benchmark 736–737
features of 18–19
in multitask environments 28, 301
initialization of 28
memory requirements 738
updating to newer versions 24
Execution model 235–243
supported types 236

F

Fixed color palettes 185
Fixed palette modes 195–204
Flickering of display 211, 289
Floating-point calculations 85
Floating-point values, displaying 80–82
Font converter 19, 139, 156, 655, 678
Font editor 156
Font files
linking 139, 156
naming convention 158
Fonts 19, 137–178
adding 156
antialiased 138, 653–655
converting (see Font converter)
creating additional 139
declaring 139, 156
default 142
defining 19
Digit fonts (monospaced) 177–178
Digit fonts (proportional) 175–176
editing 156
External Bitmap Fonts (XBF) 140
file naming convention 158
generating "C" files from 156
included with emWin 19, 137
monospaced 138, 157, 170–174
naming convention 157–158
proportional 138, 157, 159–169
scaling 19
selecting 142–143
System Independent Fonts (SIF) 140
usage of 139
Foreign Language Support 667–680
Frame window widget 300, 367–390
FRAMEWIN_AddButton 370
FRAMEWIN_AddCloseButton 371
FRAMEWIN_AddMaxButton 371
FRAMEWIN_AddMenu 372, 386
FRAMEWIN_AddMinButton 373
FRAMEWIN_ALLOW_DRAG_ON_FRAME . 369
FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT .. 369
FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT 369
FRAMEWIN_BORDER_DEFAULT 369
FRAMEWIN_CLIENTCOLOR_DEFAULT ... 369
FRAMEWIN_Create 373
FRAMEWIN_CreateAsChild 374
FRAMEWIN_CreateEx 374
FRAMEWIN_DEFAULT_FONT 369
FRAMEWIN_FRAMECOLOR_DEFAULT 369
FRAMEWIN_GetActive 375
FRAMEWIN_GetBarColor 375
FRAMEWIN_GetBorderSize 376
FRAMEWIN_GetDefaultBarColor 376
FRAMEWIN_GetDefaultBorderSize 376
FRAMEWIN_GetDefaultClientColor 377
FRAMEWIN_GetDefaultFont 377
FRAMEWIN_GetDefaultTextColor ... 377, 384
FRAMEWIN_GetDefaultTitleSize 378
FRAMEWIN_GetFont 377
FRAMEWIN_GetText 378
FRAMEWIN_GetTextAlign 378
FRAMEWIN_GetTitleHeight 378
FRAMEWIN_IBORDER_DEFAULT 369
FRAMEWIN_IsMaximized 379
FRAMEWIN_IsMinimized 379
FRAMEWIN_Maximize 379
FRAMEWIN_Minimize 380
FRAMEWIN_OwnerDraw 380
FRAMEWIN_Restore 381
FRAMEWIN_SetActive 381
FRAMEWIN_SetBarColor 382
FRAMEWIN_SetBorderSize 382
FRAMEWIN_SetClientColor 383
FRAMEWIN_SetDefaultBarColor 383
FRAMEWIN_SetDefaultBorderSize 383
FRAMEWIN_SetDefaultClient Color 383
FRAMEWIN_SetDefaultFont 384
FRAMEWIN_SetDefaultTitleHeight 384
FRAMEWIN_SetFont 385
FRAMEWIN_SetMoveable 385
FRAMEWIN_SetOwnerDraw 385
FRAMEWIN_SetText 387
FRAMEWIN_SetTextAlign 387
FRAMEWIN_SetTextColor 388
FRAMEWIN_SetTextColorEx 388
FRAMEWIN_SetTitleHeight 389
FRAMEWIN_SetTitleVis 389
FRAMEWIN_TITLEHEIGHT_DEFAULT 369
Full-color mode, of bitmaps 185
Function replacement macro 28
Function-level linking 24

G

GIF file support 124, 132
Graph widget 300, 391–414
GRAPH_AttachData 394
GRAPH_AttachScale 395
GRAPH_CreateEx 395
GRAPH_DATA_XY_AddPoint 406
GRAPH_DATA_XY_Create 406
GRAPH_DATA_XY_Delete 407
GRAPH_DATA_XY_SetLineStyle 408
GRAPH_DATA_XY_SetOffX 407
GRAPH_DATA_XY_SetPenSize 408
GRAPH_DATA_YT_AddValue 403
GRAPH_DATA_YT_Create 403
GRAPH_DATA_YT_Delete 404
GRAPH_DATA_YT_MirrorX 404
GRAPH_DATA_YT_SetAlign 405
GRAPH_DATA_YT_SetOffY 405
GRAPH_DetachData 396
GRAPH_SCALE_Create 409
GRAPH_SCALE_Delete 410
GRAPH_SCALE_SetFactor 410
GRAPH_SCALE_SetFont 411
GRAPH_SCALE_SetNumDecs 411

