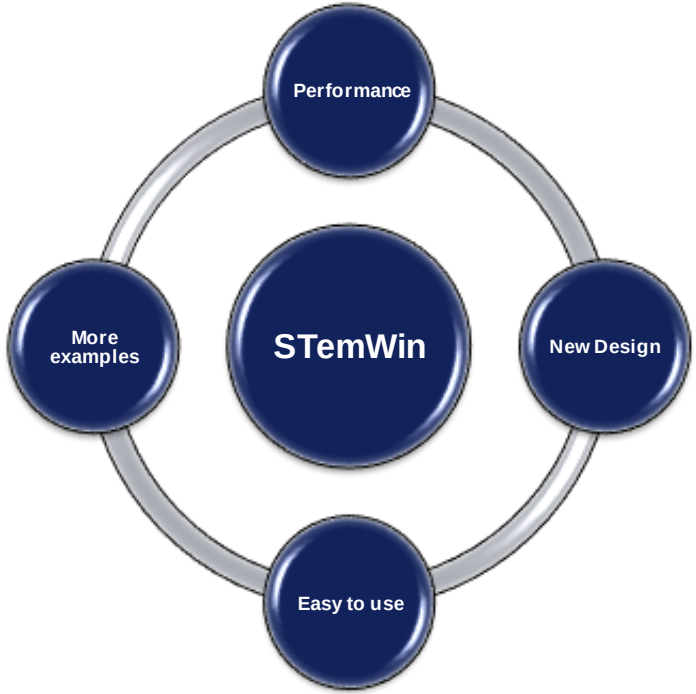
STemWin Training

Overview

- STemWin is a professional graphical stack library, enabling Graphical User Interfaces (GUI), building up with any STM32, any LCD and any LCD controller, taking benefit from STM32 hardware accelerations, whenever possible.

- Our focus in the coming STemWin packages will be the enhancement of the performance, the look and feel of the new applications and demonstrations and the ease the use of the stack with the integration with CubeMX and the enhancement of the documentation (AN, User manual ..)



Content

1. Introduction
2. Tools
3. Configuration
4. Core functions
5. Memory devices
6. Antialiasing
7. Window manager
8. Widget library
9. GUI-Builder

The shipment

Lib:

- All STemWin binary files generated with different compilers, optimizations, core and pixel component order.

Documentation:

- API-documentation
- This training (Will be included in coming version)

Config:

- LCD configuration files, LCD driver template and STemWin configuration files

Software:

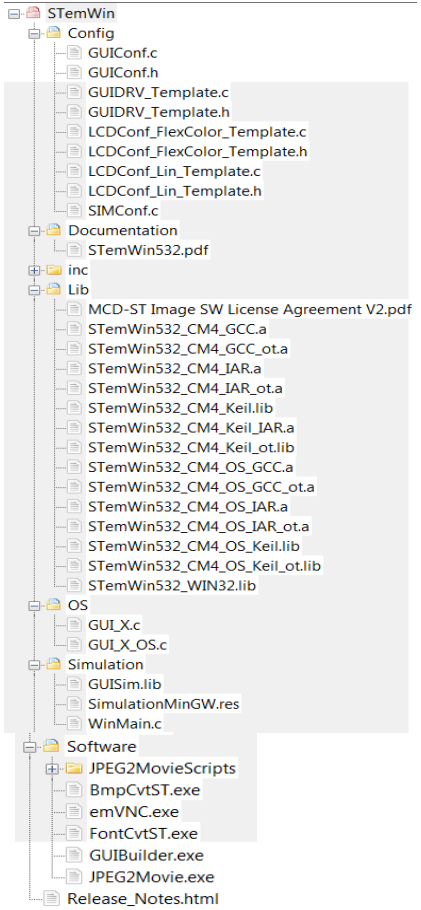
- All tools which are part of the license

Inc:

All STemWin header files

OS:

STemWin interface files with OS



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**

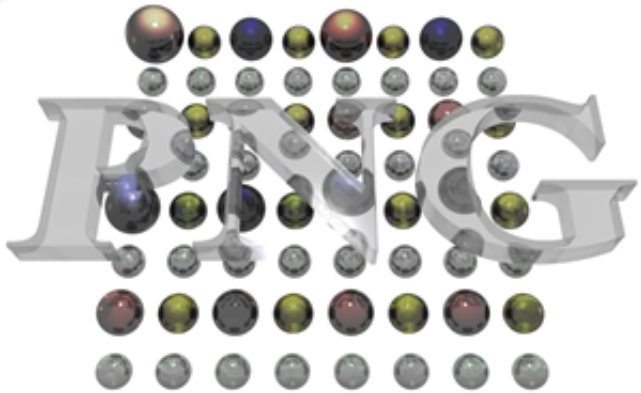
www.segger.com/downloads/emwin/emWin_PNG

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.



