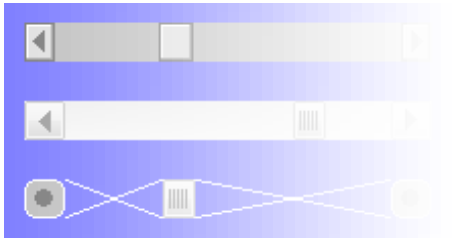
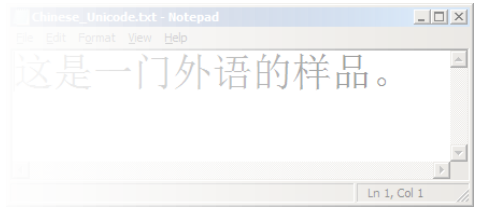
emWin Training

Content





The shipment

Start:

- All you need for a new project including WIN32 simulation
- Preconfigured WIN32 simulation project

Documentation:

- API-documentation
- This training including all samples

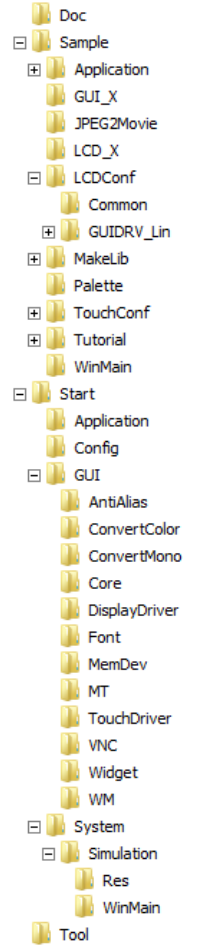
Samples:

- Application samples
- Tutorials
- Display configuration samples
- RTOS, VNC, Library generation, ...

Tools:

- All tools which are part of the license

Note: Most of the upcomming questions of an emWin newbie could be cleared up by a looking into the samples and the documentation.



TrueType and PNG

PNG support for emWin can be achieved by using the libpng' library (www.libpng.org) which is not part of emWin. An adapted version of this library ready to use with emWin is **available under**

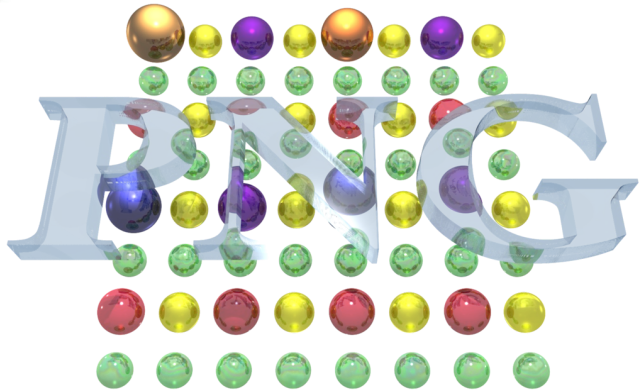
www.segger.com/downloads/emwin/emWin_PNG

License: BSD style license and copyright notice.

TTF support for emWin can be achieved by using the FreeType font library which is not part of emWin. An adapted version of this library ready to use with emWin is **available under**

www.segger.com/downloads/emwin/emWin_FreeType

License: BSD style license with credit clause.



'Hello World' with emWin

The **application entry point** in the simulation is:

```
void MainTask(void) {
  ...
}
```

Keep source of hardware project identical to the simulation. Call `MainTask()` from within the `main()` function or start it as a separate task:

```
OS_CREATETASK(&TCB, "MainTask",
              MainTask, 100, Stack);
```

Only one include file is required (for the core functions):

```
#include „GUI.h“
```

It is required to call `GUI_Init()` at first.

A loop is required to keep the application alive:

```
while(1) {
  GUI_Delay(10);
}
```

```
#include "GUI.h"

void MainTask(void) {
  GUI_Init();
  GUI_DispString("Hello world");
  while (1) {
    GUI_Delay(10);
  }
}
```

Color management

How does emWin work with colors?

- Color information 32bpp
- Config switch **GUI_USE_ARGB** for ABGR or ARGB format
- Color conversion into '**Index Values**' and vice versa
- `LCD_X_Config()` must set up the right color conversion
- A 'Fixed Palette Mode' is recommended
- A 'Custom Palette' could degrade performance

Alpha channel

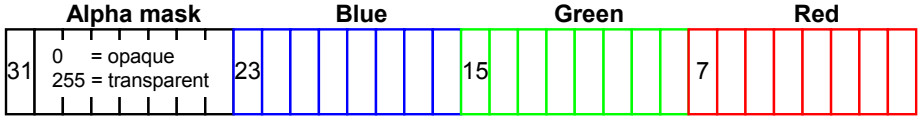
- **GUI_USE_ARGB = 0**:
  0 means opaque, 255 means 100% transparent
- **GUI_USE_ARGB = 1**:
  0 means 100% transparent, 255 means opaque

TrueColor - 8 bits for each component (24 or 32bpp)

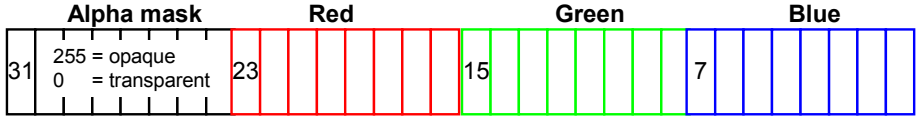
HighColor - 5 bits blue, 6 bits green, 5 bits red (16bpp)

Note: Changing the color format within a project could lead into wrong colors.

GUI_USE_ARGB = 0



GUI_USE_ARGB = 1




```
#include "GUI.h"

void MainTask(void) {
  GUI_Init();
  #if (GUI_USE_ARGB == 0)
    GUI_SetBkColor(0x00FFFF);
    GUI_SetColor(0xFF0000);
  #else
    GUI_SetBkColor(0xFFFFFF00);
    GUI_SetColor(0xFF0000FF);
  #endif
  GUI_Clear();
  GUI_DispString("Hello world");
  while (1) {
    GUI_Delay(10);
  }
}
```

```
#include "GUI.h"

void MainTask(void) {
  GUI_Init();
  GUI_SetBkColor(GUI_YELLOW);
  GUI_SetColor(GUI_BLUE);
  GUI_Clear();
  GUI_DispString("Hello world");
  while (1) {
    GUI_Delay(10);
  }
}
```

Fixed- and custom palette mode

Fixed palette mode:

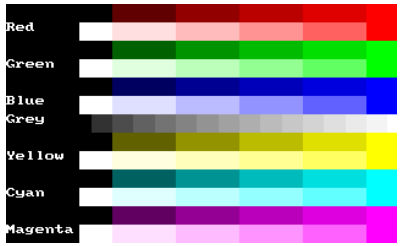
- Conversion routines are used for converting a color into an index value and vice versa.
- Most common used fixed palette modes:

  GUICC_M8888I, GUICC_M565, GUICC_8666

Custom palette mode:

- A custom palette lists all the available colors.
- Color conversion is done by searching the closest match from all available palette colors.
- Could **degrade performance**.
- Fixed palette mode to be used:

  GUICC_0



Note: Best performance is achieved with fixed palette modes.

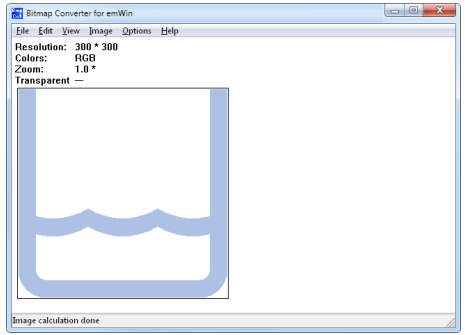
Introduction summary

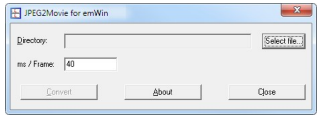
You should now know the following:

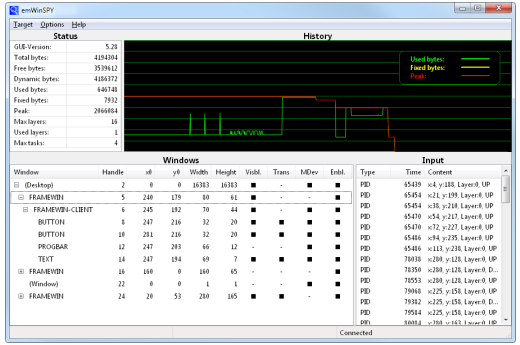
- **Difference between 'Color' and 'Index' values**
- **Application entry point of the simulation**
- **The difference between GUI_USE_ARGB = 0 and GUI_USE_ARGB = 1**
- **How to add PNG and TrueType support be added**
- **The difference between 'fixed palette mode' and 'custom palette mode'**

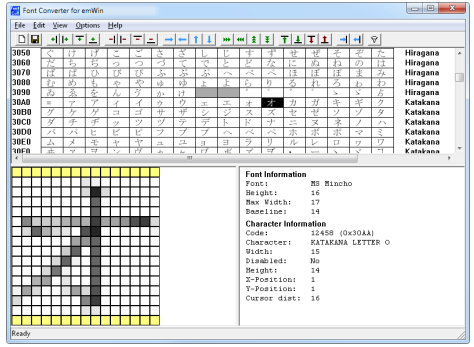
Tools overview

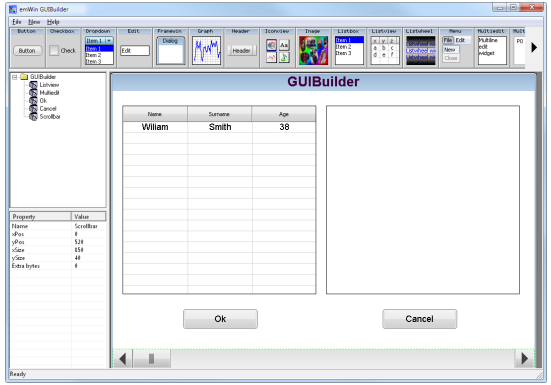
- **Bitmap converter**
  Converts images and saves them as C-files or streamed bitmaps.
- **emWinView**
  To be used while stepping through the debugger of the simulation.
- **emWinSPY**
  Shows runtime information of the embedded target on a PC.
- **emVNC**
  VNC client to be used for VNC connections with or without emWin.
- **emWinPlayer**
  Movie player to be used to play EMF files.
- **Bin2C**
  Converts any kind of file into a byte array.
- **U2C**
  Used for converting UTF8-text into C-code.
- **JPEG2Movie**
  Used for creating emWin movie files in conjunction with FFMPEG.
- **Font converter (optional)**
  Can be used for converting any font installed on the host system into emWin compatible formats.
- **GUIBuilder (shipped with window manager)**
  Used to generate dialog based applications without writing any line of code.

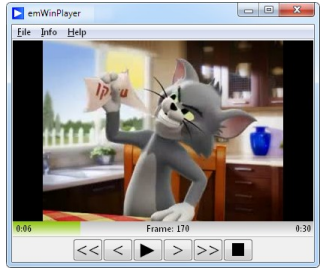












BitmapConverter - Supported image formats

Input:

- **BMP, GIF** (animations)**, PNG** (alpha channel) and **JPEG**

Output:

- **C files**

  Most recommended image format in case of enough addressable ROM.

- **Streamed bitmaps**

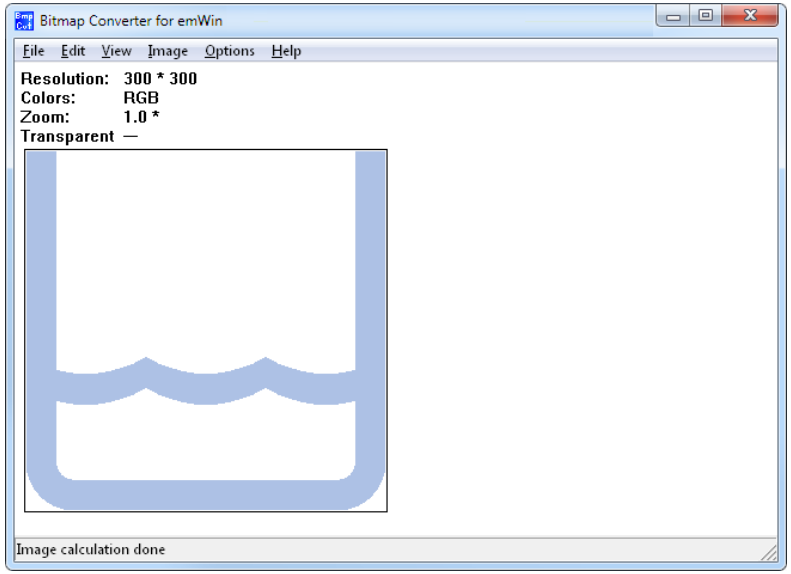
  Most recommended binary format for systems short on addressable ROM.
  Accessed by `GetData()` function passed by the customer.

- **Animated sprites and cursors**

  GIF animations converted automatically into animated sprites or cursors.

- **BMP-, GIF and PNG**

  ...



BitmapConverter - Manipulation options

- **Color conversion**

  Use '**Image\Convert Into\...**'.

- **Dithering**

  Use '**Image\Dither into\...**' .

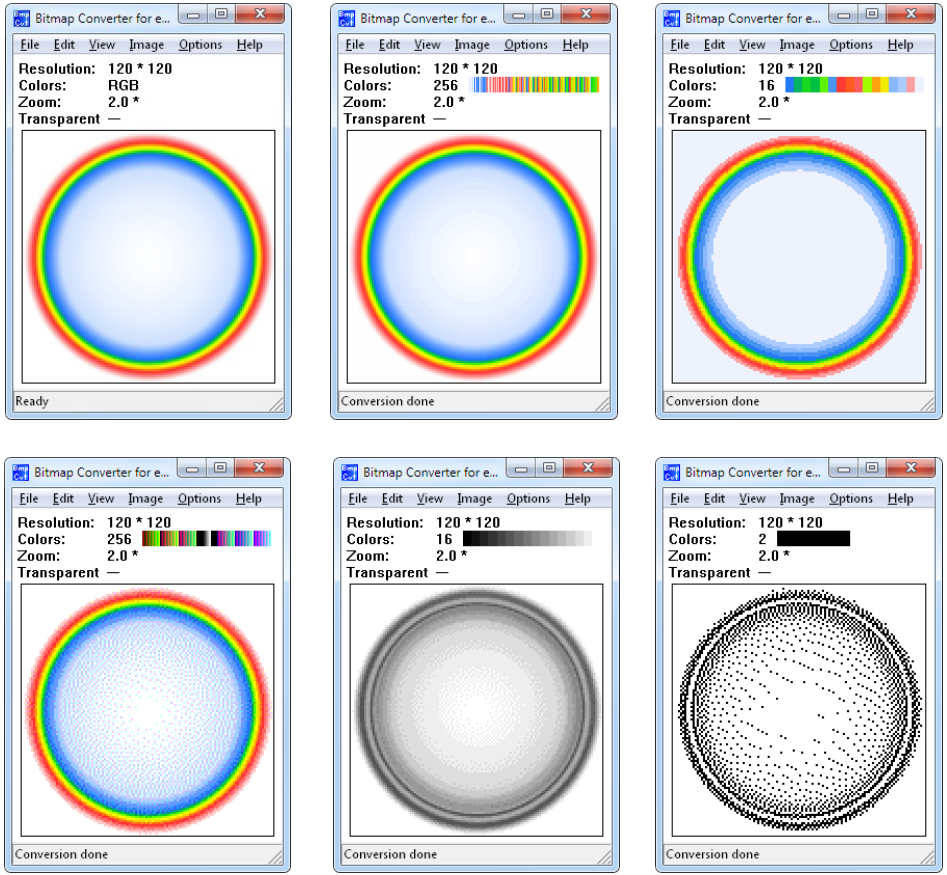
- **Color reduction**

  Use '**Image\Reduce Colors\...**'.

- **Flipping, rotating, inverting and scaling**

  Use '**Image\Flip, Rotate, Invert and Scale**'

Important: When saving images in high color format, the format should fit **exactly** to the layer configuration. Otherwise color conversion is required for each pixel during the drawing operation which degrades the performance significantly. **--- KEEP IN MIND ---**



BitmapConverter - Transparency

Meaning of transparency:

- Supported for palette based bitmaps (1-8bpp).
- Pixel data are index values pointing into the bitmap palette.
- Index value 0 means pixel is transparent.
- Not to be mixed up with alpha blending.

