AN4323
Application note

Getting started with STemWin Library

Introduction

A partnership with Segger Microcontroller GmbH & Co. KG enables STMicroelectronics to provide STemWin Library, a product based on Segger's graphic library emWin.

STemWin Library is a professional graphical stack library enabling the building up of Graphical User Interfaces (GUIs) with any STM32, any LCD/TFT display and any LCD/TFT controller, taking advantage of STM32 hardware accelerations whenever possible.

STemWin Library is a comprehensive solution coming with rich features such as JPG, GIF and PNG decoding, many widgets (checkboxes, buttons…) and a VNC server allowing to display remotely a local display, but also professional tools such as GUIBuilder to create dialog boxes by drag and drop, a font converter, etc.

The letters XYZ in the terms STemWinXYZ and STemWinLibraryXYZ refer to the latest STemWin version as described in the firmware package release notes.

This graphic library is fully integrated inside the STM32Cube firmware package (such as STM32CubeF2, STM32CubeF3 and STM32CubeF4). It can be downloaded free from STMicroelectronics web site (http://www.st.com/stm32cube)

Table 1. Applicable Software

Type	Part numbers and product categories
MCU Software	STemWin, STM32CubeF3, STM32CubeF2, STM32CubeF4

Contents

List of tables

List of figures

1 Library and package presentation

The STemWin Library package includes a set of firmware libraries and software tools used to build advanced and professional GUI-based applications.

1.1 Licensing information

- STemWin Library GUI files are provided in object format and licensed under MCD-ST Image Software License Agreement V2 (the “License”); you may not use this package except in compliance with the License. You may obtain a copy of the License at: http://www.st.com.
- STemWin Library configuration and header files are provided in source format and licensed under MCD-ST Liberty Software License Agreement V2 (the “License”); you may not use this package except in compliance with the License. You may obtain a copy of the License at: http://www.st.com

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1.2 Library description

Figure 1 shows how STemWin is structured internally and how it can be implemented in a complete project.

Figure 1. STemWin layers

1. The CRC module (in RCC peripheral clock enable register) should be enabled before using the library.

STemWin Library includes two optimized drivers:

- Direct linear access (LIN) driver for the STM32F429 TFT-LCD controller with the DMA2D ChromART graphical acceleration engine,
- FlexColor (indirect access) driver for serial and parallel bus external LCD controllers available on all STM32 EVAL boards.

Refer to *Table 2* for a list of all supported display controllers.

Note: It is still possible to support any other LCD type just by implementing its own "custom" driver.

In addition to the main application, the user has to set and customize two essential interface files:

- LCD configuration file (LCDConf_stm32xxx_eval.c)
  - LCD Display initialization and configuration
  - LCD Display driver link and customizing
  - Additional hardware capability management
- GUI configuration file (GUIConf_stm32xxx_eval.c)
  - Module selection (memory device, window manager…)
  - GUI memory and heap management