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

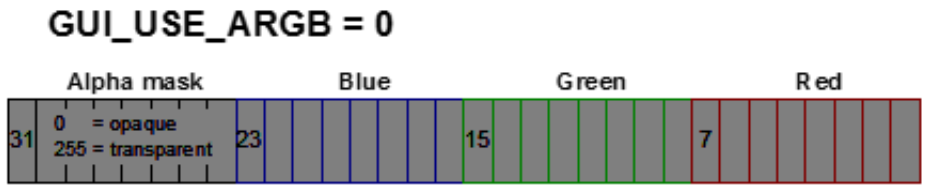
Alpha channel

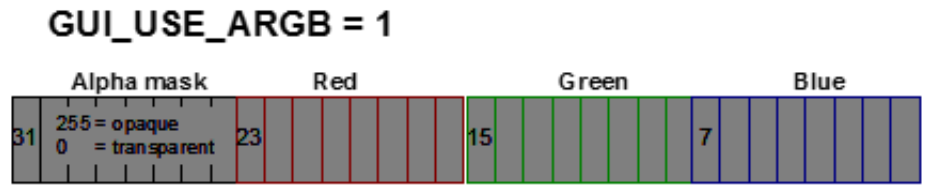
- **GUI_USE_ARGB = 0**:
0 means opaque, 255 means 100% transparent
- **GUI_USE_ARGB = 1**:
0 means 100% transparent, 255 means opaque

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.






```
#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);
  }
}
```

```
#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);
  }
}
```

Tools overview

- **Bitmap converter**
  Converts images and saves them as C-files or streamed bitmaps.

- **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**
  Can be used for converting any font installed on the host system into emWin compatible formats.

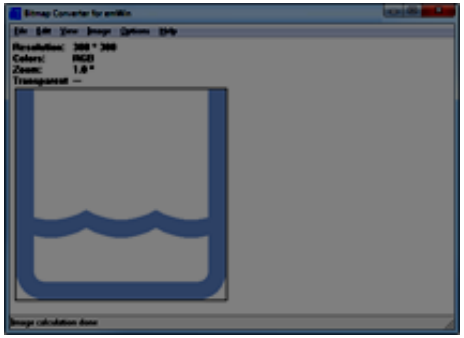
- **GUIBuilder**
  Used to generate dialog based applications without writing any line of code.

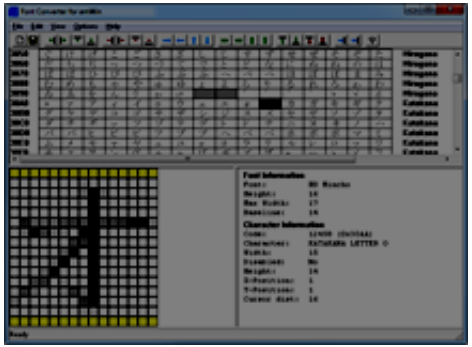
- **emVNC**
  VNC client to be used for VNC connections with or without emWin.

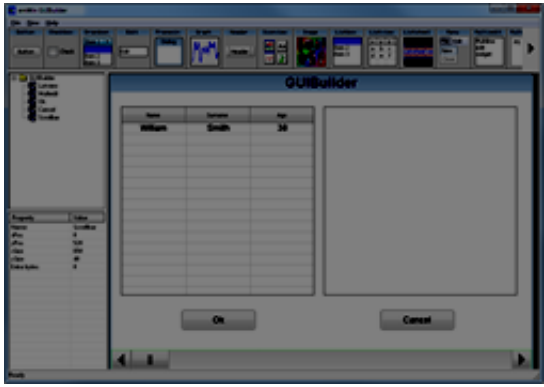
➔ emWinView and emWinSPY tools will be included in STemWin 5.44 version

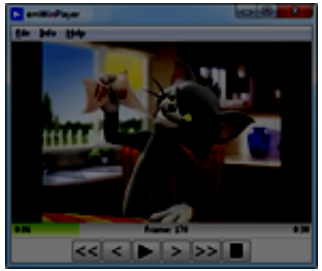
- **emWinSPY**
  Shows runtime information of the embedded target on a PC.