Creating transparent images:

- **Selecting the color of palette based bitmaps**

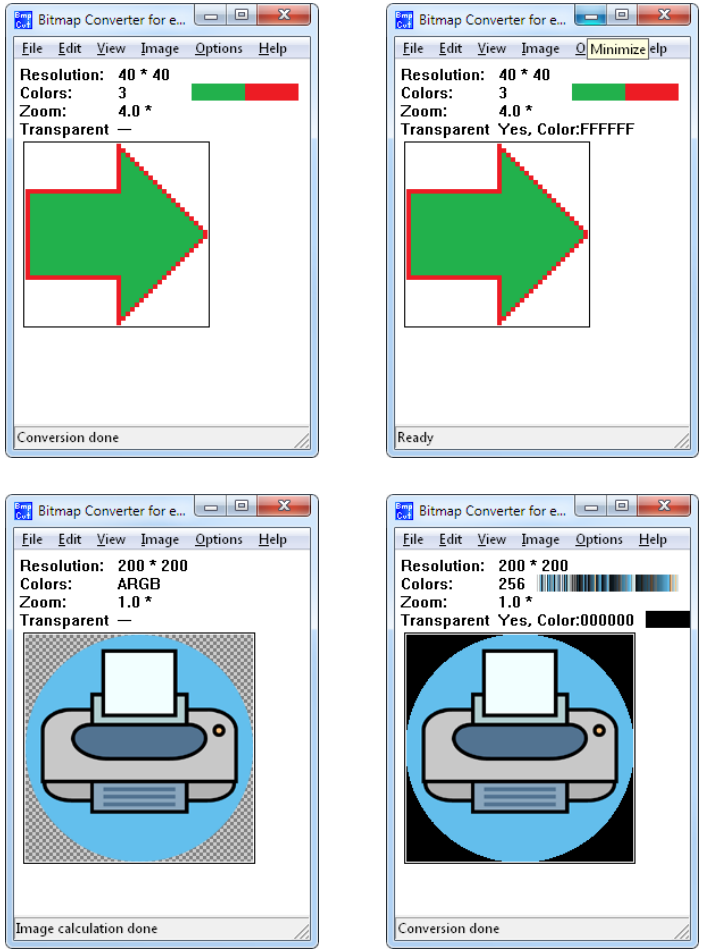
  Use 'Image\Transparency...' for selecting the color to be treated as transparent.

  → (see right side in the upper sample)

- **Converting PNG files**

  Use 'Image\Convert to\Best palette + transparency'.

  → (see right side in the lower sample)



SAMPLES: BitmapConverter\ArrowRight_40x40.bmp
BitmapConverter\Printer_200x200_ARGB.png

BitmapConverter - Alpha blending

- **True color images with alpha mask**

  Simply load PNG files and save them as 'True color bitmap with alpha channel'.

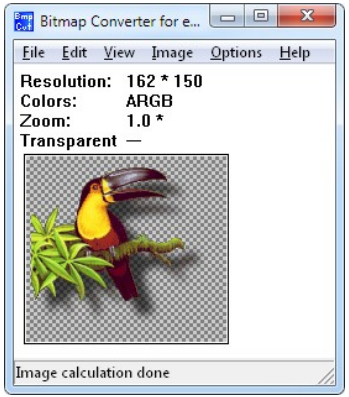
- **Alpha mask bitmaps**

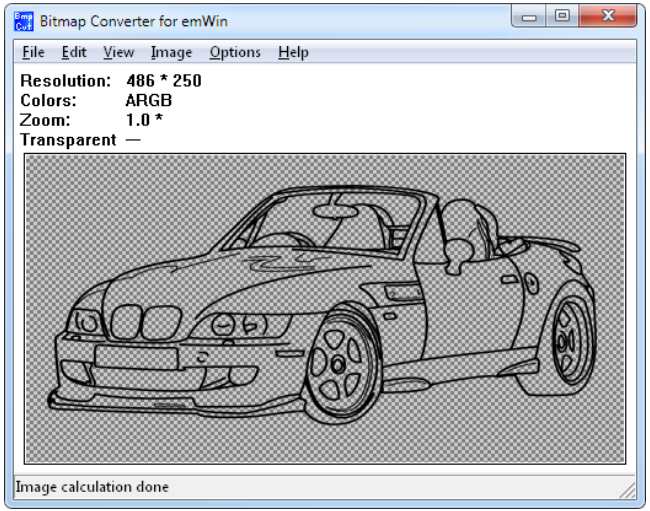
  Contains only RLE-compressed alpha channel.

  Drawn by emWin using the current foreground color.

  Recommended for smooth shapes.

Note: Alpha mask bitmaps are the most effective way to achieve smooth shapes or outlines which could be drawn in different colors. **--- KEEP IN MIND ---**





SAMPLES: BitmapConverter\Toucan_162x150.png
BitmapConverter\Car_286x250.png

- Converts **any installed font** of the host system.
- Font **types**: STD, EXT, AA2, ...
- Font **formats**: C, XBF, SIF, BDF.
- C- and XBF-font files can be **load**ed and **change**d.
- C- and XBF-font files can be **merge**d.
- **Pattern files** can be used to select characters.

Insert pixel line

Delete pixel line

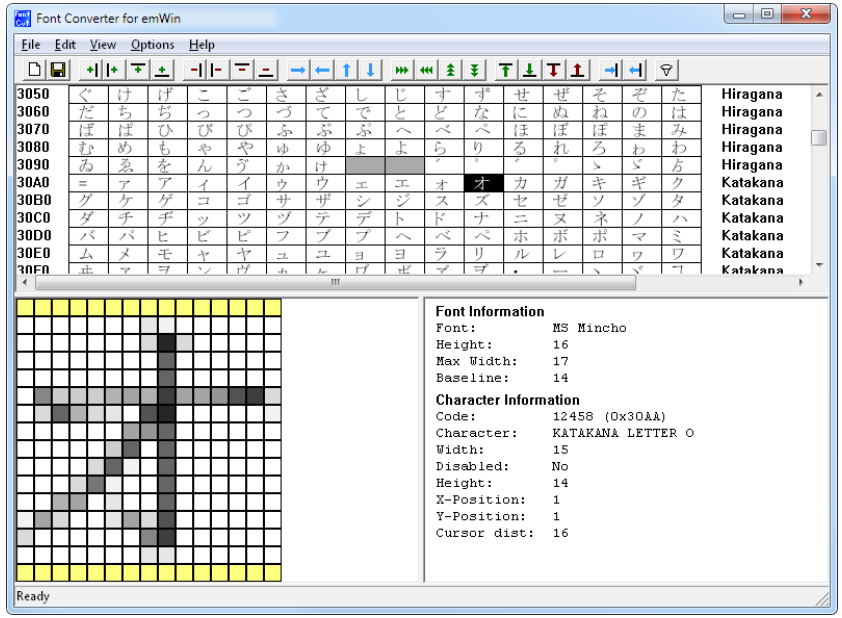
Shift content

Move position

Insert/delete pixel line to font

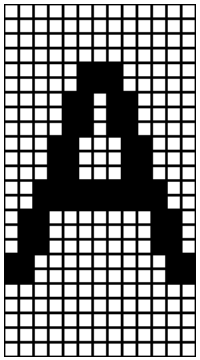
Increase/decrease cursor increment

Filter for showing only blocks with content

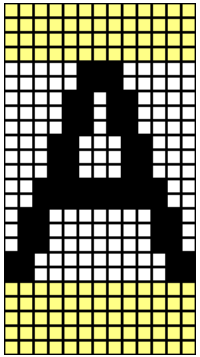


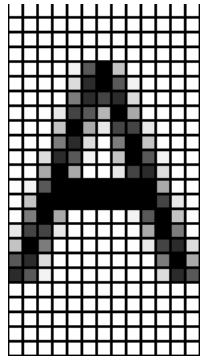
- **Standard (obsolete)**
  - Does not support compound characters.
  - Wasting ROM and quality not perfect.
- **Antialiased, 2bpp and 4bpp (obsolete)**
  - see 'Standard' above.
- **Extended**
  - Successor of 'Standard'.
  - Support for compound characters.
- **Extended, Framed**
  Perfect for undetermined backgrounds like photos.
- **Extended, antialiased, 2 and 4bpp**
  Successor of 'Antialiased, 2bpp and 4bpp' with support for compound characters.

Note: Support for 'Standard' fonts is implemented for reasons of compatibility only. The best quality could be achieved with 'Extended' fonts.

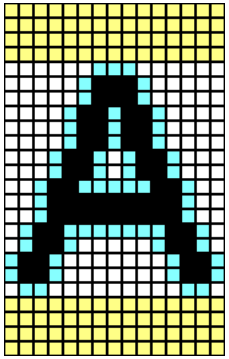


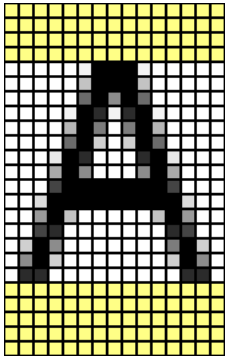
Standard
(obsolete)





Antialiased
(obsolete)





FontConverter - Font formats

- **C file format**

  Needs to reside completely in addressable memory area.

  + **Easy usage, simply compile and link**
  - **Performance degrades with big fonts**

- **SIF file format**
  (**S**ystem **I**ndependent **F**ont)

  Same information as C file format, but in binary form.

  - **Must be loaded completely into memory**
  - **Performance degrades with big fonts**

- **XBF file format**
  (E**x**ternal **B**inary **F**ont)

  Binary file format with fast character access.
  Recommended with a large number of character areas.

  + **Could reside on any media**
  - **File system with cache recommended**

- **BDF file format**
  (**B**itmap **D**istribution **F**ormat)

  Reading of font files in BDF format supported.
  A large number of BDF font files exist in the UNIX-world.
  Defined by Adobe.

FontConverter - Chinese language sample

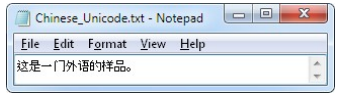
Unicode support should be enabled once at the beginning of the application:

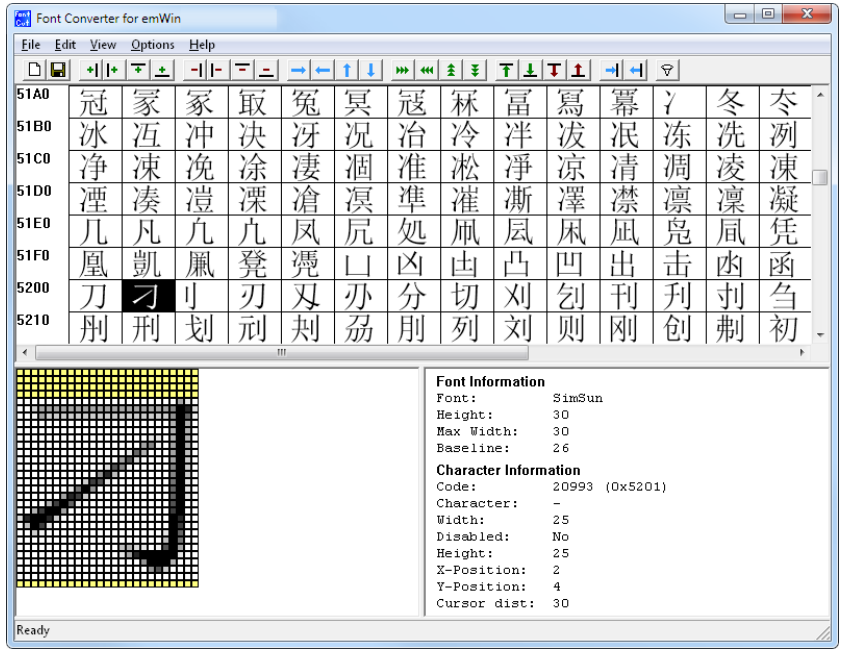
```
GUI_Init();
GUI_UC_SetEncodeUTF8(); // Enable UTF-8 support
```

The following shows the steps for getting a Chinese text sample on the display:

- Save text file with Notepad in UTF-8 and Unicode format.
- Create a new font of desired size and type.
- Disable all characters.
- Read pattern file (Unicode) with desired characters.
- Save font file.
- Convert text file (UTF-8) into C code.
- Include font and text into project.

Note: Unicode support of emWin is based on UTF-8 encoding.






```
static char acText[] = "\xe8\xbf\x99\xe6\x98\xaf\xe4\xb8\x80\xe9\x97\xa8\xe5\xa4\x96"
                       "\xe8\xaf\xad\xe7\x9a\x84\xe6\xa0\xb7\xe5\x93\x81\xe3\x80\x82";

void MainTask(void) {
  GUI_Init();
  GUI_UC_SetEncodeUTF8();
  GUI_SetFont(&GUI_FontChinese_30);
  GUI_DispString(acText);
  while (1) {
    GUI_Delay(100);
  }
}
```

FontConverter - Using XBF files

Recommended for large fonts with many characters and **gaps**.

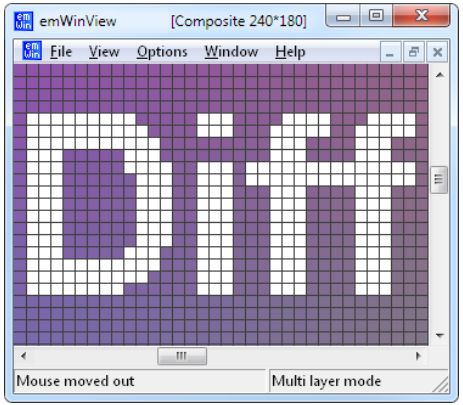
- Interface between an XBF file and emWin is the **GetData**() function.
- **GUI_XBF_CreateFont()** should be used to create the font structure.
- The **void* pointer** is **passed to GetData**().
- **GetData() has to copy** the requested **data** of the file into the given buffer.
- Return value is 0 on success.

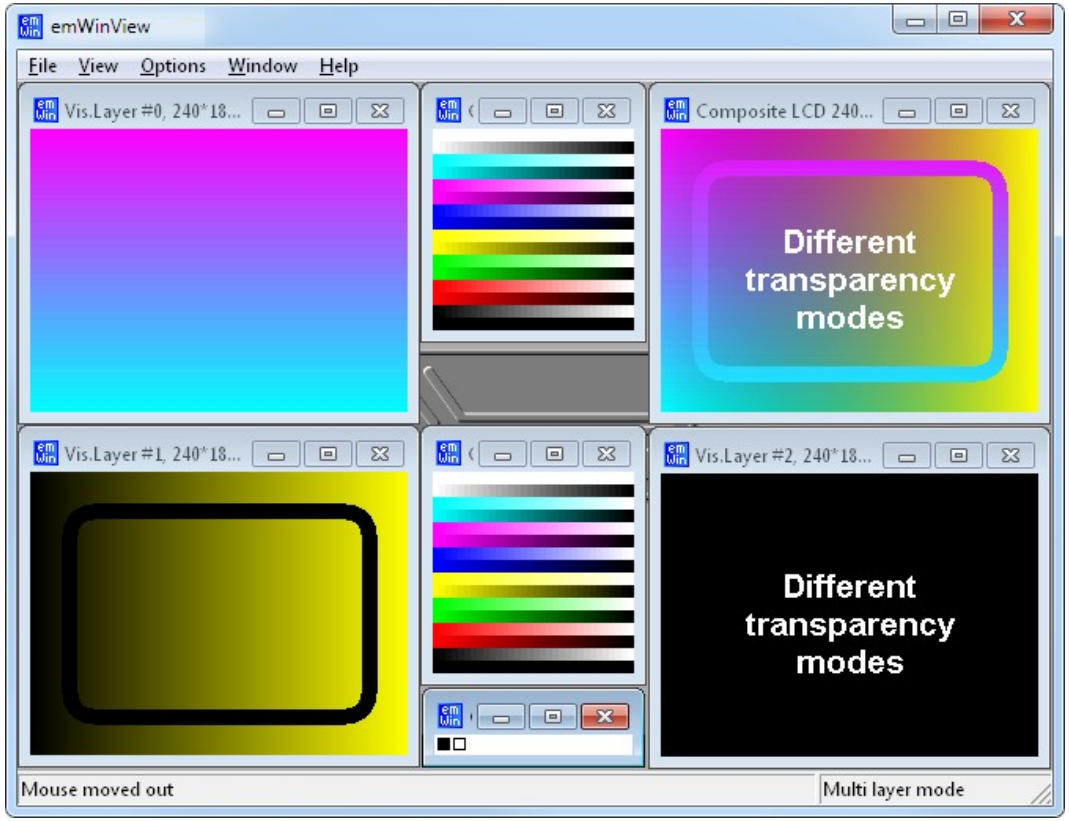
Note: XBF fonts contain an access table which makes sure that each character can be accessed with only 2 _GetData() accesses.

```
#include <windows.h>

#include "GUI.h"

/*********************************************************************
*
*       _cbGetData
*
* Parameters:
*   Off      - Position of XBF file to be read
*   NumBytes - Number of requested bytes
*   pVoid    - Application defined pointer
*   pBuffer  - Pointer to buffer to be filled by the function
*/
static int _cbGetDataXBF(U32 Off, U16 NumBytes, void * pVoid, void * pBuffer) {
  DWORD NumBytesRead;
  HANDLE hFile;

  hFile = *(HANDLE *)pVoid;
  // Set file pointer to the requested position
  if (SetFilePointer(hFile, Off, 0, FILE_BEGIN) == 0xFFFFFFFF) {
    return 1; // Error
  }
  // Read font data
  if (!ReadFile(hFile, pBuffer, NumBytes, &NumBytesRead, 0)) {
    return 1; // Error
  }
  if (NumBytesRead != NumBytes) {
    return 1; // Error
  }
  return 0; // Ok
}

/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  HANDLE       hFile;
  GUI_FONT     Font;
  GUI_XBF_DATA XBF_Data;

  GUI_Init();
  // Get file handle
  hFile = CreateFile(Filename, GENERIC_READ, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  // Create XBF font
  GUI_XBF_CreateFont(&Font,                  // Pointer to GUI_FONT structure in RAM
                     &XBF_Data,              // Pointer to GUI_XBF_DATA structure in RAM
                     GUI_XBF_TYPE_PROP_EXT, // Font type to be created
                     _cbGetDataXBF,          // Pointer to callback function
                     (void *)&hFile);        // Pointer to be passed to GetData function
  // Draw some text
  GUI_DispStringHCenterAt("Hello world!", 160, 80);
  while (1) {
    GUI_Delay(100);
  }
}
```

The viewer can be used to view the content of the display when debugging the simulation.