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

GRAPH_SCALE_SetOff 412
GRAPH_SCALE_SetPos 412
GRAPH_SCALE_SetTextColor 413
GRAPH_SCALE_SetTickDist 413
GRAPH_SetBorder 397
GRAPH_SetColor 397
GRAPH_SetGridFixedX 398
GRAPH_SetGridOffY 399
GRAPH_SetGridSpacingX 398
GRAPH_SetGridSpacingY 398
GRAPH_SetGridVis 399
GRAPH_SetLineStyleH 400
GRAPH_SetLineStyleV 400
GRAPH_SetUserDraw 402
GRAPH_SetVSizeX 401
GRAPH_SetVSizeY 401
Graphic library 19, 85–111, 719
Grayscales 185, 193
GUI subdirectories 24
GUI_AA_DisableHiRes 656
GUI_AA_DrawArc 657
GUI_AA_DrawLine 658
GUI_AA_DrawPolyOutline 658
GUI_AA_DrawPolyOutlineEx 659
GUI_AA_EnableHiRes 656
GUI_AA_FillCircle 659
GUI_AA_FillPolygon 660
GUI_AA_GetFactor 657
GUI_AA_SetFactor 657
GUI_ALLOC_AssignMemory 725
GUI_ALLOC_SetAvBlockSize 726
GUI_AssignCursorLayer 618
GUI_AUTODEV 231
GUI_AUTODEV_INFO 232
GUI_BITMAP structures 180
GUI_BMP_Draw 114
GUI_BMP_DrawEx 115
GUI_BMP_DrawScaled 115
GUI_BMP_DrawScaledEx 116
GUI_BMP_GetXSize 116
GUI_BMP_GetXSizeEx 116
GUI_BMP_GetYSize 117
GUI_BMP_GetYSizeEx 117
GUI_BMP_Serialize 117
GUI_BMP_SerializeEx 118
GUI_CalcColorDist 209
GUI_CalcVisColorError 209
GUI_Clear 73
GUI_ClearKeyBuffer 641
GUI_ClearRect 89
GUI_Color2Index 210
GUI_Color2VisColor 210
GUI_ColorIsAvailable 210
GUI_CreateDialogBox 594, 598
GUI_CURSOR_GetState 650
GUI_CURSOR_Hide 651
GUI_CURSOR_Select 651
GUI_CURSOR_SetPosition 651
GUI_CURSOR_Show 652
GUI_DEBUG_LEVEL 732
GUI_DEFAULT_BKCOLOR 731
GUI_DEFAULT_COLOR 731
GUI_DEFAULT_FONT 731
GUI_Delay 719–720
GUI_DEVICE_CreateAndLink 728
GUI_DispBin 82
GUI_DispBinAt 83
GUI_DispCEOL 73
GUI_DispChar 63
GUI_DispCharAt 64
GUI_DispChars 64
GUI_DispDec 76
GUI_DispDecAt 77
GUI_DispDecMin 77
GUI_DispDecShift 78
GUI_DispDecSpace 78
GUI_DispFloat 80
GUI_DispFloatFix 81
GUI_DispFloatMin 81
GUI_DispHex 83
GUI_DispHexAt 84
GUI_DispNextLine 64
GUI_DispSDec 78
GUI_DispSDecShift 79
GUI_DispSFloatFix 82
GUI_DispSFloatMin 82
GUI_DispString 65
GUI_DispStringAt 65
GUI_DispStringAtCEOL 65
GUI_DispStringHCenterAt 66
GUI_DispStringInRect 66
GUI_DispStringInRectEx 66
GUI_DispStringInRectWrap 67
GUI_DispStringLen 68
GUI_DrawArc 107
GUI_DrawBitmap 94
GUI_DrawBitmapEx 95
GUI_DrawBitmapHWAlpha 96
GUI_DrawBitmapMag 96
GUI_DrawCircle 104
GUI_DrawEllipse 106
GUI_DrawGradientH 90
GUI_DrawGraph 108
GUI_DrawHLine 97
GUI_DrawLine 97
GUI_DrawLineRel 98
GUI_DrawLineTo 98
GUI_DRAWMODE_XOR 87
GUI_DrawPie 109
GUI_DrawPixel 91
GUI_DrawPoint 91
GUI_DrawPolygon 100
GUI_DrawPolyLine 98
GUI_DrawRect 91–92
GUI_DrawRoundedRect 92
GUI_DrawStreamedBitmap 97
GUI_DrawVLine 99
GUI_EditBin 364
GUI_EditDec 364
GUI_EditHex 365
GUI_EditString 365
GUI_EndDialog 599
GUI_EnlargePolygon 100
GUI_Exec 720
GUI_Exec1 720
GUI_ExecCreatedDialog 598
GUI_ExecDialogBox 594, 598
GUI_FillCircle 105
GUI_FillEllipse 106
GUI_FillPolygon 101
GUI_FillRect 92
GUI_FillRectEx 92
GUI_FillRoundedRect 93
GUI_FONT structures 156