- **emWinView**
  To be used while stepping through the debugger of the simulation.









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**

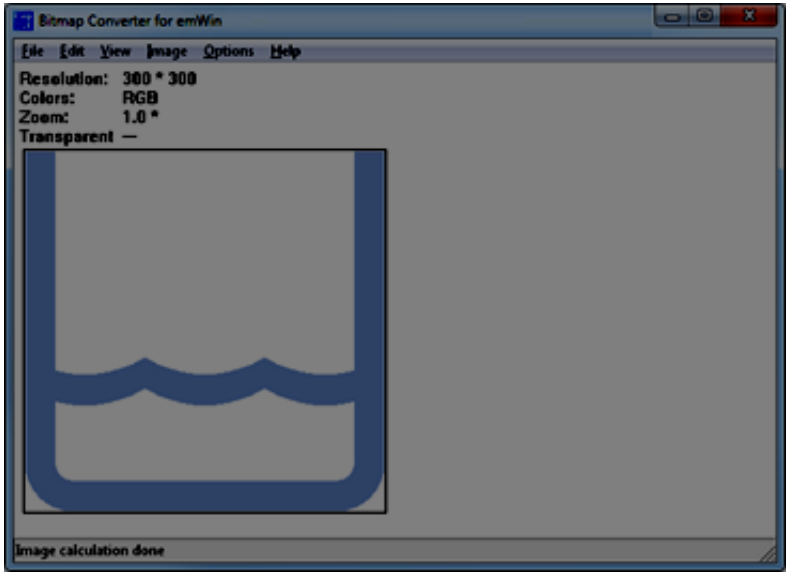
Most recommended binary format for systems short on addressable ROM.
Accessed by `GetData()` function passed by the customer.

- **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\...**' .