- **Magnification** of each layer window
- **Composite view** with multiple layers
- Content **visible** even **if** the **debugger stops**
- **Touch events** are passed to the simulation



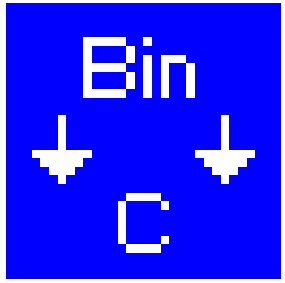


Bin2C - Converts binary files into C code

Its often useful to get a binary file into the target without having a file system available. In this case the Bin2C tool can be used. It can be used to convert any file into a C array.

Without a file system it offers the following features:

- **Direct drawing of image files**
- **Use of XBF font files**
- **Use of TrueType font files**

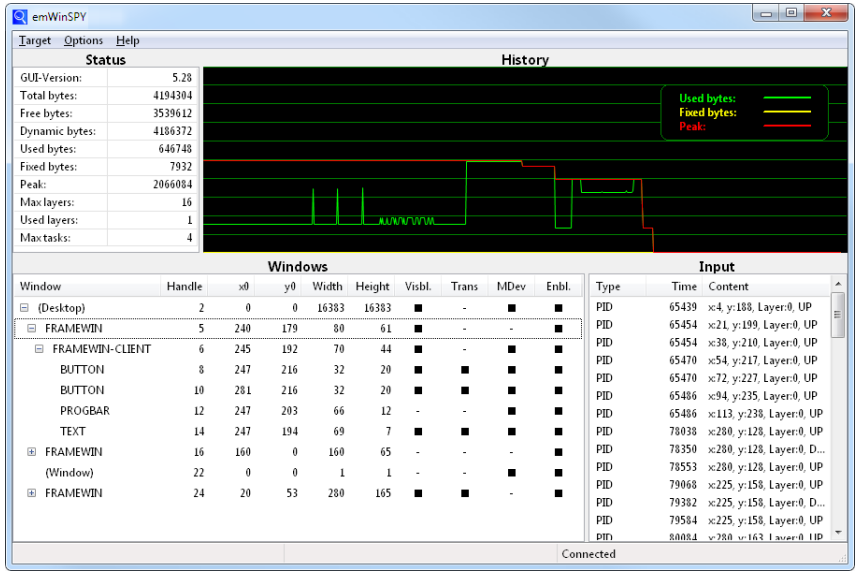


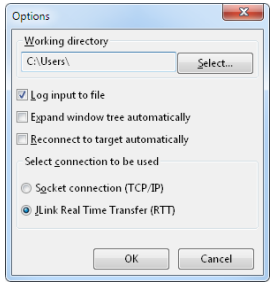

```
#include "GUI.h"

static unsigned char _acFontABC_16_EXT_XBF[] = {
  0x47, 0x55, 0x49, 0x58, 0x10, 0x00, 0x10, 0x00, 0x0D, 0x00, 0x07, 0x00, 0x0A,
  0x00, 0x41, 0x00, 0x43, 0x00, 0x24, 0x00, 0x00, 0x00, 0x20, 0x00, 0x44, 0x00,
  0x00, 0x00, 0x16, 0x00, 0x5A, 0x00, 0x00, 0x00, 0x16, 0x00, 0x09, 0x00, 0x09,
  0x00, 0x0A, 0x00, 0x00, 0x00, 0x03, 0x00, 0x02, 0x00, 0x08, 0x00, 0x14, 0x00,
  0x14, 0x00, 0x14, 0x00, 0x22, 0x00, 0x22, 0x00, 0x7F, 0x00, 0x41, 0x00, 0x80,
  0x80, 0x80, 0x80, 0x09, 0x00, 0x07, 0x00, 0x0A, 0x00, 0x01, 0x00, 0x03, 0x00,
  0x01, 0x00, 0xFC, 0x82, 0x82, 0x82, 0xFC, 0x82, 0x82, 0x82, 0x82, 0xFC, 0x09,
  0x00, 0x07, 0x00, 0x0A, 0x00, 0x01, 0x00, 0x03, 0x00, 0x01, 0x00, 0x38, 0x44,
  0x82, 0x80, 0x80, 0x80, 0x80, 0x82, 0x44, 0x38, 0x00
};

static int _cbGetDataXBF(U32 Off, U16 NumBytes, void * pVoid, void * pBuffer) {
  U8 * p;

  p = (U8 *)pVoid;
  //
  // Copy data
  //
  memcpy(pBuffer, p + Off, NumBytes);
  return 0; // Ok
}

void MainTask(void) {
  GUI_FONT     Font;
  GUI_XBF_DATA XBF_Data;

  GUI_Init();
  //
  // Create XBF font
  //
  GUI_XBF_CreateFont(&Font,                          // Pointer to GUI_FONT structure
                     &XBF_Data,                      // Pointer to GUI_XBF_DATA structure
                     GUI_XBF_TYPE_PROP_EXT,          // Font type to be created
                     _cbGetDataXBF,                  // Pointer to callback function
                     (void *)_acFontABC_16_EXT_XBF); // Pointer to be passed to GetData
  //
  // Draw some text
  //
  GUI_DispStringHCenterAt("ABC", 160, 80);
  while (1) {
    GUI_Delay(100);
  }
}
```

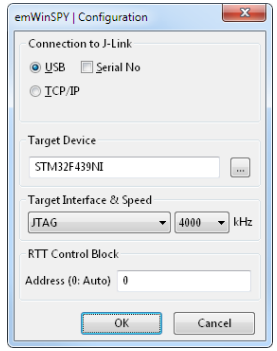
emWinSPY - Runtime information of target system

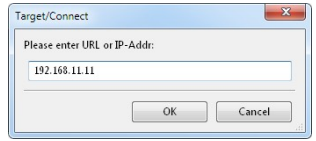
Shows runtime information of a target system on the PC:

- Current state of **memory usage** including the peak.
- History of **fixed** and **dynamic** memory usage.
- Current **state of the window manager** with information about each window like handle, position, size, visibility, transparency, memory device flag and enabled state.
- Input of **PID** and **MultiTouch** events including timestamp, position, layer and type (UP, DOWN and MOVE in case of MultiTouch).
- **Keyboard input** including key code, timestamp and type (UP and DOWN).
- **Screenshots** could be taken directly from the target.
- Could be used in simulation by GUI_SPY_X_StartServer().
- Comunicates via **TCP/IP** or **RTT** (Real Time Transfer).
- Configuration example shows how to start the SPY-server on the target hardware using TCP/IP or RTT.
- Detailed **information about RTT on www.segger.com**.





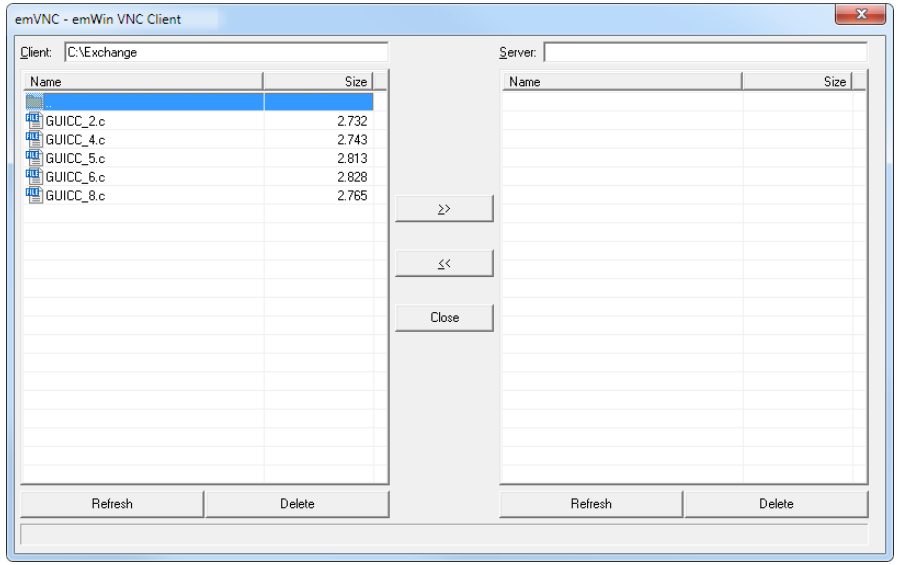




emVNC - emWin VNC client

Establish an **RFB** (**R**emote **F**rame **B**uffer) connection between the host system and a VNC server.

Client is part of the basic package of emWin.
Server is optional.



File Transfer

Only supported with emVNC. Not part of RFB standard.

Opening FT window of viewer

Option of the system menu accessible with <ALT>+<SPACE> or by clicking the application logo of the title bar.

Note: 'Open File Transfer Window' is shown only if the connected target has file transfer options enabled.

Tools summary

You should now know the following:

- **How to achieve the best performance when drawing high color bitmaps on 16bpp layers**
- **What exactly means transparency of a bitmap**
- **The most recommended bitmap format for smooth shapes and outlines**
- **The most recommended font format for languages like Chinese and other fonts with many characters and gaps**
- **How to achieve rendering of Unicode text**
- **How to achieve file transfer via VNC**
- **How to connect a target to emWinSPY**

Configuration

Compile time configuration (.h files):

- **GUIConf.h**

  Configuration of available features

- **LCDConf.h**

  Display driver configuration (obsolete)

Runtime configuration (.c files):

- **GUIConf.c**

  Configuration of dynamic memory

- **LCDConf.c**

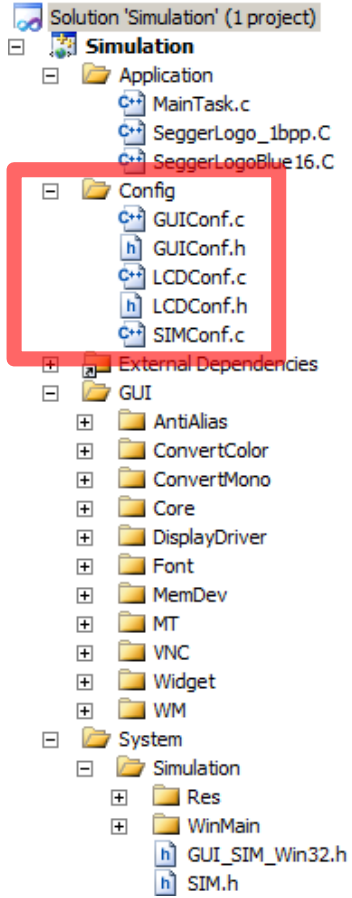
  Display driver configuration and initialization

- **SIMConf.c**

  Configuration of simulator

- **GUI_X.c / GUI_X_embOS.c**

  Not required in simulation



Configuration - GUIConf.h

Compile time configuration of emWin:

- **GUI_NUM_LAYERS**
  Defines the maximum number of available layers

- **GUI_OS**
  Enables multitasking support

- **GUI_SUPPORT_TOUCH**
  Enables optional touch screen support

- **GUI_DEFAULT_FONT**
  Default font to be used, runtime configuration routine also available.

- **GUI_WINSUPPORT**
  Enables use of optional window manager

- **GUI_SUPPORT_MEMDEV**
  Enables use of optional memory devices

```
#ifndef GUICONF_H
#define GUICONF_H

/*********************************************************************
*
*       Multi layer/display support
*/
#define GUI_NUM_LAYERS            1    // Maximum number of available layers

/*********************************************************************
*
*       Multi tasking support
*/
#define GUI_OS                    (0)  // Compile with multitasking support

/*********************************************************************
*
*       Configuration of touch support
*/
#define GUI_SUPPORT_TOUCH         (0)  // Support a touch screen (req. win-manager)

/*********************************************************************
*
*       Default font
*/
#define GUI_DEFAULT_FONT          &GUI_Font6x8

/*********************************************************************
*
*         Configuration of available packages
*/
#define GUI_WINSUPPORT                1    /* Use window manager */
#define GUI_SUPPORT_MEMDEV            1    /* Memory device package available */

#endif  /* Avoid multiple inclusion */
```

Configuration - GUIConf.c

`GUI_X_Config()` is the very first routine called during the process of initialization. Main task here is assigning memory to the dynamic memory management system of emWin:

- **GUI_ALLOC_AssignMemory()**

  Passes a pointer to a memory block and its size in bytes to the memory manager.

Further initialization functions available to be used here:

- **GUI_SetDefaultFont()**

  Sets the default font to be used.

- **GUI_SetSignalEventFunc()**
  GUI_SetWaitEventFunc()
  GUI_SetWaitEventTimedFunc()

  Optional to be used to put the GUI task into sleep mode until PID- or keyboard input occurs. Default is calling GUI_X_ExecIdle() located in GUI_X.c

Note: Memory must be accessible 8- 16- and 32 bit wise. Memory access is checked during initialization (in debug mode).

```
/*********************************************************************
*
*       Defines
*
**********************************************************************
*/
//
// Define the available number of bytes available for the GUI
//
#define GUI_NUMBYTES  0x200000

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/
/*********************************************************************
*
*       GUI_X_Config
*
* Purpose:
*   Called during the initialization process in order to set up the
*   available memory for the GUI.
*/
void GUI_X_Config(void) {
  //
  // 32 bit aligned memory area
  //
  static U32 aMemory[GUI_NUMBYTES / 4];
  //
  // Assign memory to emWin
  //
  GUI_ALLOC_AssignMemory(aMemory, GUI_NUMBYTES);
  //
  // Set default font
  //
  GUI_SetDefaultFont(GUI_FONT_6X8);
}
```

Configuration - LCDConf.c

`LCD_X_Config()`, display driver configuration:

Called immediately after `GUI_X_Config()` has been executed. Main task here is creating and configuring a display driver for each layer:

- **GUI_DEVICE_CreateAndLink()**

  Creates the driver device and links it into the device chain

- **LCD_SetSizeEx() / LCD_SetVSizeEx()**

  Display size configuration

- **LCD_SetVRAMAddrEx()**

  Required in case of linear addressable memory

`LCD_X_DisplayDriver()`, driver callback routine:

- Called by the driver for several tasks, important when using advanced features like multiple buffers, smooth scrolling or virtual pages. Must return 0 after successful initialization.

Note: LCD_X_DisplayDriver() Must return 0 after successful initialization.