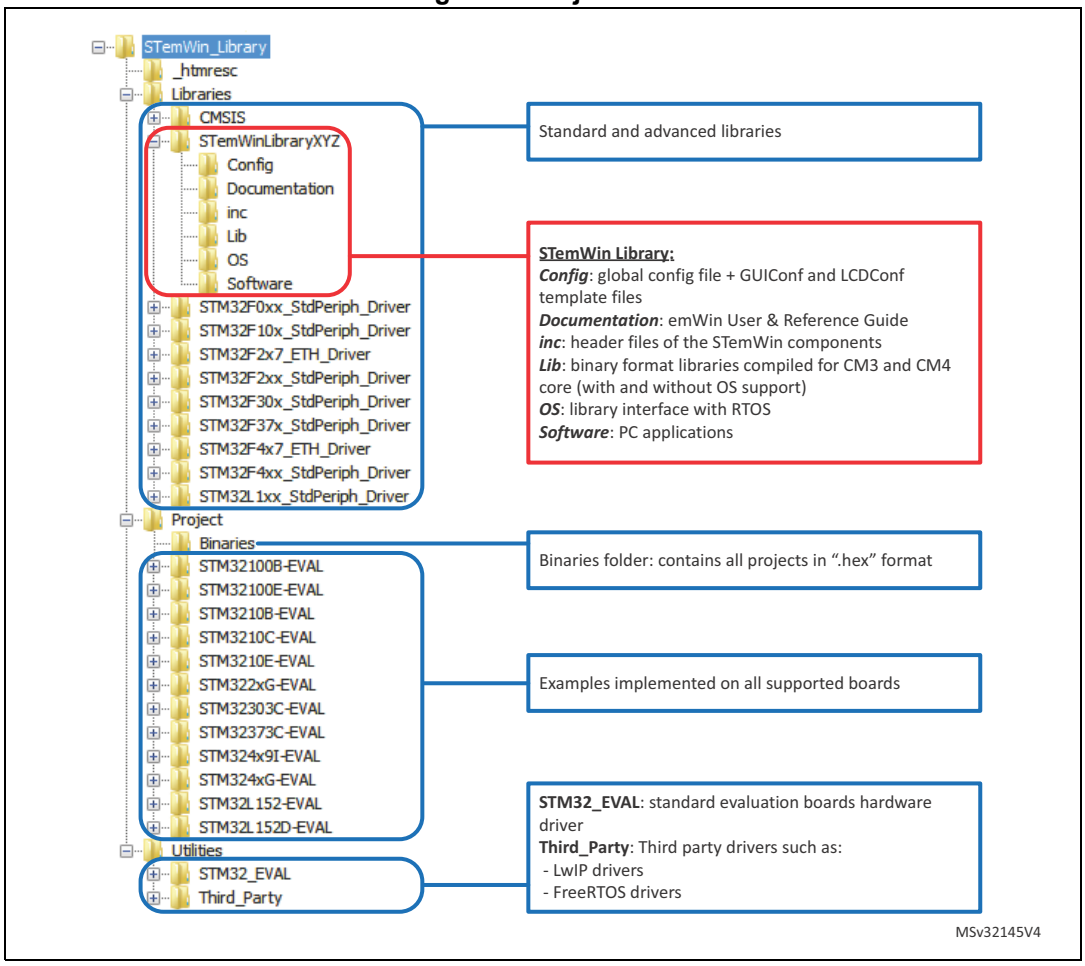
Table 2. Supported LCD controllers

Driver	Supported LCD controllers	Supported bits/pixels
GUIDRV_Lin	This driver supports every display controller with linear addressable video memory with a direct (full bus) interface. This means that the video RAM is directly addressable by the address lines of the CPU. The driver contains no controller-specific code. So it can also be used for solutions without display controller which require a driver which only manages the video RAM.	16, 18
GUIDRV_FlexColor	Epson S1D19122 FocalTech FT1509 Himax HX8301, HX8340, HX8347, HX8352, HX8353, HX8325A Hitachi HD66772 Ilitek ILI9220, ILI9221, ILI9320, ILI9325, ILI9328, ILI9335, ILI9338, ILI9340, ILI9341, ILI9342, ILI9481 LG Electronics LGDP4531, LGDP4551 Novatek NT39122 OriseTech SPFD5408, SPFD54124C, SPFD5414D Renesas R61505, R61516, R61526, R61580 Samsung S6D0117, S6E63D6 Sitronix ST7628, ST7637, ST7687, ST7712, ST7735 Solomon SSD1355, SSD1961, SSD1963, SSD2119 Syncoam SEPS525	1, 2, 4, 8, 16, 24, 32

1.3 Package organization

Figure 2 shows the project tree.

Figure 2. Project tree



1.4 Delivered binaries

STemWin Library is distributed by ST as an object code library locked to STM32 products.

The library is compiled for CM3 and CM4 cores, both with and without OS support. In the CM4 versions, the FPU is enabled.

Note that the library is compiler-dependent (IAR, ARM, GCC).

To summarize, the folder “Libraries\STemWinLibraryXYZ\Lib” contains twelve binaries:

- STemWinXYZ_CM3_IAR.a
- STemWinXYZ_CM3_Keil.a
- STemWinXYZ_CM3_GCC.a
- STemWinXYZ_CM3_OS_IAR.a
- STemWinXYZ_CM3_OS_Keil.a
- STemWinXYZ_CM3_OS_GCC.a
- STemWinXYZ_CM4_IAR.a
- STemWinXYZ_CM4_Keil.a
- STemWinXYZ_CM4_GCC.a
- STemWinXYZ_CM4_OS_IAR.a
- STemWinXYZ_CM4_OS_Keil.a
- STemWinXYZ_CM4_OS_GCC.a

2 Supported EVAL boards and examples

Table 3 lists the supported EVAL boards and examples.