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

GUI_GetBkColor 207
GUI_GetBkColorIndex 207
GUI_GetCharDistX 149
GUI_GetClientRect 88
GUI_GetColor 208
GUI_GetColorIndex 208
GUI_GetDispPosX 73
GUI_GetDispPosY 73
GUI_GetDrawMode 87
GUI_GetFont 149
GUI_GetFontDistY 150
GUI_GetFontInfo 150
GUI_GetFontSizeY 150
GUI_GetKey 642
GUI_GetLayerPosEx 620
GUI_GetLeadingBlankCols 151
GUI_GetLineStyle 99
GUI_GetOrg 608
GUI_GetPenSize 89
GUI_GetStringDistX 151
GUI_GetTextAlign 71
GUI_GetTextExtend 151
GUI_GetTime 721
GUI_GetTrailingBlankCols 151
GUI_GetVersionString 84
GUI_GetYDistOfFont 152
GUI_GetYSizeOfFont 152
GUI_GIF_Draw 125
GUI_GIF_DrawEx 125
GUI_GIF_DrawSub 126
GUI_GIF_DrawSubEx 126
GUI_GIF_DrawSubScaled 127
GUI_GIF_DrawSubScaledEx 127
GUI_GIF_GetComment 128
GUI_GIF_GetCommentEx 128
GUI_GIF_GetImageInfo 128
GUI_GIF_GetImageInfoEx 129
GUI_GIF_GetInfo 129
GUI_GIF_GetInfoEx 130
GUI_GIF_GetXSize 130
GUI_GIF_GetXSizeEx 131
GUI_GIF_GetYSize 131
GUI_GIF_GetYSizeEx 131
GUI_GotoX 72
GUI_GotoXY 72
GUI_GotoY 72
GUI_Index2Color 210
GUI_InvertRect 93
GUI_IsInFont 152
GUI_JPEG_Draw 120
GUI_JPEG_DrawEx 121
GUI_JPEG_DrawScaled 121
GUI_JPEG_DrawScaledEx 122
GUI_JPEG_GetInfo 122
GUI_JPEG_GetInfoEx 123
GUI_MagnifyPolygon 102
GUI_MAXTASK 239, 732
GUI_MEASDEV_ClearRect 233
GUI_MEASDEV_Create 233
GUI_MEASDEV_Delete 234
GUI_MEASDEV_GetRect 234
GUI_MEASDEV_Select 234
GUI_MEMCPY 732
GUI_MEMDEV_Clear 216
GUI_MEMDEV_CopyFromLCD 216
GUI_MEMDEV_CopyToLCD 217
GUI_MEMDEV_CopyToLCDAA 217
GUI_MEMDEV_CopyToLCDAt 217
GUI_MEMDEV_Create 218
GUI_MEMDEV_CreateAuto 231
GUI_MEMDEV_CreateFixed 219
GUI_MEMDEV_Delete 220
GUI_MEMDEV_DeleteAuto 231
GUI_MEMDEV_Draw 230
GUI_MEMDEV_DrawAuto 232
GUI_MEMDEV_DrawPerspectiveX 220
GUI_MEMDEV_GetDataPtr 222
GUI_MEMDEV_GetYSize 223
GUI_MEMDEV_MarkDirty 223
GUI_MEMDEV_ReduceYSize 223
GUI_MEMDEV_Rotate 224
GUI_MEMDEV_RotateHQ 224
GUI_MEMDEV_Select 226
GUI_MEMDEV_SetOrg 226
GUI_MEMDEV_Write 226
GUI_MEMDEV_WriteAlpha 227
GUI_MEMDEV_WriteAlphaAt 227
GUI_MEMDEV_WriteAt 227
GUI_MEMDEV_WriteEx 228
GUI_MEMSET 732
GUI_MessageBox 507
GUI_MOUSE_DRIVER_PS2_Init 629
GUI_MOUSE_DRIVER_PS2_OnRx 629
GUI_MOUSE_GetState 627
GUI_MoveRel 99
GUI_NUM_LAYERS 732
GUI_OS 239, 731
GUI_PID_GetState 625
GUI_PID_STATE 625
GUI_PID_StoreState 625
GUI_PNG_Draw 132
GUI_PNG_DrawEx 133
GUI_PNG_GetXSize 133
GUI_PNG_GetXSizeEx 134
GUI_PNG_GetYSize 134
GUI_PNG_GetYSizeEx 134
GUI_RestoreContext 110
GUI_RotatePolygon 103
GUI_SaveContext 110
GUI_SelectLCD 229
GUI_SelLayer 619
GUI_SendKeyMsg 641
GUI_SetAlpha 93
GUI_SetBkColor 208
GUI_SetBkColorIndex 208
GUI_SetClipRect 110
GUI_SetColor 208
GUI_SetColorIndex 209
GUI_SetDrawMode 88
GUI_SetFont 143
GUI_SetLayerAlphaEx 619
GUI_SetLayerSizeEx 620
GUI_SetLayerVisEx 620
GUI_SetLBorder 71
GUI_SetLineStyle 100
GUI_SetLUTColorEx 620
GUI_SetOrg 608
GUI_SetPenSize 89
GUI_SetTextAlign 71
GUI_SetTextMode 70
GUI_SetTextStyle 71
GUI_SIF_CreateFont 143
GUI_SIF_DeleteFont 144
GUI_SPRITE_Create 644

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

GUI_SPRITE_CreateEx 645
GUI_SPRITE_Delete 645
GUI_SPRITE_GetState 645
GUI_SPRITE_Hide 646
GUI_SPRITE_SetBitmap 646
GUI_SPRITE_SetBitmapAndPosition 646
GUI_SPRITE_SetPosition 647
GUI_StoreKey 642
GUI_StoreKeyMsg 640
GUI_SUPPORT_ARABIC 731
GUI_SUPPORT_CURSOR 731
GUI_SUPPORT_LARGE_BITMAPS 732
GUI_SUPPORT_MEMDEV 731
GUI_SUPPORT_MOUSE 731
GUI_SUPPORT_ROTATION 731
GUI_SUPPORT_SPRITE 731
GUI_SUPPORT_TOUCH 731
GUI_SUPPORT_UNICODE 732
GUI_TEXTMODE_NORMAL 70, 72
GUI_TEXTMODE_REVERSE 70, 72
GUI_TEXTMODE_TRANSPARENT 70, 72
GUI_TEXTMODE_XOR 70, 72
GUI_TOUCH_Calibrate 635
GUI_TOUCH_Exec 636
GUI_TOUCH_GetState 630
GUI_TOUCH_SetDefaultCalibration 636
GUI_TOUCH_SetOrientation 636
GUI_TOUCH_StoreState 627, 630
GUI_TOUCH_StoreStateEx 631
GUI_TOUCH_X_ActivateX 634
GUI_TOUCH_X_ActivateY 634
GUI_TOUCH_X_MeasureX 634
GUI_TOUCH_X_MeasureY 634
GUI_TRIAL_VERSION 732
GUI_TTF_CreateFont 145
GUI_TTF_DestroyCache 146
GUI_TTF_Done 146
GUI_TTF_GetFamilyName 146
GUI_TTF_GetStyleName 147
GUI_TTF_SetCacheSize 147
GUI_UC_ConvertUC2UTF8 670
GUI_UC_ConvertUTF82UC 670
GUI_UC_DispString 672
GUI_UC_Encode 671
GUI_UC_GetCharCode 671
GUI_UC_GetCharSize 671
GUI_UC_SetEncodeNone 672
GUI_UC_SetEncodeUTF8 672
GUI_VNC_AttachToLayer 715
GUI_VNC_Process 716
GUI_VNC_X_StartServer 717
GUI_WaitKey 642
GUI_WINSUPPORT 731–732
GUI_X_Config 725
GUI_X_Delay 720, 729
GUI_X_ExecIdle 729
GUI_X_GetTaskID 241
GUI_X_GetTime 721, 729
GUI_X_InitOS 241
GUI_X_Lock 241
GUI_X_Log 730
GUI_X_SIGNAL_EVENT 239
GUI_X_SignalEvent 241
GUI_X_Unlock 242
GUI_X_WAIT_EVENT 240
GUI_X_WaitEvent 242
GUI_XBF_CreateFont 148
GUI_XBF_DeleteFont 149
GUIConf.h 142, 214
GUIDRV_CompactColor_16 driver ..699–702
GUIDRV_Lin driver 696–698