LCD_X_Config:

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Display driver configuration, required for Lin-driver
  //
  if (LCD_GetSwapXY()) {
    LCD_SetSizeEx (0, YSIZE_PHYS, XSIZE_PHYS);
    LCD_SetVSizeEx(0, YSIZE_PHYS * NUM_VSCREENS, XSIZE_PHYS);
  } else {
    LCD_SetSizeEx (0, XSIZE_PHYS, YSIZE_PHYS);
    LCD_SetVSizeEx(0, XSIZE_PHYS, YSIZE_PHYS * NUM_VSCREENS);
  }
  LCD_SetVRAMAddrEx(0, (void *)VRAM_ADDR);
}
```

LCD_X_DisplayDriver:

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData) {
  int r;

  switch (Cmd) {
  case LCD_X_INITCONTROLLER: {
    //
    // Called during the initialization process in order to set up the
    // display controller and put it into operation. If the display
    // controller is not initialized by any external routine this needs
    // to be adapted by the customer...
    //
    // …
    _InitLCDController();
    return 0;
  }
  . . .
  default:
    r = -1;
  }
  return r;
}
```

Configuration - Single task / Multiple tasks

One of the files **GUI_X.c** or **GUI_X_embOS.c** is required to be included into the target system.

- **GUI_X.c – Single task execution**

  Should be used in a single task environment. 'Single task' means that the project uses emWin only from within one single task.
  Main Purpose is to supply emWin with a timing base.

- **GUI_X_embOS.c – Multi task execution**

  If emWin is used by multiple tasks this file contains additional routines required for synchronizing multiple tasks.

Note 1:	If timing routines are used OS_TimeMS needs to be incremented each ms.
Note 2:	Multitasking systems need additional task synchronization routines as available in GUI_X_embOS.c.

GUI_X.c:

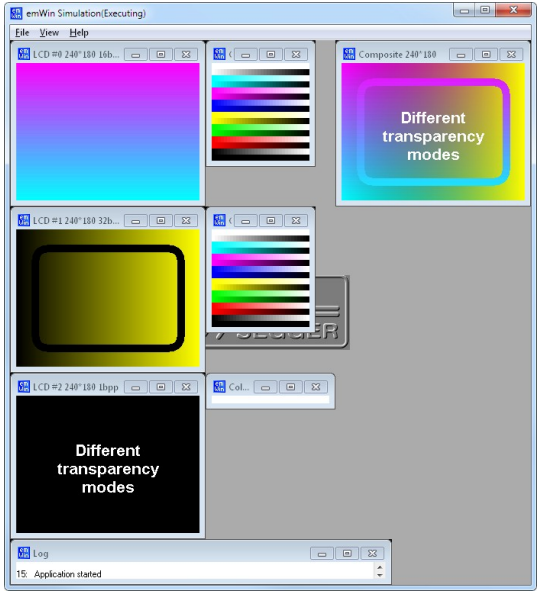
```
#include "GUI.h"

volatile int OS_TimeMS;

int GUI_X_GetTime(void) {
  return OS_TimeMS;
}

void GUI_X_Delay(int ms) {
  int tEnd = OS_TimeMS + ms;
  while ((tEnd - OS_TimeMS) > 0);
}
...
```

GUI_X_embOS.c:

```
int GUI_X_GetTime(void) {
  return OS_GetTime();
}

void GUI_X_Delay(int Period) {
  OS_Delay(Period);
}

void GUI_X_ExecIdle(void) {
  OS_Delay(1);
}

void GUI_X_InitOS(void)    { OS_CreateRSema(&_RSema);    }
void GUI_X_Unlock(void)    { OS_Unuse(&_RSema); }
void GUI_X_Lock(void)      { OS_Use(&_RSema);  }
U32  GUI_X_GetTaskId(void) { return (U32)OS_GetTaskID(); }

void GUI_X_WaitEvent(void)    {
  _pGUITask = OS_GetpCurrentTask();
  OS_WaitEvent(1);
}

void GUI_X_SignalEvent(void)    {
  if (_pGUITask) {
    OS_SignalEvent(1, _pGUITask);
  }
}

void GUI_X_WaitEventTimed(int Period) {
  static OS_TIMER Timer;
  static int Initialized;

  if (Period > 0) {
    if (Initialized != 0) {
      OS_DeleteTimer(&Timer);
    }
    Initialized = 1;
    OS_CreateTimer(&Timer, GUI_X_SignalEvent, Period);
    OS_StartTimer(&Timer);
    GUI_X_WaitEvent();
  }
}
```

Configuration - Multiple layers and transparency

- Each layer needs to be configured separately
- Different layer sizes are supported
- Different transparency modes are available
- Each layer shown separately in simulator and viewer
- Composite view available in simulator and viewer
- Number of layers not limited



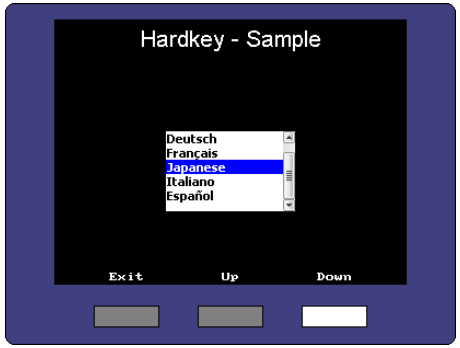

```
void LCD_X_Config(void) {
  int Layer;
  const GUI_DEVICE_API     * apDeviceAPI[] = {
    GUIDRV_LIN_24,      // Layer 0: 24bpp
    GUIDRV_LIN_8,       // Layer 1:  8bpp
    GUIDRV_LIN_1        // Layer 2:  1bpp
  };
  const LCD_API_COLOR_CONV * apColorConvAPI[] = {
    GUICC_888,          // Layer 0: 24bpp
    GUICC_8666,         // Layer 1:  8bpp
    GUICC_1             // Layer 2:  1bpp
  };
  void * apVRAM_Addr[] = {
    (void *)0x20000000, // Address of frame buffer layer 0
    (void *)0x2004B000, // Address of frame buffer layer 1
    (void *)0x2005DC00, // Address of frame buffer layer 2
  };

  for (Layer = 0; Layer < 3; Layer++) {
    GUI_DEVICE_CreateAndLink(apDeviceAPI[Layer], apColorConvAPI[Layer], 0, Layer);
    LCD_SetSizeEx (Layer, XSIZE_PHYS, YSIZE_PHYS);
    LCD_SetVSizeEx(Layer, XSIZE_PHYS, YSIZE_PHYS);
    LCD_SetVRAMAddrEx(Layer, apVRAM_Addr[Layer]);
  }
}


void SIM_X_Config() {
  int xSize, ySize;

  xSize = LCD_GetXSizeEx(0);
  ySize = LCD_GetYSizeEx(0);
  SIM_GUI_SetCompositeSize(xSize, ySize);
  SIM_GUI_SetTransMode(1, GUI_TRANSMODE_PIXELALPHA);
  SIM_GUI_SetTransMode(2, GUI_TRANSMODE_ZERO);
}


void MainTask(void) {
  int xSize, ySize;

  GUI_Init();
  xSize = LCD_GetXSize();
  ySize = LCD_GetYSize();
  GUI_SetLayerVisEx(0, 1);
  GUI_SetLayerVisEx(1, 1);
  GUI_SetLayerVisEx(2, 1);
  GUI_DrawGradientV(0, 0, xSize, ySize, GUI_MAGENTA, GUI_CYAN);
  GUI_SelectLayer(1);
  GUI_DrawGradientH(0, 0, xSize, ySize, 0xFF000000, 0x0000FFFF);
  GUI_SetColor(GUI_TRANSPARENT);
  GUI_DrawRoundedFrame(20, 20, xSize - 20, ySize - 20, 20, 10);
  GUI_SelectLayer(2);
  GUI_SetBkColorIndex(0);
  GUI_Clear();
  GUI_SetColor(GUI_WHITE);
  GUI_SetFont(GUI_FONT_24B_ASCII);
  GUI_DispStringHCenterAt("Different\ntransparency\nmodes", xSize / 2, ySize / 3);
  while (1) {
    GUI_Delay(100);
  }
}
```

Configuration - Device simulation

Multiple simulation modes available:

- **Generated frame view**

  Automatically generated frame surrounding the display with a small close button.

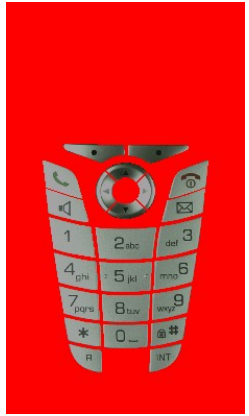


- **Custom bitmap view**

  Supports any number and any form of buttons.

- **Window view**

  Default view in multiple layer mode.



- **Magnification**

  Display can be magnified.

Configuration - Multiple buffering

Double buffering:

- Memory for 2 frame buffers required
- Waiting until next EOF interrupt required

Tripple buffering:

- Memory for 3 frame buffers required
- No waiting required

Usage:

- Early configuration
  `GUI_MULTIBUF_Config()`
- Automatic usage by window manager
  `WM_MULTIBUF_Enable()`
- Manual usage is also possible
  `GUI_MULTIBUF_Begin()`
  `GUI_MULTIBUF_End()`

Note: Multiple buffering avoids tearing and flickering effects and hides the proces of drawing. The framebuffer becomes visible at once.

```
/*********************************************************************
*
*       _LCD_CopyBuffer
*/
static void _LCD_CopyBuffer(int LayerIndex, int IndexSrc, int IndexDst) {
  U32 BufferSize, BaseAddr, AddrSrc, AddrDst;

  GUI_USE_PARA(LayerIndex);
  BufferSize = ((_XSIZE * _YSIZE * LCD_BITSPERPIXEL) >> 3);
  BaseAddr   = ((U32)&_aVRAM[0] + OFFSET_VRAM);
  AddrSrc    = BaseAddr + BufferSize * IndexSrc;
  AddrDst    = BaseAddr + BufferSize * IndexDst;
  GUI_MEMCPY((void *)AddrDst, (void *)AddrSrc, BufferSize);
}

/*********************************************************************
*
*       _ISR_EndOfFrame
*/
static void _ISR_EndOfFrame(void) {
  U32 Addr;

  if (_PendingBuffer >= 0) {
    Addr     = ((U32)&_aVRAM[0] + _XSIZE * _YSIZE * _PendingBuffer * (LCD_BITSPERPIXEL >> 3));
    SFR_ADDR = Addr;
    GUI_MULTIBUF_Confirm(_PendingBuffer);
    _PendingBuffer = -1;
  }
}

/*********************************************************************
*
*       LCD_X_DisplayDriver
*/
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * p) {
  switch (Cmd) {
  ...
  case LCD_X_SHOWBUFFER: {
      LCD_X_SHOWBUFFER_INFO * pData;

      pData = (LCD_X_SHOWBUFFER_INFO *)p;
      _PendingBuffer = pData->Index;
    }
    break;
  ...
  }
  return 0;
}

/*********************************************************************
*
*       LCD_X_Config
*/
void LCD_X_Config(void) {
  GUI_MULTIBUF_Config(NUM_BUFFERS);
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  LCD_SetDevFunc(0, LCD_DEVFUNC_COPYBUFFER, (void (*)())_LCD_CopyBuffer);
  LCD_SetSizeEx (0, LCD_XSIZE,  LCD_YSIZE);
  LCD_SetVSizeEx(0, LCD_VXSIZE, LCD_VYSIZE);
  LCD_SetVRAMAddrEx(0, (void *)((U32)&_aVRAM[0]));
}
```

Configuration - Display orientation

- **Display driver selection (recommended)**

  Some drivers like `GUDRV_Lin` consists of different modules for certain display orientations.

- **Display driver configuration (recommended)**

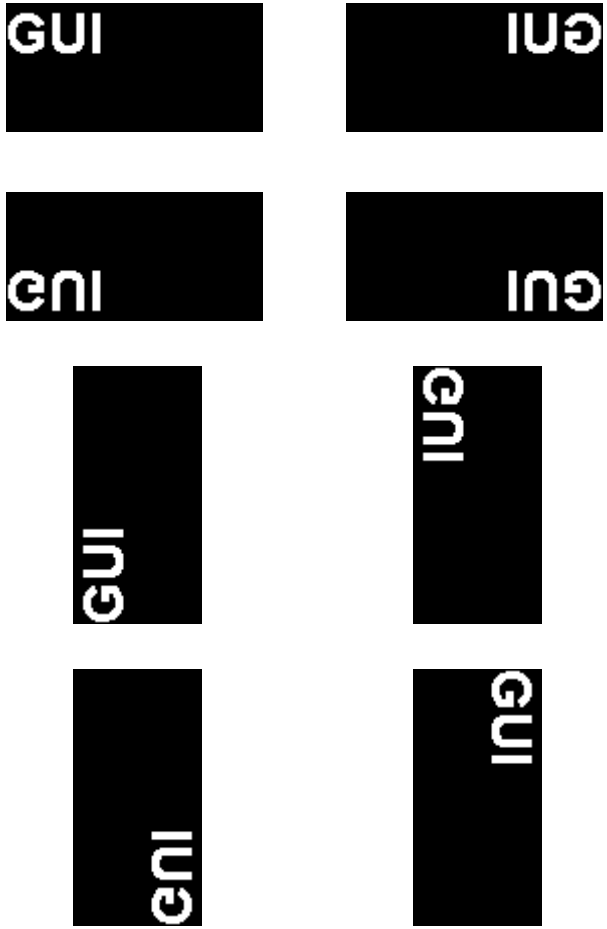
  Other drivers like the `GUIDRV_FlexColor` contain configuration functions for setting the orientation.

- **Orientation device (less performance, more RAM load)**

  Orientation device can simply be used by `GUI_SetOrientation()`.

- **Touch orientation**

  Can be determined by calling the function `GUI_TOUCH_SetOrientation()`.



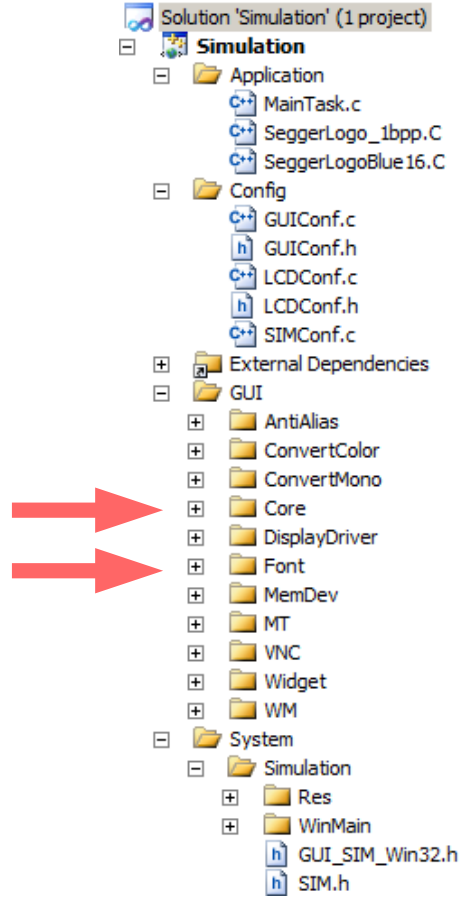
Configuration summary

You should now know the following:

- **How to achieve runtime configuration of memory usage**
- **How to achieve compile time configuration of GUI features**
- **How to set up the display drivers for the layers**
- **The advantage of tripple buffering in comparison to double buffering**
- **The most recommended way for setting up the display orientation**

Core functions - emWin BASE

- Image file support for BMP, GIF and JPEG
- Drawing of images and fonts from non addressable media
- LTR, RTL and bidirectional text support
- Software alpha blending
- Sprites and cursors, animated
- Drawing primitives
- Support for non antialiased font formats
- Drawing of text and values (dec, bin, hex, float)
- Multiple buffering
- Animations
- Drawing of QR codes
- Touch screen support
- Language support
- Standard font package (ASCII and ISO 8859-1)



Core functions - Drawing image files

- **BMP file support**

  No decompression required

- **GIF file support**

  Requires app. 16KByte RAM for decompression

- **JPEG file support**

  Requires app. 33KByte + xSize * 80 bytes

- **PNG file support** (free available on www.segger.com)

  Requires app. 21KByte + (xSize * ySize) * 4

- **DTA file support**

  Streamed bitmaps of bitmap converter

- **Drawing from non addressable areas**

  _GetData() functions are required

```
/*********************************************************************
*
*       _GetData0: BMP, JPEG, GIF
*/
static int _GetData0(void * p, const U8 ** ppData, unsigned NumBytesReq, U32 Off) {
  static char acBuffer[0x200];
  HANDLE    * phFile;
  DWORD       NumBytesRead;

  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;
}

/*********************************************************************
*
*       _GetData1: DTA, PNG
*/
static int _GetData1(void * p, const U8 ** ppData, unsigned NumBytesReq, U32 Off) {
  HANDLE * phFile;
  DWORD    NumBytesRead;
  U8     * pData;

  pData  = (U8 *)*ppData;
  phFile = (HANDLE *)p;
  //
  // Set file pointer to the required position
  //
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  //
  // Read data into buffer
  //
  ReadFile(*phFile, pData, NumBytesReq, &NumBytesRead, NULL);
  //
  // Return number of available bytes
  //
  return NumBytesRead;
}
```

Core functions - Bidirectional text

- UTF8 encoding needs to be enabled

  Should be done by `GUI_UC_SetEncodeUTF8()`.