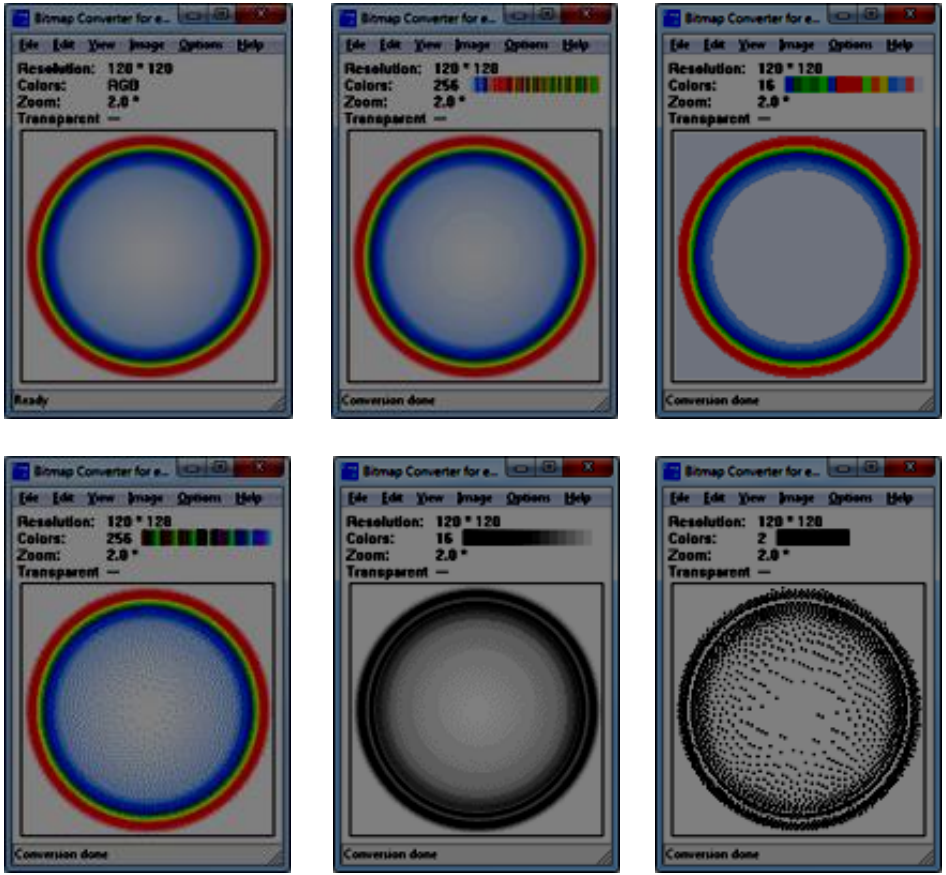
- **Color reduction**

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 ---**



FontConverter

- Converts **any installed font** of the host system.
- Font **types**: STD, EXT, AA2, ...
- Font **formats**: C, XBF (**Ex**ternal **B**inary **F**ont), SIF (**S**ystem **I**ndependent **F**ont), BDF (**B**itmap **D**istribution **F**ormat).
- 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

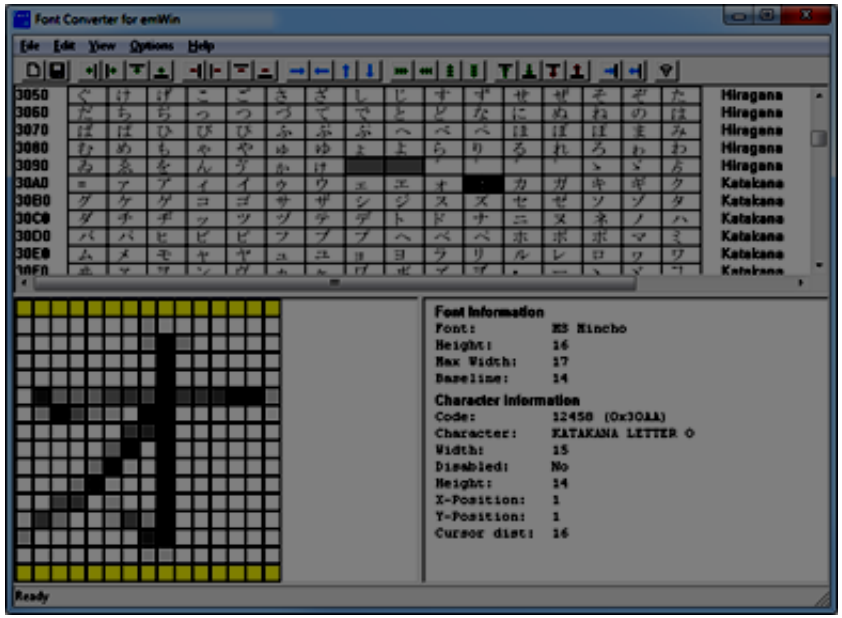

Shift content


Move position


Insert/delete pixel line to font


Increase/decrease cursor increment


Filter for showing only blocks with content



FontConverter - Chinese language sample

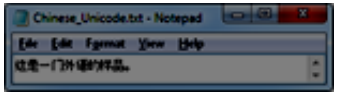
Unicode support should be enabled once at the beginning of the application:

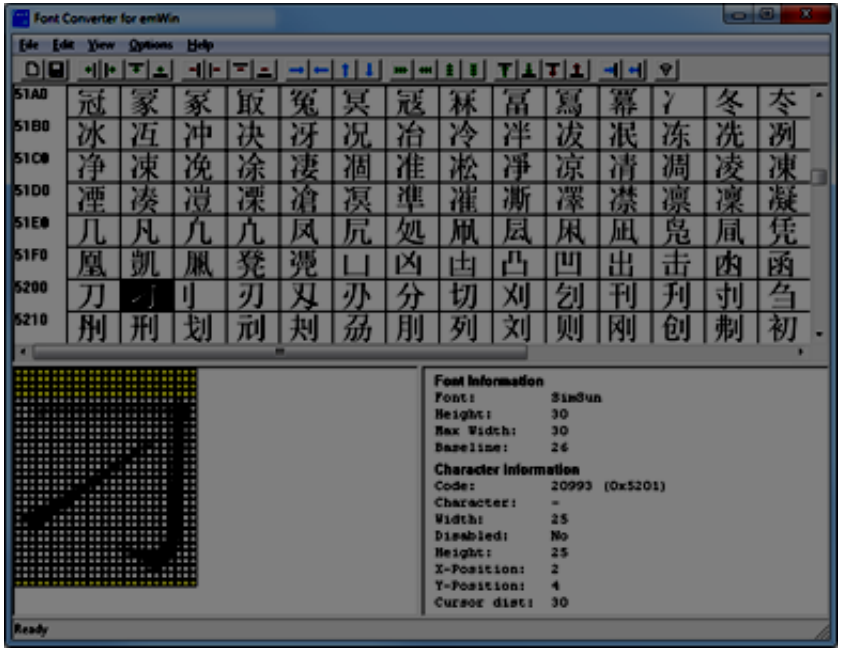
```
GUI_Init();
GUI_UC_SetEncodeUTF8(); // Enable UTF-8 support
```