H

Handle, of a window 246
Hardkey simulation 43–46
Header widget 300, 415–427
HEADER_AddItem 416
HEADER_Create 417
HEADER_CreateAttached 417
HEADER_CreateEx 418, 457
HEADER_CreateIndirect 418
HEADER_GetDefaultBkColor 418
HEADER_GetDefaultBorderH 419
HEADER_GetDefaultBorderV 419
HEADER_GetDefaultCursor 419
HEADER_GetDefaultFont 419
HEADER_GetDefaultTextColor 419
HEADER_GetHeight 420
HEADER_GetItemWidth 420
HEADER_GetNumColumns 420
HEADER_SetBitmap 420
HEADER_SetBitmapEx 421
HEADER_SetBkColor 421
HEADER_SetBMP 422
HEADER_SetBMPEx 422
HEADER_SetDefaultBkColor 422
HEADER_SetDefaultBorderH 423
HEADER_SetDefaultBorderV 423
HEADER_SetDefaultCursor 423
HEADER_SetDefaultFont 424
HEADER_SetDefaultTextColor 424
HEADER_SetDragLimit 424
HEADER_SetFont 424
HEADER_SetHeight 425
HEADER_SetItemText 425
HEADER_SetItemWidth 426
HEADER_SetStreamedBitmap 426
HEADER_SetStreamedBitmapEx 426
HEADER_SetTextAlign 425
HEADER_SetTextColor 427
Hello world program 29
Hexadecimal values, displaying 83–84
Hiding windows 247
High-resolution coordinates .. 653, 655–656

I

I/O pins, connection to 689
Icon view widget 428–433
ICONVIEW 300
ICONVIEW_AddBitmapItem 429
ICONVIEW_ALIGN_DEFAULT 428
ICONVIEW_BKCOLOR0_DEFAULT 428
ICONVIEW_BKCOLOR1_DEFAULT 428
ICONVIEW_CreateEx 430
ICONVIEW_FONT_DEFAULT 428
ICONVIEW_FRAMEX_DEFAULT 428
ICONVIEW_FRAMEY_DEFAULT 428
ICONVIEW_GetSel 431
ICONVIEW_SetBkColor 431
ICONVIEW_SetFont 432
ICONVIEW_SetSel 432
ICONVIEW_SetTextColor 433
ICONVIEW_SPACEX_DEFAULT 428

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

ICONVIEW_SPACEY_DEFAULT 428
ICONVIEW_TEXTCOLOR0_DEFAULT 428
ICONVIEW_TEXTCOLOR1_DEFAULT 428
Initializing emWin 28
Input devices 623, 639–642
  keyboard ??– 642
  mouse ??– 629
Input focus 594
Interrupt service routines 236–238, 636
Invalidation, of windows 247
ISO 8859-1 ..... 137, 153, 155

J

JPEG file support 119

K

Kernel interface routines 236–238, 240–243
Keyboard input support ??– 642

L

LCD
  caching in memory 19
  configuration of 193
  connecting to microcontroller 21
  without LCD controller 21
LCD controller
  connected to port/buffer 21
  memory-mapped 21
  support for 21, 682
LCD driver 32
  availability/selection 682
  customization of 21
LCD_CONTROLLER 700
LCD_ENDIAN_BIG 698
LCD_GetBitsPerPixel 706
LCD_GetBitsPerPixelEx 706
LCD_GetFixedPalette 706
LCD_GetFixedPaletteEx 706
LCD_GetNumColors 707
LCD_GetNumColorsEx 707
LCD_GetNumLayers 621
LCD_GetVXSize 707
LCD_GetVXSizeEx 707
LCD_GetVYSize 707
LCD_GetVYSizeEx 707
LCD_GetXMagEx 708
LCD_GetXSize 708
LCD_GetXSizeEx 708
LCD_GetYMagEx 708
LCD_GetYSize 708
LCD_GetYSizeEx 708
LCD_NUM_DUMMY_READS 701
LCD_READ_A0 686
LCD_READ_A1 686
LCD_READM_A1 701
LCD_REG01 701
LCD_SERIAL_ID 701
LCD_SetSizeEx 709
LCD_SetVRAMAddrEx 709
LCD_SetVSizeEx 603, 709
LCD_USE_PARALLEL_16 701
LCD_USE_SERIAL_3PIN 701
LCD_WRITE 689
LCD_WRITE_A0 ..... 687–688, 701
LCD_WRITE_A1 ..... 687–688, 701
LCD_WRITE_BUFFER_SIZE 701
LCD_WRITEM 690
LCD_WRITEM_A0 701
LCD_WRITEM_A1 ..... 687, 689, 701
LCD_X_Config 727
LCD_X_DisplayDriver 694, 727
LCD_X_InitController 705
LCDConf.h 21
LCDDummy driver 703
LCDNull driver 704
LCDTemplate driver ??– 703
Library, creating 25
Lines, drawing 97–99
Linking source files 24
List box widget 300, 434–451
LISTBOX_ 444
LISTBOX_AddString 436
LISTBOX_BKCOLOR0_DEFAULT 434
LISTBOX_BKCOLOR1_DEFAULT 434
LISTBOX_BKCOLOR2_DEFAULT 434
LISTBOX_Create 436
LISTBOX_CreateAsChild 436
LISTBOX_CreateEx 437
LISTBOX_CreateIndirect 437
LISTBOX_DecSel 438
LISTBOX_DeleteItem 438
LISTBOX_FONT_DEFAULT 434
LISTBOX_GetDefaultBkColor 438
LISTBOX_GetDefaultFont 438
LISTBOX_GetDefaultScrollStepH 439
LISTBOX_GetDefaultTextAlign 439
LISTBOX_GetDefaultTextColor 439
LISTBOX_GetFont 439
LISTBOX_GetItemDisabled 440
LISTBOX_GetItemSel 440
LISTBOX_GetItemText 440
LISTBOX_GetMulti 441
LISTBOX_GetNumItems 441
LISTBOX_GetScrollStepH 441
LISTBOX_GetSel 441
LISTBOX_GetTextAlign 442
LISTBOX_IncSel 442
LISTBOX_InsertString 442
LISTBOX_InvalidateItem 442
LISTBOX_OwnerDraw 443
LISTBOX_SetAutoScrollH 443
LISTBOX_SetBkColor 444
LISTBOX_SetDefaultBkColor 444
LISTBOX_SetDefaultFont 445
LISTBOX_SetDefaultScrollStepH 445
LISTBOX_SetDefaultTextColor 445
LISTBOX_SetFont 446
LISTBOX_SetItemDisabled 446
LISTBOX_SetItemSel 446
LISTBOX_SetItemSpacing 447
LISTBOX_SetMulti 447
LISTBOX_SetOwnerDraw 447
LISTBOX_SetScrollbarWidth 449
LISTBOX_SetScrollStepH 449
LISTBOX_SetSel 449
LISTBOX_SetString 449
LISTBOX_SetTextAlign 450
LISTBOX_SetTextColor 450
LISTBOX_TEXTCOLOR0_DEFAULT 434
LISTBOX_TEXTCOLOR1_DEFAULT 434
LISTBOX_TEXTCOLOR2_DEFAULT 434
Listview widget 300, 452–558