- Bidirectional text needs to be enabled

  Should be done by `GUI_UC_EnableBIDI()`.

- Font file needs to be available

  The FontConverter or TTF support is required because emWin does not contain Arabic or Hebrew fonts.

- Once enabled all text based functions support bidirectional text

Note: Bidirectional text support for the complete 'Basic Multilingual Plane' multisupport requires app. 60 Kbyte additional ROM.

```
GUI_CONST_STORAGE GUI_FONT GUI_FontArabic24 = {…};

static char * _apText[] = {
  { "\nBidirectional text\n\n\xd8\xb9\xd9\x84\xd8\xa7 1, 2, 345 "
    "\xd8\xba\xd9\x86\xd9\x8a XYZ \xd8\xa3\xd9\x86\xd8\xa7"},
  { "\nBeautiful\n\n\xd8\xac\xd9\x80\xd9\x85\xd9\x8a\xd9\x84"},
  { "\nI'm from Lebanon.\n\n\xd8\xa3\xd9\x86\xd8\xa7 \xd9\x85"
    "\xd9\x86 \xd9\x84\xd8\xa8\xd9\x86\xd8\xa7\xd9\x86"},
  { "\nI'm from Canada.\n\n\xd8\xa3\xd9\x86\xd8\xa7 \xd9\x85"
    "\xd9\x86 \xd9\x83\xd9\x86\xd8\xaf\xd8\xa7"},
  { "\nIsn't it like that?\n\n\xd8\xa3\xd9\x84\xd9\x8a\xd8"
    "\xb3 \xd9\x83\xd8\xb0\xd8\xa7\xd9\x84\xd9\x83\xd8\x9f"},
  { "\nDo you work?\n\n\xd9\x87\xd9\x84 \xd8\xaa\xd8\xb9\xd9"
    "\x85\xd9\x84\xd8\x9f"},
  { "\nThe book is heavy.\n\n\xd8\xa7\xd9\x84\xd9\x83\xd8\xaa"
    "\xd8\xa7\xd8\xa8 \xd8\xab\xd9\x82\xd9\x8a\xd9\x84"},
};

static void _ShowArabicTextSamples(void) {
  GUI_RECT Rect = {40, 60, 279, 199};
  GUI_SetFont(&GUI_FontArabic24); /* Set Arabic font */
  while (1) {
    int i;
    GUI_SetColor(GUI_RED);
    GUI_DrawRect(Rect.x0 - 1, Rect.y0 - 1, Rect.x1 + 1, Rect.y1 + 1);
    GUI_SetColor(GUI_WHITE);
    for (i = 0; i < GUI_COUNTOF(_apText); i++) {
      GUI_DispStringInRectWrap(_apText[i], &Rect, GUI_TA_HCENTER, GUI_WRAPMODE_WORD);
      GUI_Delay(2000);
      GUI_ClearRectEx(&Rect);
    }
  }
}

void MainTask(void) {
  GUI_Init();
  GUI_UC_SetEncodeUTF8();         /* Enable UTF8 decoding */
  GUI_UC_EnableBIDI(1);           /* Enable bidirectional text */
  GUI_SetFont(&GUI_FontArabic24); /* Select font with required characters */
  GUI_DispStringHCenterAt("Arabic language sample", 160, 5);
  _ShowArabicTextSamples();
}
```

Core functions - Alpha blending

Different capabilities of alpha blending:

- **Setting of alpha value**

  GUI_SetAlpha() can be used to set the alpha value for subsequent drawing operations.

- **Automatic alpha blending**

  GUI_EnableAlpha() can be used to enable automatic alpha blending. The upper 8 bits of the current color are then used for alpha blending. Should be disabled after use.

- **PNG file support**

  GUI_PNG_Draw() can be used to draw PNG files with alpha value.

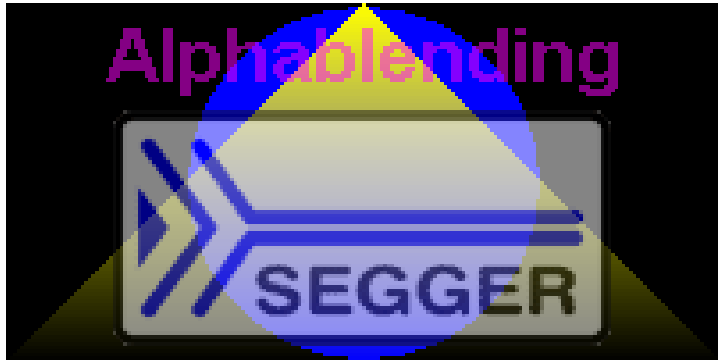
- **True color bitmaps with alpha channel**

  BitmapConverter supports conversion of PNG files to 32bpp bitmaps with alpha channel.

- **Compressed alpha channel bitmaps**

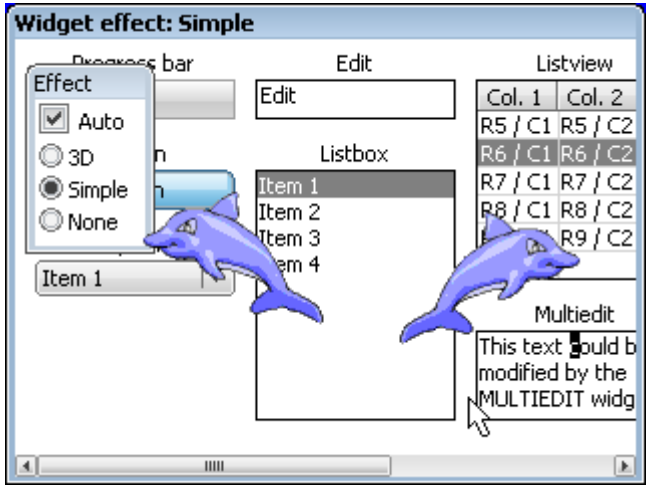
  Uses the alpha mask to draw the image in the current color.

```
/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  U32 Alpha;
  int i;

  GUI_Init();
  GUI_EnableAlpha(1);
  GUI_SetColor(GUI_BLUE);
  GUI_FillCircle(100, 50, 49);
  for (i = 0; i < 100; i++) {
    Alpha = (i * 255 / 100) << 24;
    GUI_SetColor(GUI_YELLOW | Alpha);
    GUI_DrawHLine(i, 100 - i, 100 + i);
  }
  GUI_SetAlpha(0x80);
  GUI_DrawBitmap(&_LogoBitmap, 30, 30);
  GUI_SetAlpha(0);
  GUI_SetColor(GUI_MAGENTA | 0x80000000);
  GUI_SetFont(&GUI_Font24B_ASCII);
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_DispStringHCenterAt("Alphablending", 100, 3);
  GUI_EnableAlpha(0);
  while (1) {
    GUI_Delay(100);
  }
}
```



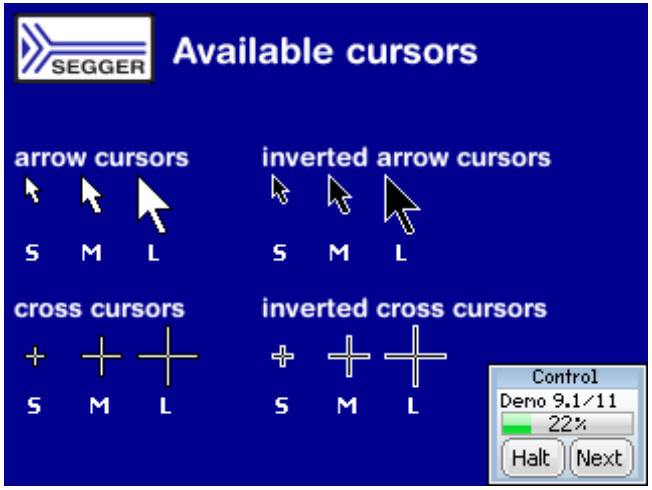
Software sprites

- Index based bitmaps and **alpha bitmaps** can be used
- Sprites manage the background automatically
- GIF animations could be used for **animated sprites**



Cursors

- Cursors are based on sprites
- Window manager automatically manages position
- GIF animations could be used for **animated cursors**



Core functions - Animated Sprites and Cursors

Cursors can be animated:

- **GUI_CURSOR_SetAnim()**

  BitmapConverter automatically converts animated GIFs to animated cursors.

Sprites can be animated:

- **GUI_SPRITE_CreateAnim()**

  BitmapConverter automatically converts animated GIFs to animated cursors.

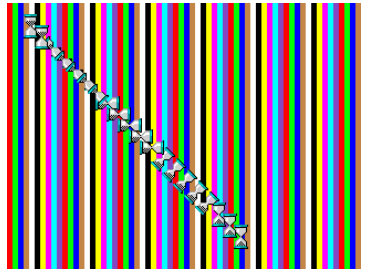
Note: All subimages must have the same size.

Animated Cursor

```
GUI_BITMAP const * _apbmHourGlassM[] = {
  ...
}

static void _DrawStripes(void) {
  ...
}

void MainTask(void) {
  GUI_Init();
  _DrawStripes();
  GUI_CURSOR_SetAnimEx(_apbmHourGlassM, 11, 11, 50, NULL, GUI_COUNTOF(_apbmHourGlassM), 0);
  GUI_CURSOR_SetPosition(100, 100);
  while (1) {
    GUI_Delay(100);
  }
}
```

Animated Sprites

```
GUI_BITMAP const * _apbmHourGlassM[] = {
  ...
}

static void _DrawStripes(void) {
  ...
}

void MainTask(void) {
  int i;
  GUI_HSPRITE ahSprite[20];

  GUI_Init();
  _DrawStripes();
  for (i = 0; i < GUI_COUNTOF(ahSprite); i++) {
    ahSprite[i] = GUI_SPRITE_CreateAnim(_apBM, 10 + i * 10, 10 + i * 10, 50, NULL, GUI_COUNTOF(_apBM));
    GUI_Delay(5);
  }
  while (1) {
    GUI_Delay(100);
  }
}
```



SAMPLES: AnimatedSprites\MainTask_AnimatedCursor.c
AnimatedSprites\MainTask_AnimatedSprites.c

Core functions - Drawing QR-codes

Basic routines for QR (Quick Response) codes are available.

Steps to show a QR code:

- **Create** QR code bitmap
- **Draw** QR code bitmap
- **Delete** QR code bitmap (if no longer used)

Terms:

- **Error correction**
  QR codes contain Reed-Solomon code blocks for error correction. The higher the level, the better the correction.
- **Module**
  One 'pixel' is called a Module.
- **Version**
  QR code capacity depends on version (1-40).
  Version could be calculated automatically.

Note: emWin supports QR codes in byte encoding.

```
#include "GUI.h"

#define PIXEL_SIZE 4

static const char sText[] = "www.segger.com";

void MainTask(void) {
  GUI_HMEM hQR;

  GUI_Init();
  hQR = GUI_QR_Create(sText, PIXEL_SIZE, GUI_QR_ECLEVEL_H, 0);
  GUI_QR_Draw(hQR, 50, 50);
  GUI_QR_Delete(hQR);
  while (1) {
    GUI_Delay(100);
  }
}
```

```
Size of bitmap:    100
Number of modules: 25
Version:           2
```

Core functions - Text and language resource files

Text could be 'outsourced' from code to text files. Language support API could be used to access these files.

- Use of **different languages.**
- Files could reside in **RAM or any non addressable medium** like a file system accessed by a **GetData()** function.
- TXT and CSV files are supported.

If system is **low on RAM**:

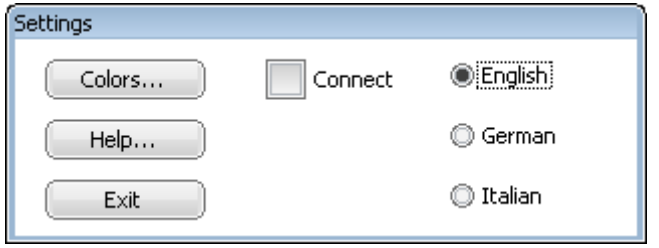
- **GUI_LANG_GetTextBuffered..()** copies the text into the given buffer.

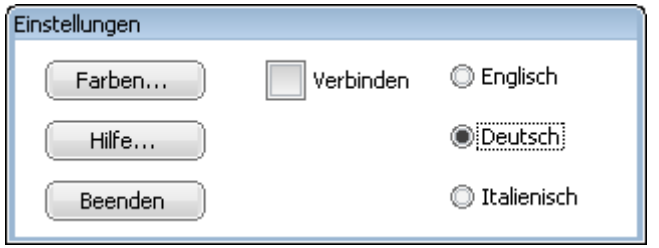
  Advantage: RAM usage is constant.
  Disadvantage: Buffer managment overhead.

If system has **enough RAM** for the used text:

- **GUI_LANG_GetText..()** returns a pointer to the requested string which remains in RAM.

  Advantage: Text pointers remain valid
  Disadvantage: Text usage increases RAM usage.





Core functions - Text and language resource file format

Different formats are supported:

- **TXT file support**
  One language per file. Multiple TXT files can be used simultaneously.
- **CSV file support**
  Multiple languages in one file. Only one CSV file can be used simultaneously.

TXT file format

```
日本語
emWin
は様々な
言語に
対応しています
```

CSV file format

```
"Deutsch","English"
"emWin","emWin"
"unterstÃ¼tzt","supports"
"verschiedene","different"
"Sprachen","languages"
```

```
#include "GUI.h"

#define GERMAN  0
#define ENGLISH 1

static int _GetData(void * pVoid, const U8 ** ppData, unsigned NumBytes, U32 Off) {
  ...
}

static char * _pFileName = "GUI_LANG_CSV_Ger_Eng.csv";

void MainTask(void) {
  int NumLanguages, i, Language, NumItems;
  const char * pString;

  GUI_Init();
  //
  // Enable UTF8 encoding (if required)
  //
  GUI_UC_SetEncodeUTF8();
  //
  // Read CSV file
  //
  NumLanguages = GUI_LANG_LoadCSVEx(_GetData, _pFileName);
  //
  // Set language
  //
  Language = GERMAN;
  GUI_LANG_SetLang(Language);
  GUI_LANG_GetNumItems(Language);
  NumItems = GUI_LANG_GetNumItems(GERMAN);
  //
  // Draw all
  //
  for (i = 0; i < NumItems; i++) {
    pString = GUI_LANG_GetText(i);
    GUI_DispString(pString);
    GUI_DispNextLine();
  }
}
```

emWin animations support multiple independend animated items.

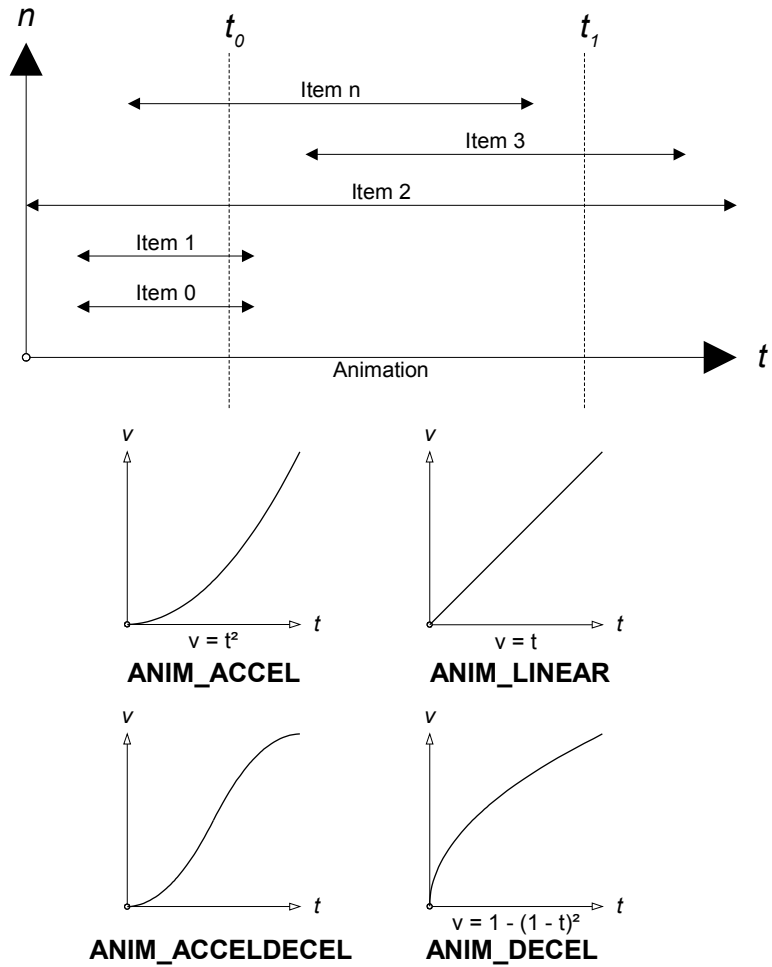
- Multiple items on the timeline
- Each item has its own period

Within the period of an item emWin calculates a position value.