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);
  }
}
```

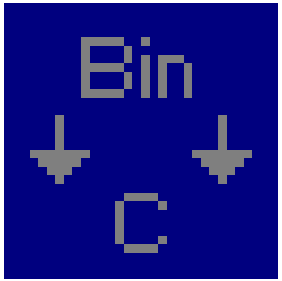
SAMPLES: FontConverter\Chinese*.*
FontConverter\FontChinese_30_EXT.c

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 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);
  }
}
```

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**

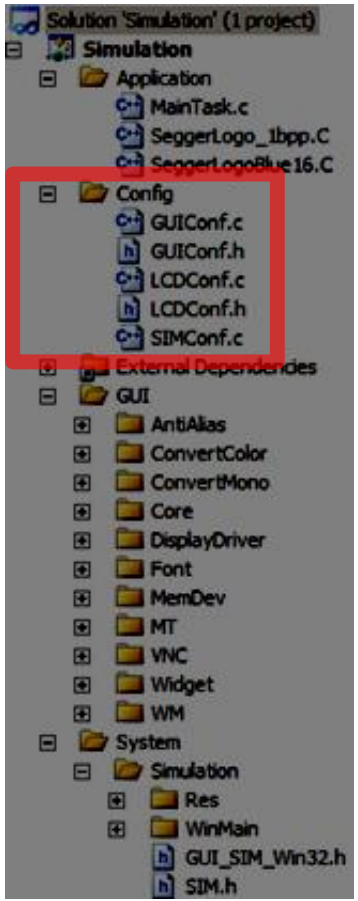
Display driver configuration and initialization

- **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.

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 - 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)**

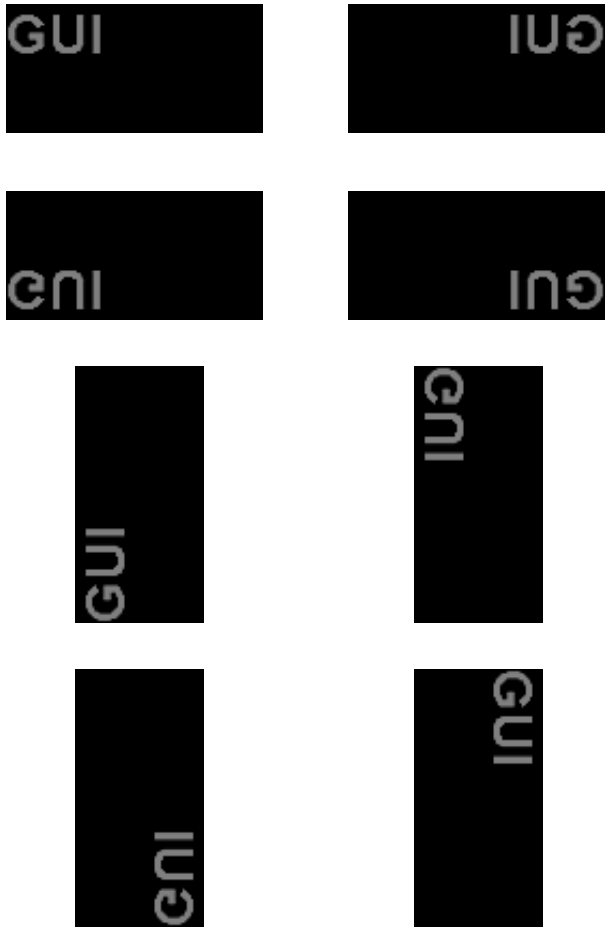
Other drivers like the **`GUIDRV_FlexColor`** contain configuration functions for setting the orientation.