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

LISTVIEW_ 459–461
LISTVIEW_AddColumn 455
LISTVIEW_AddRow 455
LISTVIEW_CompareDec 456
LISTVIEW_CompareText 456
LISTVIEW_Create 457
LISTVIEW_CreateAttached 457
LISTVIEW_CreateIndirect 458
LISTVIEW_DecSel 458
LISTVIEW_DeleteColumn 458
LISTVIEW_DeleteRow 459
LISTVIEW_DisableSort 459
LISTVIEW_EnableSort 460
LISTVIEW_GetBkColor 460
LISTVIEW_GetFont 461
LISTVIEW_GetHeader 461
LISTVIEW_GetNumColumns 462
LISTVIEW_GetNumRows 462
LISTVIEW_GetSel 462
LISTVIEW_GetSelUnsorted 463
LISTVIEW_GetTextColor 463
LISTVIEW_GetUserData 463
LISTVIEW_IncSel 464
LISTVIEW_InsertRow 464
LISTVIEW_SetAutoScrollH 464
LISTVIEW_SetAutoScrollV 465
LISTVIEW_SetBkColor 465
LISTVIEW_SetColumnWidth 466
LISTVIEW_SetCompareFunc 466
LISTVIEW_SetDefaultBkColor 467
LISTVIEW_SetDefaultFont 467
LISTVIEW_SetDefaultGridColor 468
LISTVIEW_SetDefaultTextColor 468
LISTVIEW_SetFixed 468
LISTVIEW_SetFont 468
LISTVIEW_SetGridVis 469
LISTVIEW_SetItemBkColor 469
LISTVIEW_SetItemText 470
LISTVIEW_SetItemTextColor 470
LISTVIEW_SetLBorder 471
LISTVIEW_SetRBorder 471
LISTVIEW_SetRowHeight 472
LISTVIEW_SetSel 472
LISTVIEW_SetSelUnsorted 472
LISTVIEW_SetSort 473
LISTVIEW_SetTextAlign 473
LISTVIEW_SetTextColor 474
LISTVIEW_SetUserData 474
LISTWHEEL 300
Listwheel widget 476
LISTWHEEL_AddString 477
LISTWHEEL_CreateEx 478
LISTWHEEL_CreateIndirect 478
LISTWHEEL_GetFont 478
LISTWHEEL_GetItemText 479
LISTWHEEL_GetLBorder 479
LISTWHEEL_GetLineHeight 479
LISTWHEEL_GetPos 480
LISTWHEEL_GetRBorder 480
LISTWHEEL_GetSel 481
LISTWHEEL_GetSnapPosition 481
LISTWHEEL_GetTextAlign 481
LISTWHEEL_GetUserData 481
LISTWHEEL_MoveToPos 482
LISTWHEEL_OwnerDraw 482
LISTWHEEL_SetFont 483
LISTWHEEL_SetLBorder 483
LISTWHEEL_SetLineHeight 484
LISTWHEEL_SetOwnerDraw 484
LISTWHEEL_SetRBorder 485
LISTWHEEL_SetSel 485
LISTWHEEL_SetSnapPosition 486
LISTWHEEL_SetText 486
LISTWHEEL_SetTextAlign 487
LISTWHEEL_SetTextColor 487
LISTWHEEL_SetUserData 488
Lookup table (LUT) 204, 705

M

Measurement device object 233–234
Memory devices 211–297
  auto 231–233
  banding 229–231
  basic usage of 214
  disabling of 214, 290
  software for 211
  with window manager 289
Memory, reducing consumption of 180, 185
MENU 301
Menu widget 489–506
MENU_AddItem 493
MENU_Attach 494
MENU_CreateEx 494
MENU_DeleteItem 495
MENU_DisableItem 496
MENU_EnableItem 496
MENU_GetDefaultBkColor 497
MENU_GetDefaultBorderSize 497
MENU_GetDefaultEffect 497
MENU_GetDefaultFont 498
MENU_GetDefaultTextColor 498
MENU_GetItem 498
MENU_GetItemText 499
MENU_GetNumItems 499
MENU_GetOwner 499
MENU_IF_DISABLED 491
MENU_IF_SEPARATOR 491
MENU_InsertItem 500
MENU_MSG_DATA 490, 728
MENU_ON_INITMENU 490
MENU_ON_ITEMACTIVATE 490
MENU_ON_ITEMPRESSED 490
MENU_ON_ITEMSELECT 490
MENU_Popup 500
MENU_SetBkColor 501
MENU_SetBorderSize 502
MENU_SetDefaultBkColor 502
MENU_SetDefaultBorderSize 502
MENU_SetDefaultEffect 503
MENU_SetDefaultFont 503
MENU_SetDefaultTextColor 503
MENU_SetFont 504
MENU_SetItem 504
MENU_SetOwner 505
MENU_SetSel 505
MENU_SetTextColor 506
Message box widget 507–508
MESSAGEBOX_Create 508
Messages, sent by callback routines 250
Monospaced fonts (see Fonts)
Mouse support ??– 629
MULTIEDIT 301
Multiedit widget 509–520