- Predefined position calculations available
- Custom position calculation possible

Using animations:

- **Create** animation **object** with GUI_ANIM_Create()
- **Add items** with GUI_ANIM_AddItem()
- **Start animation** with GUI_ANIM_Start()
- **Call GUI_ANIM_Exec()** periodically and check **return code**
- If return code is 1 the animation is at the end. Now it could be restarted with GUI_ANIM_Start() or deleted with GUI_ANIM_Delete()



Core functions summary

You should now know the following:

- **The character sets are included in the standard fonts shipped with emWin**
- **How to achieve BiDi support in the application**
- **The meaning of 'Automatic alpha blending' ant how to use it**
- **How to achieve drawing of animated sprites**
- **The difference between GUI_LANG_GetText() and GUI_LANG_GetTextBuffered()**
- **How to use animations with emWin**

How do they work?

Drawing operations can be passed to a memory device instead to the display. A memory device is a hardware independent destination device for drawing operations.

What can they be used for?

- Preventing flickering
- Container for decompressed images
- Scaling and rotating
- Fading operations
- Window animations
- Transparency effects

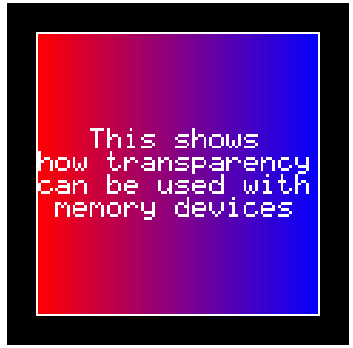
What means 'transparency' here?

Memory devices with transparency 'know' the pixels which have been accessed. One additional bit is required per pixel for storing this information. Supported by 1, 8 and 16bpp memory devices.

Do memory devices support alpha blending?

Yes, 32bpp memory devices support alpha blending, but not 'transparency'.

```
#include "GUI.h"

void MainTask(void) {
  GUI_MEMDEV_Handle hMem;
  GUI_RECT Rect = { 10, 10, 109, 109 };

  GUI_Init();
  hMem = GUI_MEMDEV_Create(Rect.x0, Rect.y0, Rect.x1 - Rect.x0 + 1, Rect.y1 - Rect.y0 + 1);
  GUI_DrawGradientH(Rect.x0, Rect.y0, Rect.x1, Rect.y1, GUI_RED, GUI_BLUE);
  GUI_MEMDEV_Select(hMem);
  GUI_DrawRectEx(&Rect);
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_DispStringInRect("This shows\n"
                       "how transparency\n"
                       "can be used with\n"
                       "memory devices"
                       , &Rect
                       , GUI_TA_HCENTER | GUI_TA_VCENTER);
  GUI_MEMDEV_Select(0);
  GUI_MEMDEV_Write(hMem);
  while (1) {
    GUI_Delay(100);
  }
}
```



Memory devices - Using as image container

If drawing of the same image **is required multiple times** and if enough memory is available, using memory devices as image container offers the following advantages:

- Decompression only required one time
- Alpha blending support (32bpp memory devices)
- Fast drawing operations

```
#include <windows.h> // Required only for file handling

#include "GUI.h"

/*********************************************************************
*
*       _GetData1: DTA, PNG
*/
static int _GetData1(void * p, const U8 ** ppData, unsigned NumBytesReq, U32 Off) {
  HANDLE * phFile;
  DWORD    NumBytesRead;
  U8     * pData;

  pData  = (U8 *)*ppData;
  phFile = (HANDLE *)p;
  //
  // Set file pointer to the required position
  //
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  //
  // Read data into buffer
  //
  ReadFile(*phFile, pData, NumBytesReq, &NumBytesRead, NULL);
  //
  // Return number of available bytes
  //
  return NumBytesRead;
}


/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  GUI_MEMDEV_Handle hMem;
  HANDLE hFile;

  GUI_Init();
  GUI_DrawGradientV(0, 0, 319, 239, GUI_BLUE, 0);
  hMem = GUI_MEMDEV_CreateFixed(0, 0, 162, 150,
                                GUI_MEMDEV_NOTRANS, GUI_MEMDEV_APILIST_32, GUICC_8888);
  hFile = CreateFile("IMAGE.png", GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  GUI_MEMDEV_Select(hMem);
  GUI_SetBkColor(GUI_TRANSPARENT);
  GUI_Clear();
  GUI_PNG_DrawEx(_GetData1, (void *)&hFile, 0, 0);
  GUI_MEMDEV_Select(0);
  CloseHandle(hFile);
  GUI_MEMDEV_WriteAt(hMem, 0, 0);
  GUI_MEMDEV_WriteAt(hMem, 160, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```

Memory devices - Scaling and rotating images

A set of functions optimized for several purposes are available:

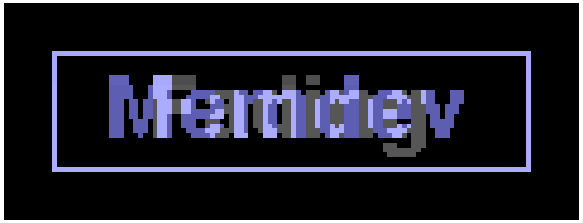
- **GUI_MEMDEV_Rotate()**
  Fast routine using 'nearest neighbour' method
- **GUI_MEMDEV_RotateAlpha()**
  Fast routine using with additional alpha value
- **GUI_MEMDEV_RotateHQ()**
  Uses high quality method (bilinear)
- **GUI_MEMDEV_RotateHQAlpha()**
  Uses high quality method with additional alpha value
- **GUI_MEMDEV_RotateHQHR()**
  Uses high quality method and high resolution
- **GUI_MEMDEV_RotateHQT()**
  Uses high quality method optimized for images with a large amount of transparent pixels

```
#include <windows.h>

#include "GUI.h"

#define TIME_MIN 20

static int _GetData1(void * p, const U8 ** ppData, unsigned NumBytesReq, U32 Off) {
  HANDLE * phFile;
  DWORD    NumBytesRead;
  U8     * pData;

  pData  = (U8 *)*ppData;
  phFile = (HANDLE *)p;
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  ReadFile(*phFile, pData, NumBytesReq, &NumBytesRead, NULL);
  return NumBytesRead;
}

void MainTask(void) {
  GUI_MEMDEV_Handle hMemImage, hMemWork, hMemBk;
  HANDLE hFile;
  I32 a1000, Add, TimeStart, Time0, TimeUsed;

  GUI_Init();
  GUI_DrawGradientV(0, 0, 319, 239, GUI_WHITE, GUI_GRAY);
  hMemImage = GUI_MEMDEV_CreateFixed(10, 10, 50, 50,
                                     GUI_MEMDEV_NOTRANS, GUI_MEMDEV_APILIST_32, GUICC_8888);
  hMemWork  = GUI_MEMDEV_CreateFixed(10, 10, 50, 50,
                                     GUI_MEMDEV_NOTRANS, GUI_MEMDEV_APILIST_32, GUICC_8888);
  hMemBk    = GUI_MEMDEV_CreateFixed(10, 10, 50, 50,
                                     GUI_MEMDEV_NOTRANS, GUI_MEMDEV_APILIST_32, GUICC_8888);
  hFile = CreateFile("IMAGE.png", GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  GUI_MEMDEV_Select(hMemImage);
  GUI_SetBkColor(GUI_TRANSPARENT);
  GUI_Clear();
  GUI_PNG_DrawEx(_GetData1, (void *)&hFile, 10, 10);
  GUI_MEMDEV_Select(0);
  CloseHandle(hFile);
  GUI_MEMDEV_CopyFromLCD(hMemBk);
  a1000 = Add = 0;
  TimeStart = GUI_GetTime();
  while (1) {
    Time0 = GUI_GetTime();
    GUI_MEMDEV_Select(hMemWork);
    GUI_MEMDEV_Write(hMemBk);
    GUI_MEMDEV_RotateHQ(hMemImage, hMemWork, 0, 0, -a1000, 1000); // High quality
    a1000 = (GUI_GetTime() - TimeStart) * 90 - Add;
    if (a1000 > 360000) {
      Add   += 360000;
      a1000 -= 360000;
    }
    GUI_MEMDEV_CopyToLCD(hMemWork);
    TimeUsed = GUI_GetTime() - Time0;
    if (TIME_MIN > TimeUsed) {
      GUI_X_Delay(TIME_MIN - TimeUsed);
    }
  }
}
```

Memory devices - Fading and animation functions

Animations can be used to inject some life into the application. They will always help to let the users eye smoothly capture what happens in the application.

- Automatic fading function available
- Animation functions require window manager



```
#include "GUI.h"

void MainTask(void) {
  GUI_MEMDEV_Handle hMem0, hMem1;
  GUI_RECT Rect = { 10, 10, 109, 34 };

  GUI_Init();
  hMem0 = GUI_MEMDEV_CreateFixed(Rect.x0, Rect.y0,
                                 Rect.x1 - Rect.x0 + 1, Rect.y1 - Rect.y0 + 1,
                                 GUI_MEMDEV_NOTRANS, GUI_MEMDEV_APILIST_32, GUICC_8888);
  hMem1 = GUI_MEMDEV_CreateFixed(Rect.x0, Rect.y0,
                                 Rect.x1 - Rect.x0 + 1, Rect.y1 - Rect.y0 + 1,
                                 GUI_MEMDEV_NOTRANS, GUI_MEMDEV_APILIST_32, GUICC_8888);
  GUI_SetFont(GUI_FONT_20B_ASCII);
  GUI_MEMDEV_Select(hMem0);
  GUI_SetColor(GUI_LIGHTBLUE);
  GUI_DrawRectEx(&Rect);
  GUI_DispStringInRect("Memdev", &Rect, GUI_TA_HCENTER | GUI_TA_VCENTER);
  GUI_MEMDEV_Select(hMem1);
  GUI_SetColor(GUI_WHITE);
  GUI_DrawRectEx(&Rect);
  GUI_DispStringInRect("Fading", &Rect, GUI_TA_HCENTER | GUI_TA_VCENTER);
  while (1) {
    GUI_MEMDEV_FadeDevices(hMem0, hMem1, 500);
    GUI_MEMDEV_FadeDevices(hMem1, hMem0, 500);
  }
}

#include <stddef.h>

#include "DIALOG.H"

void _cbBk(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_DrawGradientV(0, 0, 319, 239, GUI_BLUE, GUI_MAGENTA);
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  WM_SetCallback(WM_HBKWIN, _cbBk);
  WM_Exec();
  FRAMEWIN_SetDefaultSkin(FRAMEWIN_SKIN_FLEX);
  FRAMEWIN_SetDefaultFont(GUI_FONT_20B_ASCII);
  FRAMEWIN_SetDefaultTextColor(FRAMEWIN_CI_ACTIVE,   GUI_DARKGRAY);
  FRAMEWIN_SetDefaultTextColor(FRAMEWIN_CI_INACTIVE, GUI_DARKGRAY);
  FRAMEWIN_SetDefaultTextAlign(GUI_TA_HCENTER);
  hWin = FRAMEWIN_CreateEx(60, 60, 200, 120, WM_HBKWIN, WM_CF_SHOW, 0, 0, "FRAMEWIN", NULL);
  while (1) {
    GUI_MEMDEV_FadeInWindow  (hWin, 400);
    GUI_Delay(400);
    GUI_MEMDEV_FadeOutWindow (hWin, 400);
    GUI_MEMDEV_MoveInWindow  (hWin,   0, 240,  45, 400);
    GUI_Delay(400);
    GUI_MEMDEV_MoveOutWindow (hWin, 320, 240, -45, 400);
    GUI_MEMDEV_ShiftInWindow (hWin, 400, GUI_MEMDEV_EDGE_TOP);
    GUI_Delay(400);
    GUI_MEMDEV_ShiftOutWindow(hWin, 400, GUI_MEMDEV_EDGE_BOTTOM);
  }
}
```

SAMPLES: MemoryDevices\MainTask_MemDevFading.c
MemoryDevices\MainTask_NestedModal.c

Memory devices summary

You should now know the following:

- **The difference between 'transparency' and alpha blending in the context of memory devices**
- **How to avoid multiple decompression when showing compressed images**
- **How to achieve image rotation and how to optimize it for different purposes**

Antialiasing

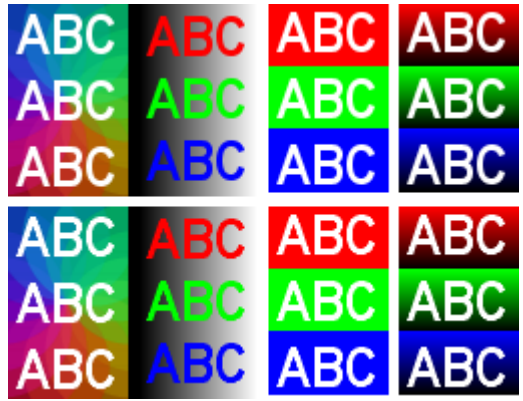
Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground.

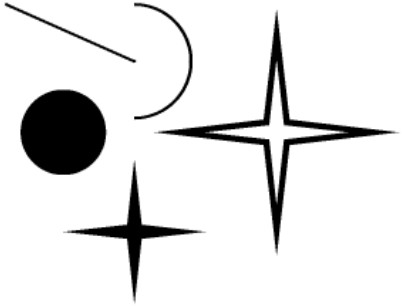
emWin supports antialiased drawing of

- **Text**
  Font converter is required for creating AA fonts.
- **Arcs**
  `GUI_AA_DrawArc()`
- **Circles**
  `GUI_AA_FillCircle()`
- **Lines**
  `GUI_AA_DrawLine()`
- **Polygons**
  `GUI_AA_DrawPolyOutline()`
  `GUI_AA_FillPolygon()`

Note: Performance of antialiased drawing operations degrades significantly in comparison to non antialiased drawing operations.

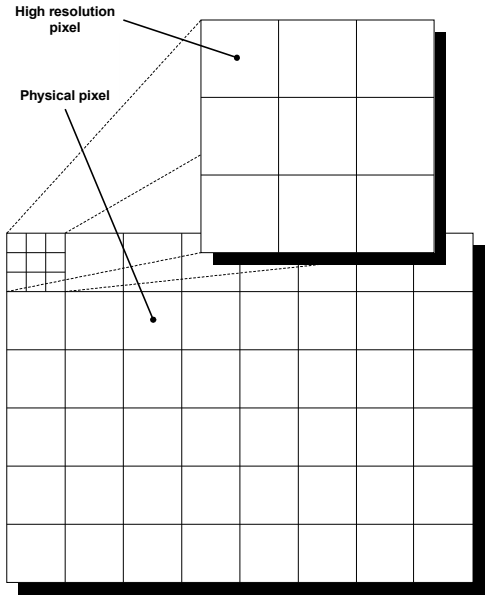
```
#include "GUI.h"

static GUI_POINT _aPoint[] = {
  {  -5,  -5 },  {   0, -50 },  {   5,  -5 },  {  50,   0 },
  {   5,   5 },  {   0,  50 },  {  -5,   5 },  { -50,   0 },
};

void MainTask(void) {
  GUI_Init();
  GUI_SetBkColor(GUI_WHITE);
  GUI_SetColor(GUI_BLACK);
  GUI_Clear();
  GUI_SetPenSize(2);
  GUI_AA_DrawLine(10, 10, 100, 50);
  GUI_AA_DrawArc(100, 50, 40, 40, 270, 450);
  GUI_AA_FillCircle(50, 100, 30);
  GUI_AA_DrawPolyOutline(_aPoint, GUI_COUNTOF(_aPoint), 4, 200, 100);
  GUI_AA_FillPolygon(_aPoint, GUI_COUNTOF(_aPoint), 100, 170);
  while (1) {
    GUI_Delay(100);
  }
}
```





Antialiasing - High resolution antialiasing

High resolution antialiasing of emWin uses the virtual space determined by the antialiasing factor and the display size. The advantage is that items can be placed not only at physical positions of your display but also "between" them.



Note: The higher the factor the better the quality. Factor should be between 2 and 8.