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

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

- **Touch orientation**

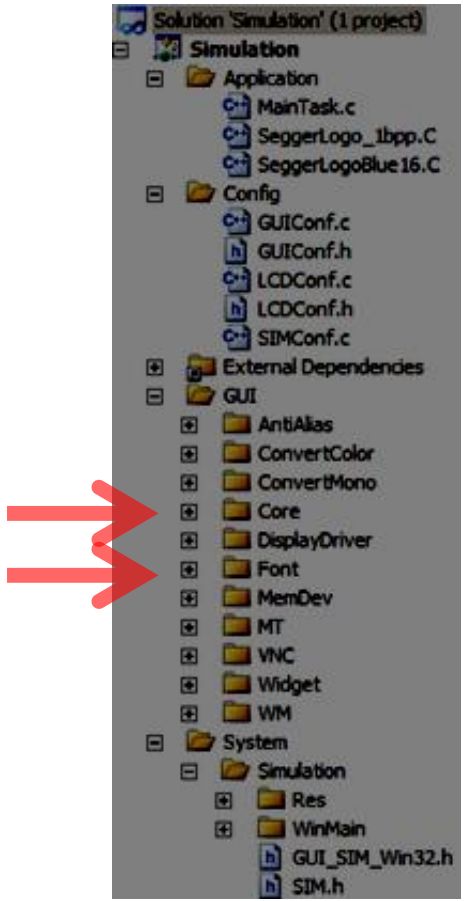
Can be determined by calling the function **`GUI_TOUCH_SetOrientation()`**.



Core functions - emWin BASE

Image file support for BMP, GIF and JPEG

- Drawing of images and fonts from non addressable media
- bidirectional text support
- Sprites and cursors, animated
- Support for non antialiased font formats
- Drawing of text and values (dec, bin, hex, float)
- Multiple buffering
- Animations
- Touch screen support
- Language support



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

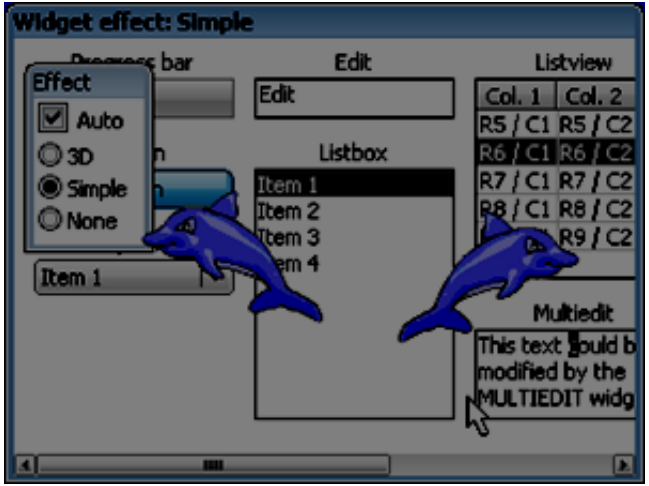
- **Drawing from non addressable areas**

_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;
}
```

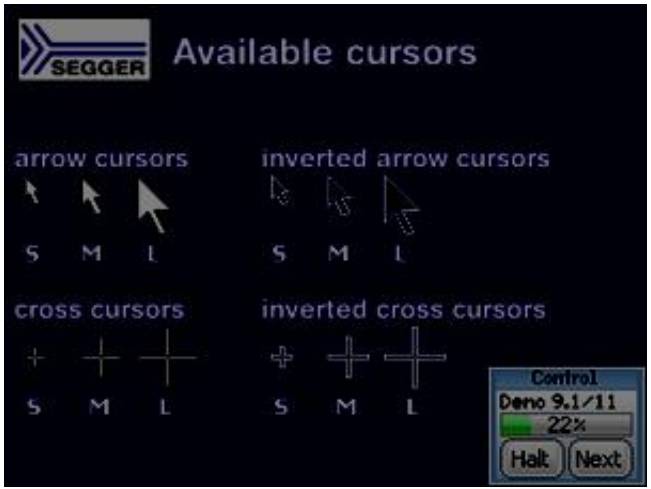
Software sprites

- 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**



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

Memory devices

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

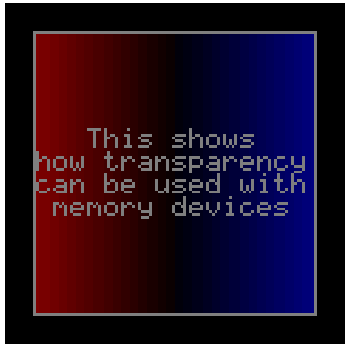
What means 'transparency' here?

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 - 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);
    }
  }
}
```