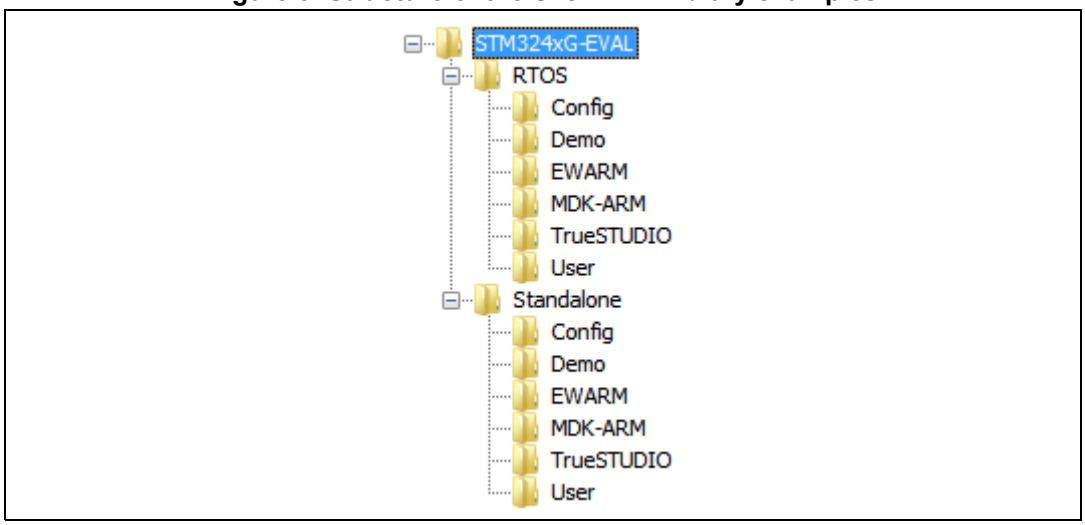
Table 3. Supported EVAL boards and examples

STM32 Series/EVAL board		LCD interface	LCD driver
STM32F1	STM3210B-EVAL	SPI	FlexColor
	STM3210C-EVAL	SPI	
	STM3210E-EVAL	FSMC	
	STM32100B-EVAL	SPI	
	STM32100E-EVAL	FSMC	
STM32F2	STM322xG-EVAL	FSMC	
STM32F3	STM32303C-EVAL	SPI	
	STM32373C-EVAL	SPI	
STM32F4	STM324xG-EVAL	FSMC	
	STM324x9I-EVAL	LTDC	Lin
STM32L1	STM32L152-EVAL	SPI	FlexColor
	STM32L152D-EVAL	FSMC	

As illustrated in *Figure 3*, each project contains two sub-projects:

- RTOS-based example
- Standalone example

Figure 3. Structure of the STemWin Library examples



The Config\ folder contains the two interface files described in *Section 1.2*:

- GUIConf_stm32xxx_eval.c/.h
- LCDConf_stm32xxx_eval.c/.h

In the RTOS example, the Config\ directory also contains the Free RTOS configuration file (FreeRTOSConfig.h).

The main application source code is located in the Demo\ folder. It consists of a generic demonstration composed of several sub-modules such as:

- Radial menu (selects an icon from a radial menu using STemWin motion support)
- List view (shows some features of the LISTVEW widget)
- Bitmap (shows different bitmaps with and without compression)
- Antialiased text (outputs anti-aliased text on various backgrounds)
- Tree view (shows a hierarchical view of the files in a directory and some moving sprites)
- Icon view (demonstrates the use of the ICONVIEW widget)

…

All these modules cannot be fully supported on all EVAL boards: each example will contain more or less modules depending on the memory available on the board.

The user can choose which module to activate in file GUIDEMO.h located under Demo\.

A portion of code (module selection) from the GUIDEMO.h file is shown below.

```
/*********************************************************************
*
*       Configuration of modules to be used
*
**********************************************************************
*/

#ifndef   SHOW_GUIDEMO_BITMAP
  #define SHOW_GUIDEMO_BITMAP                (1)
#endif

#ifndef   SHOW_GUIDEMO_COLORBAR
  #define SHOW_GUIDEMO_COLORBAR          (0)
#endif

#ifndef   SHOW_GUIDEMO_CURSOR
  #define SHOW_GUIDEMO_CURSOR               (1)
#endif

#ifndef   SHOW_GUIDEMO_GRAPH
  #define SHOW_GUIDEMO_GRAPH                  (1)
#endif

#ifndef   SHOW_GUIDEMO_LISTVIEW
  #define SHOW_GUIDEMO_LISTVIEW              (1)
#endif
```

```
#ifndef   SHOW_GUIDEMO_SPEED
  #define SHOW_GUIDEMO_SPEED                  (1)
#endif

#ifndef   SHOW_GUIDEMO_TREEVIEW
  #define SHOW_GUIDEMO_TREEVIEW            (0)
#endif

#ifndef   SHOW_GUIDEMO_ICONVIEW
  #define SHOW_GUIDEMO_ICONVIEW            (1)
#endif

#ifndef   SHOW_GUIDEMO_RADIALMENU
  #define SHOW_GUIDEMO_RADIALMENU     (1)
#endif

#ifndef   SHOW_GUIDEMO_VSCREEN
  #define SHOW_GUIDEMO_VSCREEN            (0)
#endif

#ifndef   SHOW_GUIDEMO_AUTOMOTIVE
  #define SHOW_GUIDEMO_AUTOMOTIVE    (0)
#endif
```