```
#include "GUI.h"

#define TIME_MIN         20
#define TIME_PER_TURN 40000
#define DURATION           (TIME_PER_TURN / 16)

typedef struct {
  void (* pFunc)(const GUI_POINT * pPoints, int NumPoints, int x0, int y0);
  int Factor;
} PARA;

static GUI_POINT _aPoint[] = { ... };

static void _MagnifyPoints(const GUI_POINT * pPointSrc, GUI_POINT * pPointDest, int NumPoints, int Factor) {
  int i;
  for (i = 0; i < NumPoints; i++) {
    (pPointDest + i)->x = (pPointSrc + i)->x * Factor;
    (pPointDest + i)->y = (pPointSrc + i)->y * Factor;
  }
}

static void _Loop(void (* pFunc)(const GUI_POINT * pPoints,
                  int NumPoints, int x0, int y0), int Factor, GUI_MEMDEV_Handle hMem, int x, int y) {
  GUI_POINT aPoint   [GUI_COUNTOF(_aPoint)];
  GUI_POINT aPointRot[GUI_COUNTOF(_aPoint)];
  int TimeStart, Time0, TimeUsed;
  float a;
  GUI_AA_SetFactor(Factor);
  _MagnifyPoints(_aPoint, aPoint, GUI_COUNTOF(_aPoint), Factor);
  TimeStart = GUI_GetTime();
  do {
    Time0 = GUI_GetTime();
    a = (float)(Time0 - TimeStart) / TIME_PER_TURN * 3.1415f * 2;
    GUI_RotatePolygon(aPointRot, aPoint, GUI_COUNTOF(_aPoint), a);
    GUI_Clear();
    GUI_DispStringHCenterAt("Factor: ", x - 5, 10);
    GUI_DispDecMin(Factor);
    pFunc(aPointRot, GUI_COUNTOF(_aPoint), x * Factor, y * Factor);
    GUI_MEMDEV_CopyToLCD(hMem);
    TimeUsed = GUI_GetTime() - Time0;
  } while ((Time0 - TimeStart) < DURATION);
}

void MainTask(void) {
  int xSize, ySize, i;
  GUI_MEMDEV_Handle hMem;
  PARA aPara[] = {
    {(void (*)(const GUI_POINT * pPoints, int NumPoints, int x0, int y0))GUI_FillPolygon,    1},
    {(void (*)(const GUI_POINT * pPoints, int NumPoints, int x0, int y0))GUI_AA_FillPolygon, 2},
    {(void (*)(const GUI_POINT * pPoints, int NumPoints, int x0, int y0))GUI_AA_FillPolygon, 4},
    {(void (*)(const GUI_POINT * pPoints, int NumPoints, int x0, int y0))GUI_AA_FillPolygon, 8},
  };
  GUI_Init();
  xSize = LCD_GetXSize();
  ySize = LCD_GetYSize();
  hMem = GUI_MEMDEV_Create(0, 0, xSize, ySize);
  GUI_MEMDEV_Select(hMem);
  GUI_AA_EnableHiRes();
  while (1) {
    for (i = 0; i < GUI_COUNTOF(aPara); i++) {
      _Loop(aPara[i].pFunc, aPara[i].Factor, hMem, xSize / 2, ySize / 2);
    }
  }
}
```

Antialiasing summary

You should now know the following:

- **The difference between antialiasing and high resolution antialiasing**
- **Drawing of antialiased primitives degrades performance**

Window manager

The **W**indow **M**anager:

- **Management system for a hierarchic window structure**

  Each layer has its own desktop window. Each desktop window has its own hierarchic tree of child windows.

- **Callback mechanism based system**

  Communication is based on an event driven callback mechanism. **All** drawing operations should be done within the `WM_PAINT` event.

- **Foundation of widget library**

  All widgets are based on the functions of the WM.

Basic capabilities:

- **Automatic clipping**
- **Automatic use of multiple buffers**
- **Automatic use of memory devices**
- **Automatic use of display driver cache**
- **Motion support**

```
#include "WM.h"

void _cbWin(WM_MESSAGE * pMsg) {
  int xSize, ySize;

  switch (pMsg->MsgId) {
  case WM_PAINT:
    xSize = WM_GetWindowSizeX(pMsg->hWin);
    ySize = WM_GetWindowSizeY(pMsg->hWin);
    GUI_Clear();
    GUI_DrawRect(0, 0, xSize - 1, ySize - 1);
    GUI_DispStringHCenterAt("Window", xSize / 2, 10);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void _cbBk(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_DrawGradientV(0, 0, 319, 239, GUI_BLUE, GUI_MAGENTA);
    break;
  default:
    WM_DefaultProc(pMsg);
    break;
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  WM_SetCallback(WM_HBKWIN, _cbBk);
  hWin = WM_CreateWindowAsChild(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW, _cbWin, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```

Window manager - Callback mechanism

Window management is based on a callback mechanism.

That requires a **callback routine** for each window which **should support at least** the following:

- **Painting**

  Each window has to draw itself. This should be done when receiving a `WM_PAINT` message.

- **Default message handling**

  Plain windows need to call the function `WM_DefaultProc()` to avoid undefined behavior of the window.

WM needs to 'stay alive'. This can be done within a simple loop after creating the windows. It has nothing to do but calling `GUI_Delay()` which does the following:

- **Window (re)drawing**
- **PID management**
- **Key input management**
- **Timer management**

```
#include "WM.h"

void _cbWin(WM_MESSAGE * pMsg) {
  int xSize, ySize;

  switch (pMsg->MsgId) {
  case WM_PAINT:
    xSize = WM_GetWindowSizeX(pMsg->hWin);
    ySize = WM_GetWindowSizeY(pMsg->hWin);
    GUI_Clear();
    GUI_DrawRect(0, 0, xSize - 1, ySize - 1);
    GUI_DispStringHCenterAt("Window", xSize / 2, 10);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void _cbBk(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_DrawGradientV(0, 0, 319, 239, GUI_BLUE, GUI_MAGENTA);
    break;
  default:
    WM_DefaultProc(pMsg);
    break;
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  WM_SetCallback(WM_HBKWIN, _cbBk);
  hWin = WM_CreateWindowAsChild(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW, _cbWin, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```

Window manager - Keyboard and PID input

Communication between the application and the user is mostly done by keyboard and/or **P**ointer **I**nput **D**evices. The following functions are available for that:

- **GUI_StoreKeyMsg()**

  If a keyboard event occurs (pressing or unpressing a key) it should be passed to this routine.

- **GUI_PID_StoreState()**

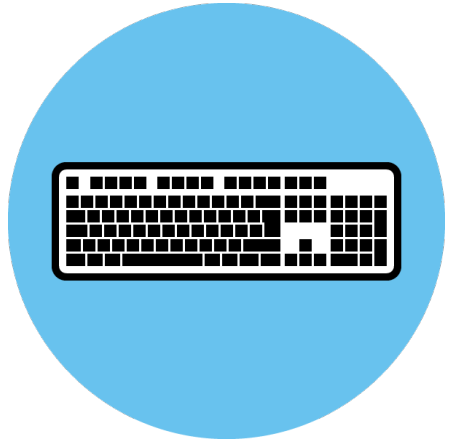
  If a PID event occurs (pressed, unpresed or moving) it should be passed to this routine.

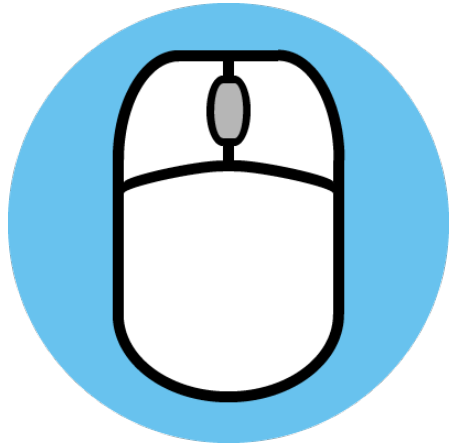
The WM automatically polls the keyboard and the PID buffer.

Keyboard input is passed to the **currently focussed window**.

PID input is passed to the **uppermost window of event position**.

Note: The above routines can also be called from within an interrupt or from a separate task without the multitasking option.





Invalidating instead of drawing

If things need to be redrawn the most recommended way is changing the values to be drawn immediately when the event occurs and invalidating the window (or a part of it). The WM sends `WM_PAINT` messages automatically to each invalid window.

- **WM_InvalidateWindow()**

  Invalidates the complete window. Easy and most often used for invalidation.

- **WM_InvalidateRect()**

  Invalidates only a rectangular part of the window.

Note: Invalidating transparent windows causes invalidation of the window behind to make sure the background is (re)drawn.

Note: It is highly recommended to draw things only within the WM_PAINT event.

```
#include "WM.h"

void _cbWin(WM_MESSAGE * pMsg) {
  int xSize, ySize;
  static int Value;
  WM_KEY_INFO * pInfo;

  switch (pMsg->MsgId) {
  case WM_PAINT:
    xSize = WM_GetWindowSizeX(pMsg->hWin);
    ySize = WM_GetWindowSizeY(pMsg->hWin);
    GUI_Clear();
    GUI_DrawRect(0, 0, xSize - 1, ySize - 1);
    GUI_DispStringHCenterAt("Window", xSize / 2, 10);
    GUI_GotoXY(xSize / 2, 50);
    GUI_SetTextAlign(GUI_TA_HCENTER);
    GUI_DispDecMin(Value);
    break;
  case WM_KEY:
    pInfo = (WM_KEY_INFO *)pMsg->Data.p;
    if (pInfo->PressedCnt) {
      switch (pInfo->Key) {
      case GUI_KEY_UP:
        Value++;
        break;
      case GUI_KEY_DOWN:
        Value--;
        break;
      }
      WM_InvalidateWindow(pMsg->hWin);
    }
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void _cbBk(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_DrawGradientV(0, 0, 319, 239, GUI_BLUE, GUI_MAGENTA);
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  WM_SetCallback(WM_HBKWIN, _cbBk);
  hWin = WM_CreateWindowAsChild(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW, _cbWin, 0);
  WM_SetFocus(hWin);
  while (1) {
    GUI_Delay(100);
  }
}
```

Window manager - Invalidation of transparent windows

The following should be observed with transparent windows:

- Invalidating a transparent window causes invalidation of the background window.
- Many widgets with skinning are transparent per default.
- Background invalidation is done to make sure, the content of the whole window is redrawn.
- The window manager invalidates as long as an opaque window is found.

If the **shape** of the window to be invalidated **remains constant**, the **background does not need to be invalidated**. That could be achieved by setting the flag `WM_CF_CONST_OUTLINE`.

The following functions are available:

- `WM_SetTransState()`
- `WM_CreateWindow...()`

Useful for example for a keyboard keys with constant outlines.

`WM_CF_CONST_OUTLINE`

Could avoid unnecessary invalidations of background windows

Window manager - Software timer

Timers can be used for triggering periodic or individual events.

- Unlimited number of timers
- Only one line of code required for creation

The callback routine of the window receives a `WM_TIMER` message after the period is expired.

For periodic use it should be restarted.

- **WM_TIMER**
  The message gets the timer handle in the pMsg->Data.v variable. It can be used to restart the timer.
- **WM_CreateTimer()**
  Used to create a timer.
- **WM_RestartTimer()**
  Should be used to restart the timer. The period can be changed.
- **WM_DeleteTimer()**
  Should be used to delete any unused timer.

Note: Timers should be deleted if they are not used anymore.

Create timer

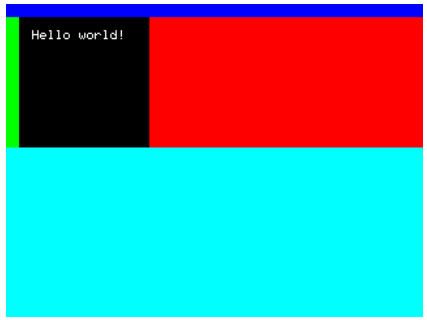
Receive WM_TIMER messages

Restart timer for periodic use

Delete timer if no longer used

Window manager - Tiling algorithm

Default behaviour of the WM is '**early clipping**'. That means clipping is done before WM_PAINT is send. An overlapped window then receives more than one paint message. It is drawn tile by tile:



Under certain circumstances it could make sense to use '**late clipping**'. In this case each drawing operation manages the clipping by itself. This could be done by using the flag WM_CF_LATE_CLIP.

Note: Normally early clipping provides the better performance.

The following messages are used:

- **WM_PRE_PAINT**
  Send before the first WM_PAINT message is send
- **WM_PAINT**
  One or more WM_PAINT messages for the drawing
- **WM_POSTPAINT**
  Send immediately after the last WM_PAINT message

```
#include "WM.h"

void _cbBk(WM_MESSAGE * pMsg) {
  GUI_COLOR aColor[] = { GUI_BLUE, GUI_GREEN, GUI_RED, GUI_CYAN, GUI_MAGENTA, GUI_YELLOW };
  static int Index;

  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(aColor[Index++]);
    if (Index == GUI_COUNTOF(aColor)) {
      Index = 0;
    }
    GUI_Clear();
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void _cbWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_Clear();
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  WM_SetCallback(WM_HBKWIN, _cbBk);
  hWin = WM_CreateWindowAsChild(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW, _cbWin, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```

Window manager - Preventing flickering effects

- **Multiple buffering**

  WM manages buffer handling automatically

- **Display driver cache**

  Drawing operations are performed in a cache.
  Gets handled by WM.

- **Memory devices**

  Drawing operations are redirected into memory devices.
  Devices are getting created/deleted automatically.

Note: Memory devices can only prevent flickering operations 'per window'. Each widget is a window and the progress of drawing the screen can be noticed anyhow.

Multiple buffering

Direct addressable frame buffer

Display driver cache

Indirect addressable frame buffer

Memory devices

Last choice

Window manager - Window animations

Window animation functions are available for the following effects:

- **Moving in and out**

  Any widget/window can be moved in/out from a determined position. The window will be enlarged/shrinked and moved/turned to its final position.

- **Shifting in and out**

  The given window is shifted in/out from/to a determined edge of the display.

- **Fading in and out**

  As the name implies the window the effect fades in/out the given window.

Note 1:	The functions require RAM for up to 3 whole screen memory devices (32bpp).
Note 2:	The animation functions are only available in conjunction with memory devices.
Note 3:	Recommended on fast systems only.

Window manager - Adding user data

- **WM_CreateWindowAsChild()**

  Last parameter defines the number of bytes for user data.

- **WM_SetUserData()**

  Adds user data to a window object.

- **WM_GetUserData()**

  Retrieves the data from the window object.

Note: User data allows custom defined object related data and helps to avoid global variables.

User data instead of global variables

```
#include "WM.h"

void _cbWin(WM_MESSAGE * pMsg) {
  int xSize, ySize;
  GUI_COLOR * pColor;

  switch (pMsg->MsgId) {
  case WM_PAINT:
    xSize = WM_GetWindowSizeX(pMsg->hWin);
    ySize = WM_GetWindowSizeY(pMsg->hWin);
    WM_GetUserData(pMsg->hWin, &pColor, sizeof(GUI_COLOR *));
    GUI_DrawGradientV(0, 0, xSize - 1, ySize - 1, *(pColor + 0), *(pColor + 1));
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

void MainTask(void) {
  WM_HWIN hWin;
  GUI_COLOR aColor[] = { GUI_MAGENTA, GUI_BLUE };
  GUI_COLOR * pColor = aColor;

  GUI_Init();
  WM_SetDesktopColor(GUI_BLACK);
  hWin = WM_CreateWindowAsChild(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW, _cbWin, sizeof(GUI_COLOR *));
  WM_SetUserData(hWin, &pColor, sizeof(GUI_COLOR *));
  while (1) {
    GUI_Delay(100);
  }
}
```

Window manager - Motion support

The content of a window (or the window itself) can simply be moved by wiping over the touch screen.

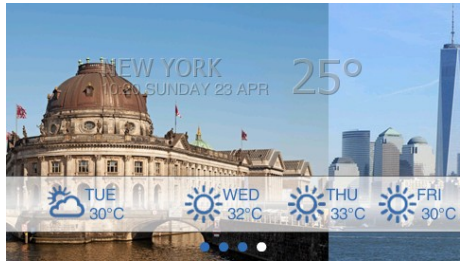
- Snapping supported
- Configurable motion properties
- Manual motion functions available

Enabling motion support for a window:

- Enable motion support with `WM_MOTION_Enable()`
- Create window with `WM_CF_MOTION_X/Y` flag

Detailed configuration can be done in the callback routine on `WM_MOTION_INIT` message.