Antialiasing

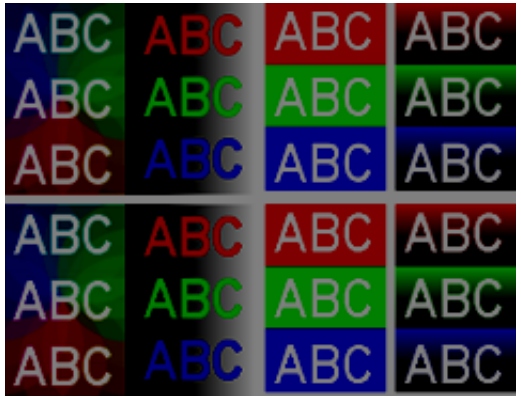
Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground.

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);
  }
}
```



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 - 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 - Preventing flickering effects

- **Multiple buffering**

WM manages buffer handling automatically

- **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

Memory devices

Last choice

Window manager - Window animations

Window animation functions are available for the following effects:

- **Moving in and out**

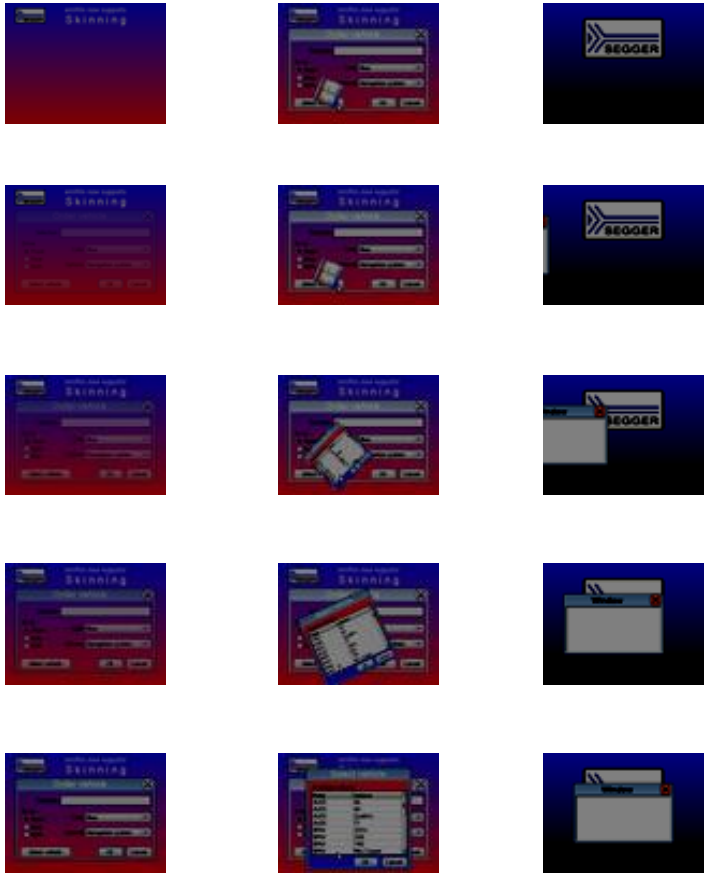
Any widget/window can be moved in/out from a determined position. The window will be enlarged/shrinked and moved/turned to its final position.

- **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 - 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.

Widget library

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

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.

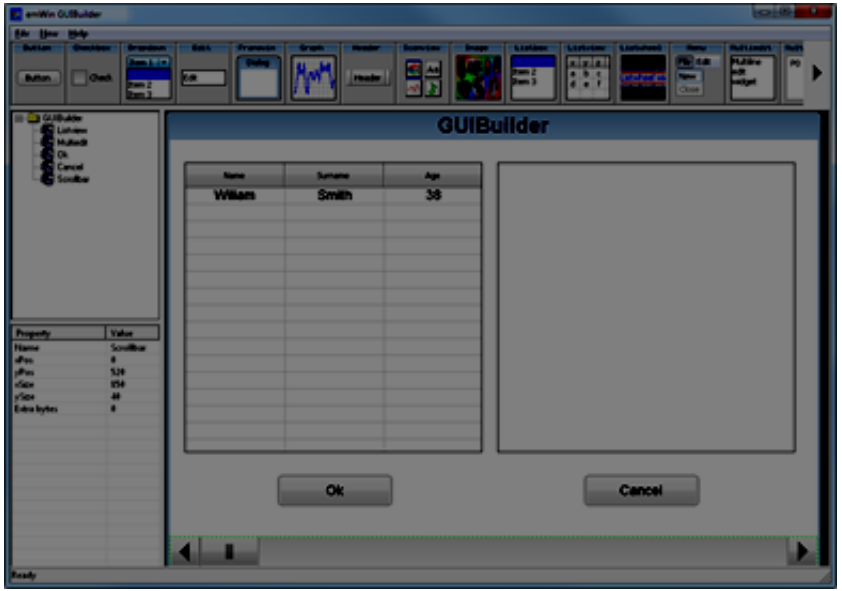
- **Configure the widgets**
  The context menu shows the available options.

- **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:

Property	Value
Name	GUIBuilder

`<WindowName>Dlg.c`



- 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

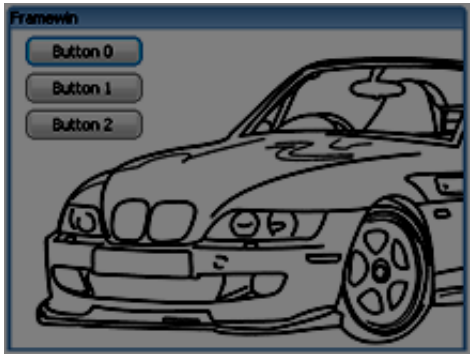
Main part of the generated file is the callback routine. It normally contains the following message handlers:

- **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;
  }
}
```