Note that the VNC module (described in detail in *Section 3.8*) is activated by default with the following configurations:

- STM324xG-EVAL + FreeRTOS
- STM322xG-EVAL + FreeRTOS
- STM324x9I-EVAL-MB1046 + FreeRTOS

The User\ directory includes the system-related files, the main program and, in some examples (where the VNC feature is supported), the network connection configuration files.

3 How to use STemWin Library step by step

This section provides an overview of the main features of STemWin Library, as well as the main settings and configuration steps. For further details, refer to the Segger's emWin User Manual.

3.1 Configuration

The configuration is basically divided into two parts: GUI configuration and LCD configuration.

- The GUI configuration covers the configuration of default colors and fonts and of available memory.
- The LCD configuration is more hardware-dependent and enables the user to define the physical size of the display, the display driver and the color conversion routines to be used.

When a new LCD controller needs to be supported, two essential files must be created, in addition to the already existing OS configuration file: GUIConf_stm32xxx_eval.c and LCDConf_stm32xxx_eval.c.

3.1.1 GUIConf_stm32xxx_eval.c

In this file, the user should implement the GUI_X_Config() function which is the very first routine called during the initialization process. Its main task is to set up the available memory for the GUI and to then assign it to the dynamic memory management system.

This operation is done via the GUI_ALLOC_AssignMemory() function: it passes a pointer to a memory block and its size (in bytes) to the memory manager.

Note: The memory must be accessible and must be 8-, 16- and 32-bit wide. Memory access is checked during initialization.

3.1.2 LCDConf_stm32xxx_eval.c

The main function here is LCD_X_Config(), called immediately after GUI_X_Config() has been executed. LCD_X_Config() allows creating and configuring a display driver for each layer by calling:

- GUI_DEVICE_CreateAndLink(), which creates the driver device and links it to the device chain;
- LCD_SetSizeEx() and LCD_SetVSizeEx(), to set the display size configuration;
- Other driver-specific configuration routines such as:
  - GUIDRV_FlexColor_Config(), in association with the usage of the FlexColor driver;
  - LCD_SetVRAMAddrEx(), required in case of linear addressable memory.

As mentioned in *Section 1.2*, STemWin Library comes with two optimized drivers, GUIDRV_Lin and GUIDRV_FlexColor, which cover all the LCDs embedded in ST EVAL-boards.

For more details about usage of these drivers, please visit Segger's website:

http://www.segger.com

In general, when a new LCD type needs to be supported, the user should check if the same LCD controller is already supported, then he just has to update the already existing LCDConf_stm32xxx_eval.c.

Another function, LCD_X_DisplayDriver(), is called by the display driver for several purposes, for example when using advanced features like multiple buffers, smooth scrolling or virtual pages. To support the corresponding task, the routine needs to be adapted to the display controller.

If the used display controller is not supported, user can easily create his own driver just by adapting the GUIDRV_Template.c file located under Library\STemWinLibraryXYZ\Config. Actually this template file contains the complete functionality needed for a display driver. The main routines that need to be adapted are _SetPixelIndex() and _GetPixelIndex(). If the display is not readable, a display data cache should be implemented instead of using the _GetPixelIndex() function.

3.1.3 GUI_X.c or GUI_X_OS.c

- GUI_X.c for single task execution:

"Single task" means that the project uses STemWin only from within one single task. The main purpose is to supply STemWin with a timing base. OS_TimeMS needs to be incremented each ms.

- GUI_X_OS.c for multitask execution:

If STemWin is used in a multitasking system, this file contains additional routines required for synchronizing tasks (for this purpose, the file GUI_X_FreeRTOS.c can be used as a template).

3.2 GUI initialization

To initialize the STemWin internal data structures and variables, GUI_Init() should be used.

Note that before initializing the GUI, the CRC module (in RCC peripheral clock enable register) should be enabled

A simple "Hello world" program illustrates this initialization, as shown below.