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

MULTIEDIT_AddKey 511
MULTIEDIT_AddText 511
MULTIEDIT_Create 512
MULTIEDIT_CreateEx 512
MULTIEDIT_CreateIndirect 513
MULTIEDIT_EnableBlink 513
MULTIEDIT_GetCursorCharPos 513
MULTIEDIT_GetCursorPixelPos 514
MULTIEDIT_GetPrompt 514
MULTIEDIT_GetText 514
MULTIEDIT_GetTextSize 514
MULTIEDIT_SetAutoScrollH 515
MULTIEDIT_SetAutoScrollV 515
MULTIEDIT_SetBkColor 516
MULTIEDIT_SetBufferSize 516
MULTIEDIT_SetCursorOffset 516
MULTIEDIT_SetFont 516
MULTIEDIT_SetInsertMode 517
MULTIEDIT_SetMaxNumChars 517
MULTIEDIT_SetPasswordMode 517
MULTIEDIT_SetPrompt 518
MULTIEDIT_SetReadOnly 518
MULTIEDIT_SetText 518
MULTIEDIT_SetTextAlign 519
MULTIEDIT_SetTextColor 519
MULTIEDIT_SetWrapNone 519
MULTIEDIT_SetWrapWord 519
MULTIPAGE 301
Multipage widget 521–533
MULTIPAGE_AddPage 523
MULTIPAGE_CreateIndirect 524
MULTIPAGE_DeletePage 524
MULTIPAGE_DisablePage 525
MULTIPAGE_EnablePage 525
MULTIPAGE_GetDefaultAlign 525
MULTIPAGE_GetDefaultBkColor 526
MULTIPAGE_GetDefaultFont 527
MULTIPAGE_GetDefaultTextColor 527
MULTIPAGE_GetSelection 527
MULTIPAGE_GetWindow 527
MULTIPAGE_IsPageEnabled 528
MULTIPAGE_SelectPage 528
MULTIPAGE_SetAlign 529
MULTIPAGE_SetBkColor 529
MULTIPAGE_SetDefaultAlign 530
MULTIPAGE_SetDefaultBkColor 530
MULTIPAGE_SetDefaultFont 530
MULTIPAGE_SetDefaultTextColor 531
MULTIPAGE_SetFont 531
MULTIPAGE_SetRotation 532
MULTIPAGE_SetText 532
MULTIPAGE_SetTextColor 533
Multiple layer support 609–621
Multitask environments 236–239, 301
  multiple tasks call emWin ... 236, 238–239
  one task calls emWin 236–237

N

Non-blocking dialog 594
NORMAL drawing mode 87
Normal text 69
Numerical value macro 28

O

Optional software 27

P

Palettes (see Color palettes)
Parent window 246
Performance 735–741
Pixels 20
Pointer input device support
  data structure for 625
  mouse input ??– 629
Polygons, drawing 100–104
PROGBAR_Create 534
PROGBAR_CreateAsChild 535
PROGBAR_CreateEx 535
PROGBAR_CreateIndirect 536
PROGBAR_DEFAULT_BARCOLOR0 534
PROGBAR_DEFAULT_BARCOLOR1 534
PROGBAR_DEFAULT_FONT 534
PROGBAR_DEFAULT_TEXTCOLOR0 534
PROGBAR_DEFAULT_TEXTCOLOR1 534
PROGBAR_SetBarColor 536
PROGBAR_SetFont 536
PROGBAR_SetMinMax 536
PROGBAR_SetText 537
PROGBAR_SetTextAlign 537
PROGBAR_SetTextColor 537
PROGBAR_SetTextPos 537–538
PROGBAR_SetValue 538
Progress bar widget 301, 534–539
Proportional fonts (see Fonts)

R

Radio button widget 301, 540–550
RADIO_Create 541
RADIO_CreateEx 542
RADIO_CreateIndirect 542
RADIO_Dec 543
RADIO_GetDefaultFont 543
RADIO_GetDefaultTextColor 543
RADIO_GetText 544
RADIO_GetValue 544
RADIO_Inc 544
RADIO_SetBkColor 545
RADIO_SetDefaultFocusColor 545
RADIO_SetDefaultFont 546
RADIO_SetDefaultImage 546
RADIO_SetDefaultTextColor 546
RADIO_SetFocusColor 547
RADIO_SetFont 547
RADIO_SetGroupID 548
RADIO_SetImage 548
RADIO_SetText 549
RADIO_SetTextColor 549
RADIO_SetValue 550
Redrawing mechanism 301
Resource semaphore 240
Resource table, for dialogs 594–595
Resource usage 735–741
Reverse text 69
RLE compression, of bitmaps .181, 186, 191

S

Sample programs 19, 28
Scroll bar widget 301, 551–558
SCROLLBAR_AddValue 552
SCROLLBAR_COLOR_ARROW_DEFAULT 551
SCROLLBAR_COLOR_SHAFT_DEFAULT .. 551