WM sends `WM_MOTION` messages to be used for moving custom defined data.



...moving the window content

The most recommended and flexible way of use. Window position remains and only custom defined data is moved.

- Managing `WM_MOTION` messages required
- Custom defined drawing

...moving the window itself

The less recommended way of use. The window is moved automatically and the redrawing is done by the WM.

Window manager summary

You should now know the following:

- **How to keep the WM alive**
- **How to avoid unnecessary invalidation of transparent windows with constant shapes**
- **What exactly influences the number of WM_PAINT messages required for redrawing**
- **How to reduce the number of WM_PAINT messages if required**
- **The available methods to avoid flickering effects**
- **How to assign application defined data to a window object**
- **How to use motion support for moving windows automatically and for moving window content**
- **How to achieve keyboard- and PID-input managed and how it is managed by the WM**

Widget = **Wi**ndow + Ga**dget**

Each widget is a window with special behavior.

Window manager functions could be used with widgets.

Creating a widget can be done with one line of code. There are basically 2 ways of creating a widget:

- **Direct creation**

  For each widget there exist creation functions:

  - `<WIDGET>_CreateEx()`
    Creation without user data.
  - `<WIDGET>_CreateUser()`
    Creation with user data.

- **Indirect creation**

  Indirect means here using a dialog box creation function and a `GUI_WIDGET_CREATE_INFO` structure which contains a pointer to the indirect creation routine:

  - `<WIDGET>_CreateIndirect()`
    Creation by dialog box creation function.

Direct creation

```
void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  hWin = FRAMEWIN_CreateEx(10, 10, 100, 50, WM_HBKWIN, WM_CF_SHOW, 0, 0, "Window", NULL);
  while (1) {
    GUI_Delay(100);
  }
}
```

Indirect creation

```
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Window", 0, 10, 10, 100, 50 }
};

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  hWin = GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), NULL, 0, 0, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```

Button, Checkbox, Dropdown, Edit, Framewin, Graph, Header, Iconview, Image, Knob, Listbox, Listview, Listwheel, Menu, Multiedit, Multipage, Progbar, Radio, Scrollbar, Slider, Spinbox, Swipelist, Text, Treeview, Window

Widget library - Extended widget initialisation

Range of parameters of the creation functions is limited.

Widgets often need further initialization.

The WM sends a message when using indirect creation:

- **WM_INIT_DIALOG**

  Send immediately to the client window of the dialog after it has been created.

Within that event the function `WM_GetDialogItem()` can be used to get the handle of a child widget.

Independent of this message the widget properties can also be modified at any other point of code.

Note: This message is only available in conjunction with indirect creation functions

Initialization

```
#include "DIALOG.h"

static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Window", 0,             10, 10, 100, 50 },
  { TEXT_CreateIndirect,     "Text",   GUI_ID_TEXT0, 10, 10,  50, 20 },
};

void _cbFrame(WM_MESSAGE * pMsg) {
  WM_HWIN hItem;

  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    hItem = WM_GetDialogItem(pMsg->hWin, GUI_ID_TEXT0);
    TEXT_SetFont(hItem, GUI_FONT_20F_ASCII);
    TEXT_SetTextColor(hItem, GUI_GREEN);
    break;
  }
}

void MainTask(void) {
  WM_HWIN hWin;

  GUI_Init();
  hWin = GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), _cbFrame, 0, 0, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```

Widget library - Transparent widgets

A non transparent window has to draw its complete window area. A transparent window lets the background shine through.

To make a widget looking (semi)transparent the transparency flag need to be set. This can be done by the following ways:

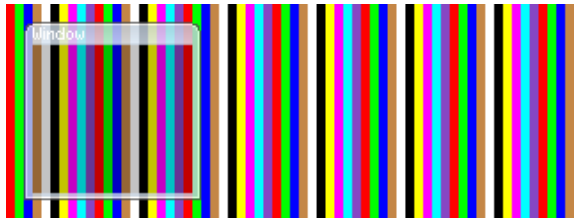
- **WM_CF_HASTRANS**

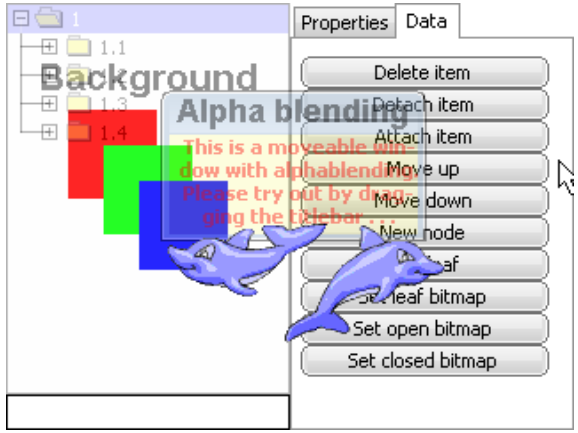
  This transparency flag can be used when creating the window.

- **WM_SetHasTrans()**

  This function can be used after the window has been created

Note: Invalidating a transparent window causes also redrawing of the window behind the transparent window

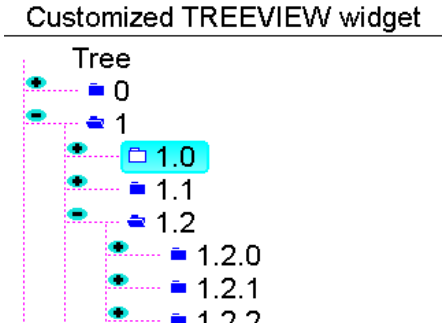




Owner drawing

Sometimes the default API functions do not allow customizing the widgets exactly to the designers vision. For that case several widgets (Graph, Listbox, Listwheel, Treeview) support owner drawing routines `<WIDGET>_SetOwnerDraw()`.

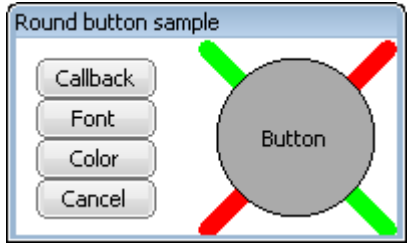
Method supported by several widgets



Overwriting the callback function

If the owner drawing function do not offer the required functionality or if there is no owner drawing routine available, the callback routine of the widget can be overwritten with `WM_SetCallback()`.

Works with each kind of window



Widget library - Skinning

Skinning is used to change the appearance of one or multiple widgets. Currently emWin supports 2 skins:

- **Classic skin**

  Old classic style. Look can be changed by using the API functions described in the 'Widget' chapter of the documentation.

- **FLEX_SKIN (default)**

  Flexible skin, can easily be modified by custom skinning routines.

The default skin for each kind of widget can be set by:

```
<WIDGET>_SetDefaultSkin()
```

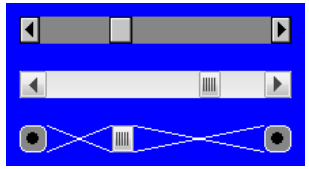
The skin of each single wigdet can be set by:

```
<WIDGET>_SetSkin()
```

Properties of FLEX_SKIN can be fetched / set by:

```
<WIDGET>_GetSkinFlexProps()
<WIDGET>_SetSkinFlexProps()
```

```
#include "DIALOG.h"

static int _ScrollbarSkinCust(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_CREATE:
    WM_SetHasTrans(pDrawItemInfo->hWin);
    break;
  case WIDGET_ITEM_DRAW_BUTTON_L:
  case WIDGET_ITEM_DRAW_BUTTON_R:
    GUI_SetColor(GUI_GRAY);
    GUI_FillRoundedRect(pDrawItemInfo->x0, pDrawItemInfo->y0,
                        pDrawItemInfo->x1, pDrawItemInfo->y1, 4);
    GUI_SetColor(GUI_WHITE);
    GUI_DrawRoundedRect(pDrawItemInfo->x0, pDrawItemInfo->y0,
                        pDrawItemInfo->x1, pDrawItemInfo->y1, 4);
    GUI_SetColor(GUI_BLACK);
    GUI_FillCircle((pDrawItemInfo->x1 + pDrawItemInfo->x0) / 2,
                   (pDrawItemInfo->y1 + pDrawItemInfo->y0) / 2, 4);
    break;
  case WIDGET_ITEM_DRAW_SHAFT_L:
  case WIDGET_ITEM_DRAW_SHAFT_R:
    GUI_SetColor(GUI_WHITE);
    GUI_DrawLine(pDrawItemInfo->x0, pDrawItemInfo->y0, pDrawItemInfo->x1, pDrawItemInfo->y1);
    GUI_DrawLine(pDrawItemInfo->x0, pDrawItemInfo->y1, pDrawItemInfo->x1, pDrawItemInfo->y0);
    break;
  default:
    return SCROLLBAR_DrawSkinFlex(pDrawItemInfo);
  }
  return 0;
}

void MainTask(void) {
  WM_HWIN hWin0, hWin1, hWin2;

  WM_SetCreateFlags(WM_CF_MEMDEV);
  GUI_Init();
  WM_SetDesktopColor(GUI_BLUE);
  SCROLLBAR_SetDefaultSkinClassic();
  hWin0 = SCROLLBAR_CreateEx(10, 10, 200, 20, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_SCROLLBAR0);
  SCROLLBAR_SetDefaultSkin(SCROLLBAR_SKIN_FLEX);
  hWin1 = SCROLLBAR_CreateEx(10, 50, 200, 20, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_SCROLLBAR1);
  hWin2 = SCROLLBAR_CreateEx(10, 90, 200, 20, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_SCROLLBAR2);
  SCROLLBAR_SetSkin(hWin2, _ScrollbarSkinCust);
  SCROLLBAR_SetValue(hWin0, 30);
  SCROLLBAR_SetValue(hWin1, 70);
  SCROLLBAR_SetValue(hWin2, 30);
  while (1) {
    GUI_Delay(100);
  }
}
```



Widget library summary

You should now know the following:

- **The difference between direct- and indirect widget creation**
- **How to manage extended widget initialization within a dialog**
- **The difference between 'owner drawing', 'overwriting callback' and 'skinning'**
- **Why it is possible to use most of the window manager functions with widgets**

GUI-Builder

Tool for creating dialogs without knowledge of the C language.

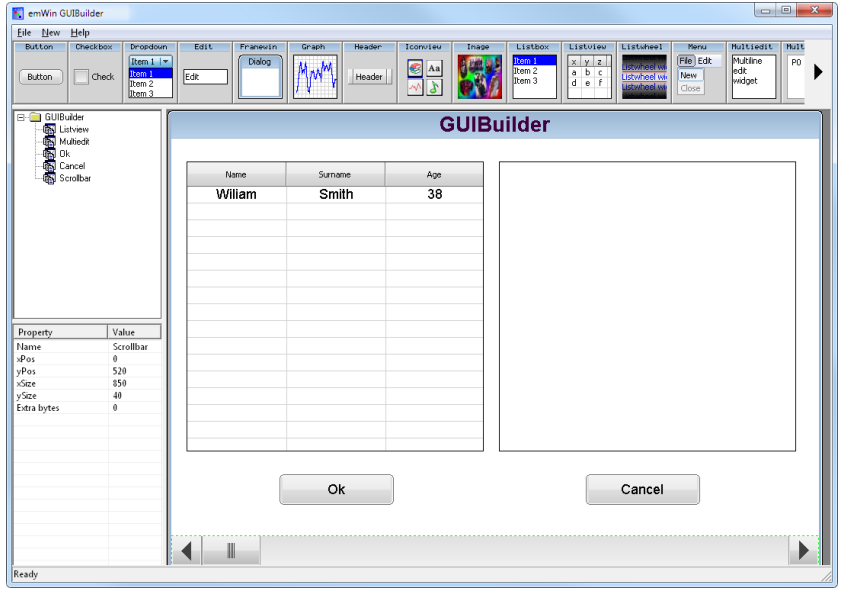
Basic usage:

- **Check project path in** `GUIBuilder.ini`
  Located in the application folder.
- **Start GUI-Builder**
- **Start with** `FRAMEWIN` **or** `WINDOW` **widget**
  Only these widgets could serve as dialog parent window.
- **Place widgets within the parent window**
  Placed and size widgets by moving them with the mouse and/or by editing the properties in the property window.
- **Configure the widgets**
  The context menu shows the available options.
- **Save dialog**
  Each dialog is saved in a separate file. The filenames are generated automatically by the name of the parent window.

The filenames are automatically generated by the name of the parent window:

`<WindowName>Dlg.c`

Property	Value
Name	GUIBuilder



- Files can be opened by **drag and drop**
- **Multiple dialogs** allowed simultaneously
- Each dialog is saved in a separate file
- **Filenames** are generated **automatically**

GUI-Builder - Filenames and code sections

Filenames

The project path contains one C file for each parent widget. The filenames are automatically generated by the names of the parent windows:

```
<WindowName>Dlg.c
```

Creation routine

The file contains a creation routine for the dialog. The routine name includes the name of the parent window:

```
WM_HWIN Create<WindowName>(void);
```

Simply call this routine to create the dialog:

```
hWin = CreateFramewin();
```

User defined code

The generated code contains a couple of comments to add user code between them. To be able to read back the file with the GUI-Builder the code must be between these comments.

Note: Adding code outside the user code comments makes the file unreadable for the GUI-Builder.

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

// USER START (Optionally insert additional includes)
// USER END

#include "DIALOG.h"

/*********************************************************************
*       Defines
**********************************************************************
*/

. . . ID definitions

// USER START (Optionally insert additional defines)
// USER END

/*********************************************************************
*       Static data
**********************************************************************
*/

// USER START (Optionally insert additional static data)
// USER END

/*********************************************************************
*       _aDialogCreate
*/
. . . Resource table

/*********************************************************************
*       Static code
**********************************************************************
*/

// USER START (Optionally insert additional static code)
// USER END

/*********************************************************************
*       _cbDialog
*/
. . . Callback routine

/*********************************************************************
*       Public code
**********************************************************************
*/
/*********************************************************************
*       CreateXXX
*/
. . . Creation routine

/************************** End of file ****************************/
```

SAMPLES: GUIBuilder\CatsMovie.gif
GUIBuilder\SmileyMovie.gif

GUI-Builder - Callback routine

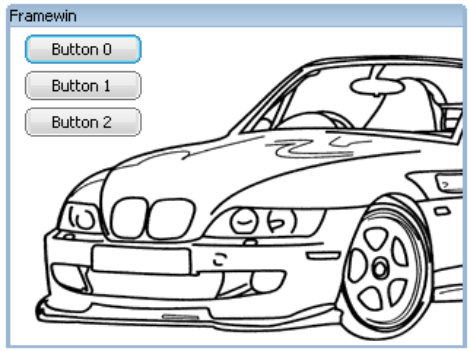
Main part of the generated file is the callback routine. It normally contains the following message handlers:

- **WM_INIT_DIALOG**

  The widget initialization is done here immediately after creating all widgets of the dialog. The user code area can be used to add further initialization.

- **WM_NOTIFY_PARENT**

  Contains (empty) message handlers to be filled with user code. For each notification of the widget there is one message handler.
  Further reactions on notification messages can be added.




```
/*********************************************************************
*
*       _cbDialog
*/
static void _cbDialog(WM_MESSAGE * pMsg) {
  WM_HWIN hItem;
  int Id, NCode;
  // USER START (Optionally insert additional variables)
  // USER END

  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    //
    // Initialization of 'Button'
    //
    hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_0);
    BUTTON_SetFont(hItem, GUI_FONT_20_ASCII);
    // USER START (Optionally insert additional code for further widget initialization)
    // USER END
    break;
  case WM_NOTIFY_PARENT:
    Id    = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;
    switch(Id) {
    case ID_BUTTON_0: // Notifications sent by 'Button'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      case WM_NOTIFICATION_RELEASED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      // USER START (Optionally insert additional code for further notification handling)
      // USER END
      }
      break;
    // USER START (Optionally insert additional code for further Ids)
    // USER END
    }
    break;
  // USER START (Optionally insert additional message handling)
  // USER END
  default:
    WM_DefaultProc(pMsg);
    break;
  }
}
```

GUI-Builder summary

You should now know the following:

- **Which kinds of widgets could be used for starting a dialog**
- **Which file should be used to set up the project path**
- **What needs to be observed when adding user code**
- **How the file names are generated when saving the project**