"Hello world" example:

```
void Main(void) {
  int xPos, yPos;

  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_CRC, ENABLE);
  GUI_Init();

  xPos = LCD_GetXSize() / 2;
  yPos = LCD_GetYSize() / 3;
  GUI_SetFont(GUI_FONT_COMIC24B_ASCII);
  GUI_DispStringHCenterAt("Hello world!", xPos, yPos);
```

```
  while(1);
}
```

3.3 Core functions

3.3.1 Image file display

STemWin currently supports the BMP, JPEG, GIF and PNG file formats. The library includes rich APIs for each one of these image formats (fully documented in the STemWin User Manual). An approximation of the memory resources needed for each image type is given in *Section 4: Performance and footprint*.

3.3.2 Bidirectional text

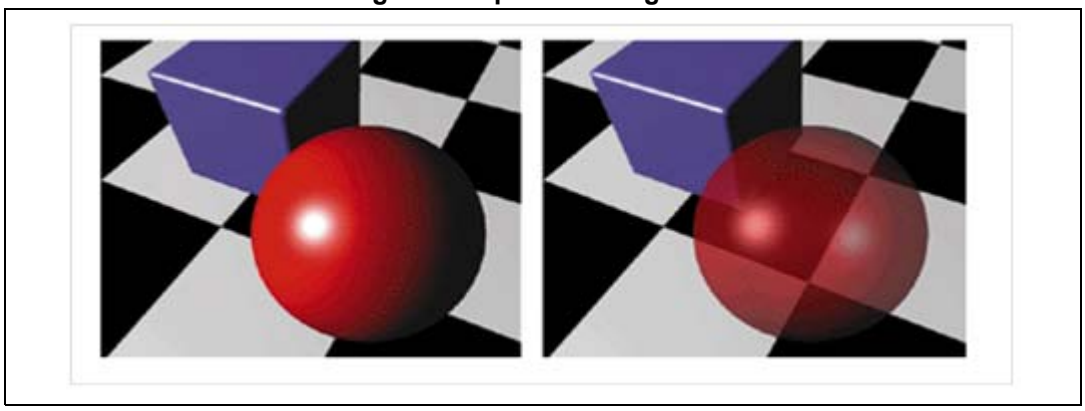
Drawing Arabic or Hebrew text with STemWin is quite easy and is supported automatically in each text-based function. It only needs to be enabled once by using the following command:

```
GUI_UC_EnableBIDI()
```

3.3.3 Alpha blending

Alpha blending is a method combining the alpha channel with other layers in an image in order to create the appearance of semi-transparency (see *Figure 4*).

Figure 4. Alpha blending effect



The user can enable automatic alpha blending using the following command:

```
GUI_EnableAlpha()
```

He can also give an alpha value to determine how much of a pixel should be visible and how much of the background should show through:

```
GUI_SetUserAlpha()
```

3.3.4 Sprites and cursors

A sprite is an image which can be shown above all other graphics on the screen.

A sprite preserves the screen area it covers. It can be moved or removed at any time, fully restoring the screen content. Animation by use of multiple images is also possible.

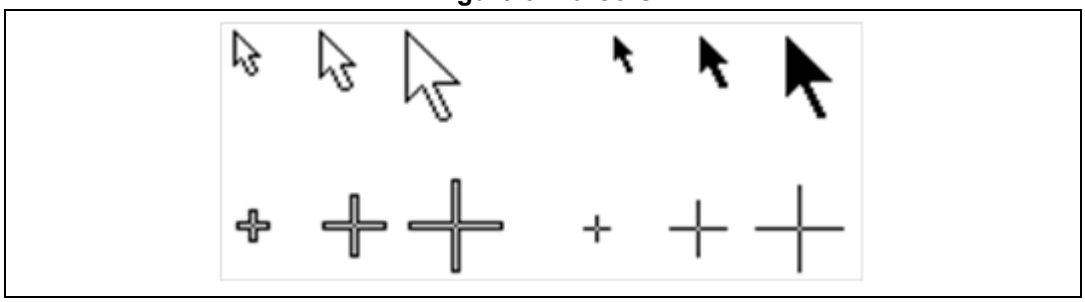
Figure 5. Animated sprites

Sprites can be animated (*Figure 5*) by calling `GUI_SPRITE_CreateAnim()`.

Note that sprites manage the background automatically.

STemWin also includes a system-wide cursor (*Figure 6*), which can also be animated by using `GUI_CURSOR_SetAnim()`. Cursors are actually based on sprites.