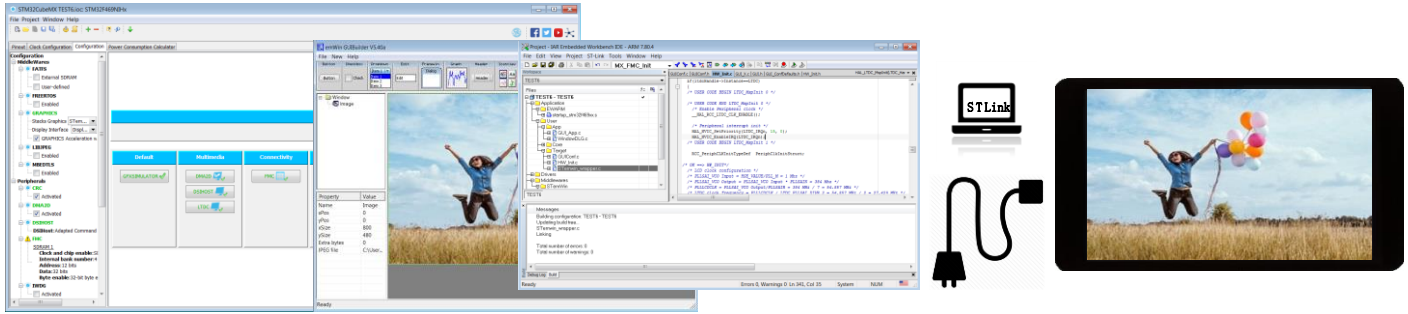
STemWin Strategy

Integration In CubeMX

The creation of a graphical interface and configured application to be loaded directly to the target (F4/F7) will become possible using CubeMX init tool.

The user can choose the MCU, configure the needed IPs (LTDC, DSI, SDRAM, CRC ..) and the specific parameters of the graphical stack using CubeMX.

From CubeMX, the user can launch the STemWin designer tool (GUIBuilder) and create his own interface, and then create the project and load the program on the target.



New Applications

The target of the new applications that will be delivered with coming STM32Cube firmware packages is to highlight the main features of STemWIn and answers the main requests observed from regions and customers as the display of different languages, the use of hardware accelerator to enhance the performance, the animation, rotation and zoom.

STemWin new applications:

- ✓ STemWin_HelloWorld
- ✓ STemWin_Animation:
- ✓ STemWin_MemoryDevices
- ✓ STemWin_fonts
- ✓ STemWin_Acceleration

New look

The focus in coming STemWin demonstrations and applications in addition of the performance will be the look and feel of the graphical interfaces.

The target is to use high quality bitmaps and ensure the fluidity of the animations and transitions

STM32L469-DISCO Demonstration

STM32H743I-EVAL Demonstration

Questions