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

SCROLLBAR_COLOR_THUMB_DEFAULT . 551
SCROLLBAR_Create 552
SCROLLBAR_CreateAttached 553
SCROLLBAR_CreateEx 554
SCROLLBAR_CreateIndirect 554
SCROLLBAR_Dec 554
SCROLLBAR_GetDefaultWidth 555
SCROLLBAR_GetValue 555
SCROLLBAR_Inc 555
SCROLLBAR_SetColor 555
SCROLLBAR_SetDefaultColor 556
SCROLLBAR_SetDefaultWidth 556
SCROLLBAR_SetNumItems 556
SCROLLBAR_SetPageSize 557
SCROLLBAR_SetState 557
SCROLLBAR_SetValue 557
SCROLLBAR_SetWidth 558
SCROLLBAR_THUMB_SIZE_MIN_DEFAULT .. 551
Selection switch macro 28
Shift JIS ??– 680
  creating fonts 678
Showing windows 247
Sibling window 246
SIM_GUI_CreateLCDInfoWindow() 51
SIM_GUI_CreateLCDWindow 51
SIM_GUI_Exit 52
SIM_GUI_Init 52
SIM_GUI_SetCompositeColor 39
SIM_GUI_SetCompositeSize 39
SIM_GUI_SetLCDColorBlack 40
SIM_GUI_SetLCDColorWhite 40
SIM_GUI_SetLCDPos 41
SIM_GUI_SetLCDWindowHook 52
SIM_GUI_SetMag 41
SIM_GUI_SetTransColor 42
SIM_GUI_UseCustomBitmaps 42
SIM_HARDKEY_GetNum 44
SIM_HARDKEY_GetState 44
SIM_HARDKEY_SetCallback 45
SIM_HARDKEY_SetMode 43, 45
SIM_HARDKEY_SetState 46
SIM_SetTransColor 37
Simulator 19, 31–53
  directory structure for 34
  usage of with emWin source 34–35
  usage of with emWin trial version .. 32–33
Single task system 236–237
Slider widget 301, 559–565
SLIDER_BKCOLOR0_DEFAULT 559
SLIDER_COLOR0_DEFAULT 559
SLIDER_Create 560
SLIDER_CreateEx 560
SLIDER_CreateIndirect 561
SLIDER_Dec 561
SLIDER_FOCUSCOLOR_DEFAULT 559
SLIDER_GetValue 561
SLIDER_Inc 561
SLIDER_SetBkColor 562
SLIDER_SetDefaultFocusColor 562
SLIDER_SetFocusColor 563
SLIDER_SetNumTicks 563
SLIDER_SetRange 563
SLIDER_SetValue 564
SLIDER_SetWidth 564
Source files, linking 24
Sprintf 75
Sprites 643–647
Standard fonts 137
Subdirectories, of GUI 24
Superloop 236–237
Support 743–748
Syntax, conventions used 20

T

Text
  alignment 71–72
  displaying 61–69
  modes 69–70
  positioning 62, 72–73
Text widget 301, 566–571
TEXT_Create 567
TEXT_CreateAsChild 567
TEXT_CreateEx 567
TEXT_CreateIndirect 568
TEXT_DEFAULT_BK_COLOR 566
TEXT_DEFAULT_TEXT_COLOR 566
TEXT_DEFAULT_WRAPMODE 566
TEXT_FONT_DEFAULT 566
TEXT_GetDefaultFont 568
TEXT_SetBkColor 568
TEXT_SetDefaultFont 569
TEXT_SetDefaultTextColor 569
TEXT_SetDefaultWrapMode 569
TEXT_SetFont 570
TEXT_SetText 570
TEXT_SetTextAlign 570
TEXT_SetTextColor 570
TEXT_SetWrapMode 570
Tick 719–720
Time-related functions 719–721
Toggle behavior, of hardkeys 43, 46
Top window 247
TOUCH_X_ActivateY 634
TOUCH_X_MeasureY 634
Touch-screens 19
Transparency 247
Transparent reversed text 69
Transparent text 69
TREEVIEW 301
Treeview widget 572–591
TREEVIEW_AttachItem 575
TREEVIEW_CreateEx 576
TREEVIEW_CreateIndirect 577
TREEVIEW_DecSel 577
TREEVIEW_GetDefaultBkColor 577
TREEVIEW_GetDefaultFont 578
TREEVIEW_GetDefaultLineColor 578
TREEVIEW_GetDefaultTextColor 578
TREEVIEW_GetItem 579
TREEVIEW_GetSel 579
TREEVIEW_IncSel 580
TREEVIEW_InsertItem 580
TREEVIEW_ITEM_Collapse 587
TREEVIEW_ITEM_Create 587
TREEVIEW_ITEM_Delete 588
TREEVIEW_ITEM_Detach 588
TREEVIEW_ITEM_Expand 588
TREEVIEW_ITEM_GetInfo 589
TREEVIEW_ITEM_GetText 589
TREEVIEW_ITEM_GetUserData 589
TREEVIEW_ITEM_SetImage 590
TREEVIEW_ITEM_SetText 590

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

TREEVIEW_ITEM_SetUserData 591
TREEVIEW_SetAutoScrollH 581
TREEVIEW_SetAutoScrollV 582
TREEVIEW_SetBkColor 582
TREEVIEW_SetDefaultBkColor 582
TREEVIEW_SetDefaultFont 583
TREEVIEW_SetDefaultLineColor 583
TREEVIEW_SetDefaultTextColor 583
TREEVIEW_SetFont 583
TREEVIEW_SetHasLines 584
TREEVIEW_SetHasRows 586
TREEVIEW_SetImage 584
TREEVIEW_SetLineColor 585
TREEVIEW_SetSel 585
TREEVIEW_SetTextColor 586
Trial version, of emWin 32–33
Tutorial 28–29

U

uC/OS 235
  kernel interface routines for 242
Unicode 137, 155
  API reference 670
  displaying characters in 668
User drawn widgets 307
UTF-8 strings 669

V

Validation, of windows 247
Values, displaying 75
Vectorized symbols 100
Viewer 19–59
Virtual display 19
Virtual screen support 601–608
Visual C++ 32, 34
  directory structure for 34
VNC support ??– 717