Figure 6. Cursors



Although the cursor always exists, it is hidden by default. It is not visible until a call is made to show it (`GUI_CURSOR_Show()`), and may be hidden again at any point (`GUI_CURSOR_Hide()`).

3.4 Memory devices

A memory device is a hardware-independent destination device for drawing operations.

If a memory device is created (by calling `GUI_MEMDEV_Create()`) then validated (by calling `GUI_MEMDEV_Select()`), all drawing operations are executed in memory. The final result is displayed on the screen only when all operations have been finished. This action is done by calling `GUI_MEMDEV_CopyToLCD()`.

Memory devices can be used:

- to prevent flickering effect (due to direct drawing on the display),
- as containers for decompressed images,
- for rotating (`GUI_MEMDEV_Rotate()`) and scaling operations (*Figure 7*),
- for fading operations,
- for window animations,
- for transparency effects.

Figure 7. Scaling and rotation effect using memdev

Since memory devices need a considerable amount of memory (see component “Memory Device” in *Table 7*), it is advised to use an external memory if available.

3.5 Antialiasing

Antialiasing smoothes curves and diagonal lines by “blending” the background color with the foreground one. This is done by adding intermediate colors between object and background.

Shape antialiasing

STemWin supports antialiased drawing of:

- Text (Font Converter is required to create AA fonts)
- Arcs (`GUI_AA_DrawArc()`)
- Circles (`GUI_AA_FillCircle()`)
- Lines (`GUI_AA_DrawLine()`)
- Polygons (`GUI_AA_DrawPolyOutline()` and `GUI_AA_FillPolygon()`)

Figure 8. Shape antialiasing

3.6 Window Manager

Window Manager can be described as:

- A management system for a hierarchic window structure:
  - Each layer has its own desktop window. Each desktop window can have its own hierarchic tree of child windows.
- A callback mechanism-based system:
  - Communication is based on an event-driven callback mechanism. All drawing operations should be done within the WM_PAINT event.
- The foundation of the widget library:
  - All widgets are based on the functions of the Window Manager.

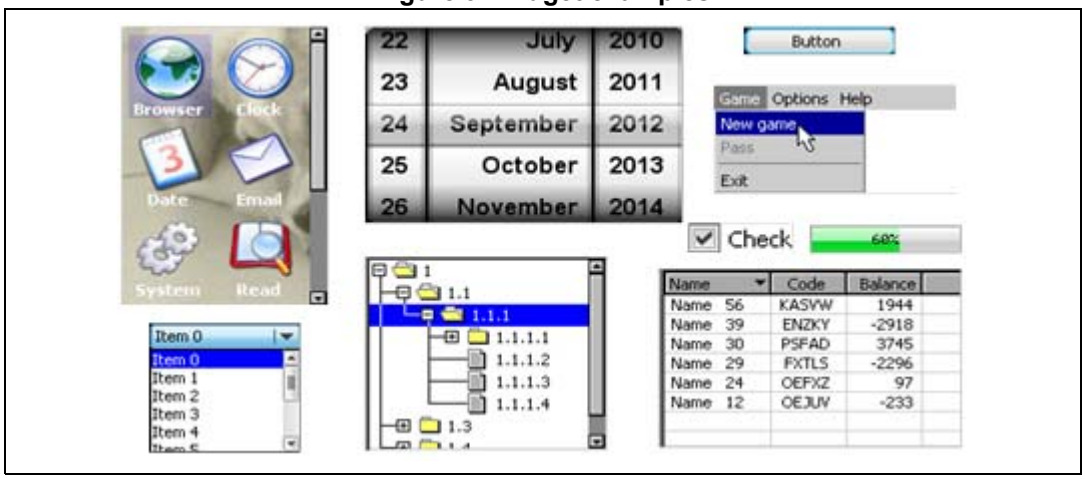
3.7 Widget library

Widgets (**Wi**ndow + Ga**dge**t) are windows with object-type properties. They require the Window Manager.

A list of all widgets available in STemWin Library can be found at: http://www.segger.com

Once a widget is created, it is treated just like any other window. The Window Manager ensures that it is properly displayed (and redrawn) whenever necessary.

Figure 9. Widget examples



Widget creation

Creating a widget can be done with one line of code.

There are basically two ways of creating a widget:

- Direct creation:

Creation functions exist for each widget:

  - `<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.

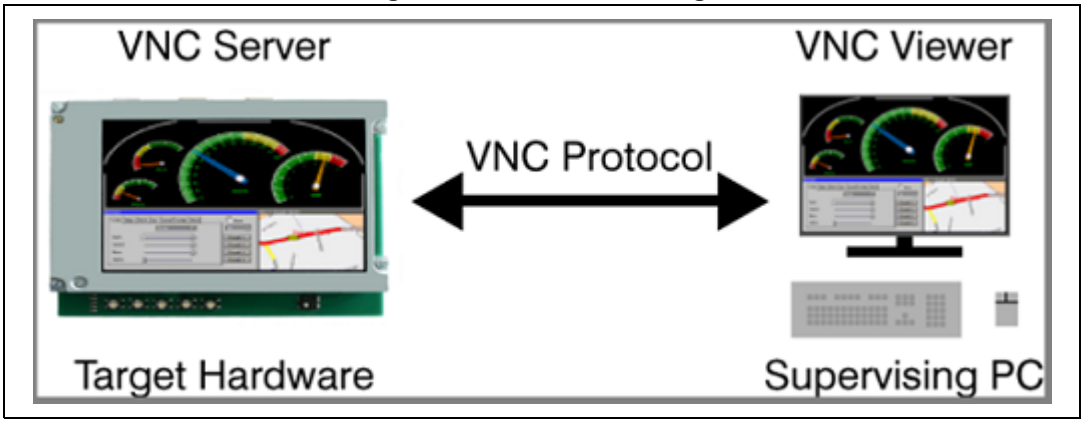
3.8 VNC server

VNC stands for "Virtual Network Computing". The VNC server is used to connect the embedded target to a network PC via TCP/IP, which allows to:

- view the LCD content on the distant PC monitor, and to
- control the embedded environment using the mouse.

In other words, the display contents of the embedded device are visible on the screen of the machine running the client (for example, a network PC); the mouse and keyboard can then be used to control the target.

Figure 10. VNC server usage



3.8.1 Requirements

- An STM32 device with embedded Ethernet IP, such as STM32F107xx, STM32F2x7xx or STM32F4x7xx.
- A TCP/IP stack should be present in the target. In the delivered demo, LwIP is used.
- The VNC server should run as a separate thread. Therefore, a multitasking system is required to use the emWin VNC server. In our case, the demo package runs with FreeRTOS.
- A VNC viewer (such as RealVNC, TightVNC, UltraVNC…) should be present in the supervising PC.

3.8.2 Process description

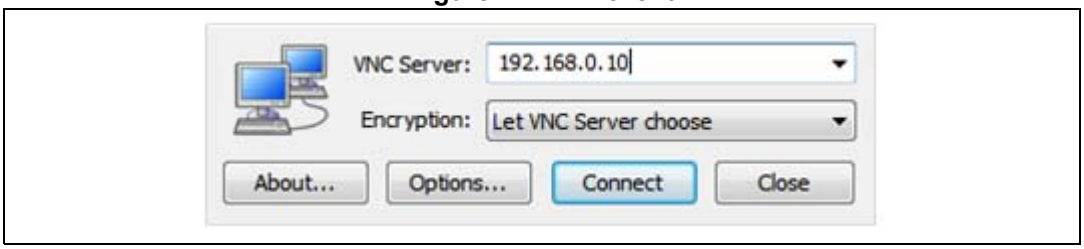
- Connect the target hardware to the network (or to the PC, if a local connection is needed) via an Ethernet cable.
- Run the demo.
  - The VNC_Server_task is a sub-task of the Background_Task.
  - After hardware (LEDs, Touch Screen, SRAM…) and GUI initialization, the TCP/IP (LwIP) stack is also initialized.
  - Then GUI_VNC_X_StartServer() is called to:

  a) initialize the VNC context and attach it to a layer,

  b) create a task for the VNC server, which listens on port 5900 until an incoming connection is detected and then runs the actual server (by calling `GUI_VNC_Process()`).
- If DHCP is enabled ("`#define USE_DHCP`" in main.h):
  - Wait for the IP address to be assigned by the DHCP server; it will be displayed in the "VNC Server" page (just after the "Intro" page).
  - If it is impossible to retrieve any IP address (DHCP timeout), a predefined static IP address is assigned and displayed.
- If DHCP is disabled (or in case of DHCP timeout):
  - Wait for a static IP address to be displayed.
  - Configure the IP address and the subnet mask of the PC with the same class address as used in the target hardware.
- Start the VNC viewer.
  - Connect to the IP address of the target hardware (see *Figure 11*).
  - The demo is then displayed on the PC.
- Using the VNC viewer, the user can:
  - Watch the running demo on the PC monitor (live streaming);
  - Control the target hardware from the PC (using the mouse);
  - Take screenshots of the demo (if needed for a manual, for example).