W

Western Latin character set (see ISO 8859-1)
WIDGET_DRAW_ITEM_FUNC 307
WIDGET_GetDefaultEffect 306
WIDGET_ITEM_DRAW 308
WIDGET_ITEM_DRAW_INFO 123
WIDGET_ITEM_GET_XSIZE 308
WIDGET_ITEM_GET_YSIZE 308
WIDGET_SetDefaultEffect 306
WIDGET_SetEffect 307
WIDGET_USE_PARENT_EFFECT 303
Widgets 19, 299–592, 719
  currently available 300
  defining behavior of 597
  dynamic memory usage 302
  handles of 299, 302
  in dialogs 593
  initialization of 594, 596–597
  member functions of 302
  using 302
Win32, kernel interface routines for 243
Window coordinates 247
Window manager 19, 245–297, 299, 720
Window objects (see Widgets)
Window widget 592
WINDOW_BKCOLOR_DEFAULT 592
WINDOW_SetDefaultBkColor 592
Windows 245–297
  clearing 73–74
  properties of 246
  terms associated with 246–247
WM_Activate 261
WM_AttachWindow 261
WM_AttachWindowAt 262
WM_BringToBottom 262
WM_BringToTop 262
WM_BroadcastMessage 263
WM_CF_ANCHOR_BOTTOM 264
WM_CF_ANCHOR_LEFT 264
WM_CF_ANCHOR_RIGHT 264
WM_CF_ANCHOR_TOP 264
WM_CF_BGND 264
WM_CF_CONST_OUTLINE 264
WM_CF_FGND 264
WM_CF_HASTRANS 264
WM_CF_HIDE 264
WM_CF_LATE_CLIP 264
WM_CF_MEMDEV 264
WM_CF_MEMDEV_ON_REDRAW 265
WM_CF_SHOW 265
WM_CF_STAYONTOP 265
WM_ClrHasTrans 263
WM_CREATE 250–251
WM_CreateTimer 290
WM_CreateWindow 263
WM_CreateWindowAsChild 265
WM_Deactivate 266
WM_DefaultProc 266
WM_DELETE 250–251
WM_DeleteTimer 291
WM_DeleteWindow 266
WM_DetachWindow 267
WM_DisableMemdev 290
WM_DisableWindow 267
WM_EnableMemdev 290
WM_EnableWindow 267
WM_Exec 267, 301, 720
WM_Exec1 268
WM_ForEachDesc 268
WM_GET_ID 250–251
WM_GetActiveWindow 269
WM_GetBackgroundWindow 270
WM_GetCallback 269
WM_GetClientRect 270
WM_GetClientRectEx 270
WM_GetClientWindow 292
WM_GetDesktopWindowEx 270–271
WM_GetDialogItem 271
WM_GetFirstChild 271
WM_GetFocussedWindow 271
WM_GetHasTrans 272
WM_GetId 292
WM_GetInsideRect 292
WM_GetInsideRectEx 293
WM_GetInvalidRect 272
WM_GetNextSibling 272
WM_GetOrgX 273
WM_GetOrgY 273
WM_GetParent 273
WM_GetPrevSibling 273
WM_GetScrollPosH 293
WM_GetScrollPosV 293
WM_GetScrollState 293
WM_GetStayOnTop 273

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

WM_GetUserData 274
WM_GetWindowOrgX 274
WM_GetWindowOrgY 274
WM_GetWindowRect 274
WM_GetWindowRectEx 275
WM_GetWindowSizeX 275
WM_GetWindowSizeY 275
WM_HasCaptured 275
WM_HasFocus 275
WM_HideWindow 276
WM_INIT_DIALOG 250–251, 594, 596
WM_InvalidateArea 276
WM_InvalidateRect 276
WM_InvalidateWindow 277
WM_IsCompletelyCovered 277
WM_IsCompletelyVisible 277
WM_IsEnabled 278
WM_IsVisible 278
WM_IsWindow 278
WM_KEY 250–251
WM_MakeModal 279
WM_MENU 490
WM_MESSAGE 250
WM_MOUSEOVER 250, 254
WM_MOUSEOVER_END 250, 254
WM_MOVE 250, 252
WM_MoveChildTo 279
WM_MoveTo 279
WM_MoveWindow 279
WM_NOTIFICATION_CHILD_DELETED ... 251
WM_NOTIFICATION_CLICKED 310, 323, 337, ..350, 415, 429, 434, 453, 476, 510, 522, ... 540, 551, 559, 574
WM_NOTIFICATION_MOVED_OUT 310, 323, 337, ..350, 415, 429, 434, 453, 476, 510, ...522, 540, 574
WM_NOTIFICATION_RELEASED ...310, 323, 337, ..350, 415, 429, 434, 453, 476, 510, ... 522, 540, 551, 559, 574
WM_NOTIFICATION_SCROLL_CHANGED 337, ... 429, 434, 453, 510
WM_NOTIFICATION_SCROLLBAR_ADDED .. 551
WM_NOTIFICATION_SEL_CHANGED .... 337, 429, ...434, 453, 476
WM_NOTIFICATION_VALUE_CHANGED 323, 350, ...510, 522, 540, 551, 559, 574
WM_NOTIFY_PARENT ... 250, 252, 302, 594
WM_NOTIFY_VIS_CHANGED 250, 252
WM_NotifyParent 280
WM_PAINT 250, 253
WM_Paint 280, 301
WM_PaintWindowAndDescs 280
WM_PID_STATE_CHANGED 250, 254
WM_ReleaseCapture 280
WM_ResizeWindow 281
WM_RestartTimer 291
WM_SelectWindow 281
WM_SendMessage 281
WM_SendMessageNoPara 282
WM_SendToParent 282
WM_SET_FOCUS 250, 253
WM_SET_ID 250, 253
WM_SetCallback 283
WM_SetCapture 283
WM_SetDesktopColor 282–283
WM_SetDesktopColorEx 283
WM_SetFocus 284
WM_SetHasTrans 284
WM_SetpfPollPID 285
WM_SetScrollPosH 294
WM_SetScrollPosV 294
WM_SetScrollState 295
WM_SetSize 285
WM_SetStayOnTop 286
WM_SetTransState 286
WM_SetUserClipRect 287
WM_SetUserData 288
WM_SetWindowPos 285
WM_SetXSize 286
WM_SetYSize 286
WM_ShowWindow 288
WM_SIZE 250, 253
WM_SUPPORT_NOTIFY_VIS_CHANGED .258
WM_SUPPORT_TRANSPARENCY 258
WM_TIMER 250, 253
WM_TOUCH 250, 255
WM_TOUCH_CHILD 250, 255
WM_Update 288
WM_UpdateWindowAndDescs 289
WM_USER 251, 257
WM_ValidateRect 289
WM_ValidateWindow 289

X

X-axis 20
XOR drawing mode 87
XOR text 69
xxxx 480

Y

Y-axis 20

Z

Z-position 247

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG

 © 1997 - 2008 SEGGER Microcontroller GmbH & Co. KG