Note: Breaking the viewer's connection to the server and then reconnecting does not result in any loss of data.

Figure 11. VNC client



3.9 Fonts

The most common fonts are included in STemWin Library as a standard font package. All of them contain the ASCII character set and most of them also the ISO 8859-1 characters.

A complete list of the embedded fonts is shown below (taken from GUI.h).

Note: The STemWin Library default font is GUI_Font6x8.

Fonts included in STemWin library

```
//
// Proportional fonts
//
#define GUI_FONT_8_ASCII        &GUI_Font8_ASCII
#define GUI_FONT_8_1            &GUI_Font8_1
#define GUI_FONT_10S_ASCII      &GUI_Font10S_ASCII
#define GUI_FONT_10S_1          &GUI_Font10S_1
#define GUI_FONT_10_ASCII       &GUI_Font10_ASCII
#define GUI_FONT_10_1           &GUI_Font10_1
#define GUI_FONT_13_ASCII       &GUI_Font13_ASCII
#define GUI_FONT_13_1           &GUI_Font13_1
#define GUI_FONT_13B_ASCII      &GUI_Font13B_ASCII
#define GUI_FONT_13B_1          &GUI_Font13B_1
#define GUI_FONT_13H_ASCII      &GUI_Font13H_ASCII
#define GUI_FONT_13H_1          &GUI_Font13H_1
#define GUI_FONT_13HB_ASCII     &GUI_Font13HB_ASCII
#define GUI_FONT_13HB_1         &GUI_Font13HB_1
#define GUI_FONT_16_ASCII       &GUI_Font16_ASCII
#define GUI_FONT_16_1           &GUI_Font16_1
#define GUI_FONT_16_HK          &GUI_Font16_HK
#define GUI_FONT_16_1HK         &GUI_Font16_1HK
#define GUI_FONT_16B_ASCII      &GUI_Font16B_ASCII
#define GUI_FONT_16B_1          &GUI_Font16B_1
#define GUI_FONT_20_ASCII       &GUI_Font20_ASCII
#define GUI_FONT_20_1           &GUI_Font20_1
#define GUI_FONT_20B_ASCII      &GUI_Font20B_ASCII
#define GUI_FONT_20B_1          &GUI_Font20B_1
#define GUI_FONT_24_ASCII       &GUI_Font24_ASCII
#define GUI_FONT_24_1           &GUI_Font24_1
#define GUI_FONT_24B_ASCII      &GUI_Font24B_ASCII
#define GUI_FONT_24B_1          &GUI_Font24B_1
#define GUI_FONT_32_ASCII       &GUI_Font32_ASCII
#define GUI_FONT_32_1           &GUI_Font32_1
#define GUI_FONT_32B_ASCII      &GUI_Font32B_ASCII
#define GUI_FONT_32B_1          &GUI_Font32B_1

//
```

```
// Proportional fonts, framed
//
#define GUI_FONT_20F_ASCII      &GUI_Font20F_ASCII

//
// Monospaced
//
#define GUI_FONT_4X6            &GUI_Font4x6
#define GUI_FONT_6X8            &GUI_Font6x8
#define GUI_FONT_6X8_ASCII      &GUI_Font6x8_ASCII
#define GUI_FONT_6X8_1          &GUI_Font6x8_1
#define GUI_FONT_6X9            &GUI_Font6x9
#define GUI_FONT_8X8            &GUI_Font8x8
#define GUI_FONT_8X8_ASCII      &GUI_Font8x8_ASCII
#define GUI_FONT_8X8_1          &GUI_Font8x8_1
#define GUI_FONT_8X9            &GUI_Font8x9
#define GUI_FONT_8X10_ASCII     &GUI_Font8x10_ASCII
#define GUI_FONT_8X12_ASCII     &GUI_Font8x12_ASCII
#define GUI_FONT_8X13_ASCII     &GUI_Font8x13_ASCII
#define GUI_FONT_8X13_1         &GUI_Font8x13_1
#define GUI_FONT_8X15B_ASCII    &GUI_Font8x15B_ASCII
#define GUI_FONT_8X15B_1        &GUI_Font8x15B_1
#define GUI_FONT_8X16           &GUI_Font8x16
#define GUI_FONT_8X17           &GUI_Font8x17
#define GUI_FONT_8X18           &GUI_Font8x18
#define GUI_FONT_8X16X1X2       &GUI_Font8x16x1x2
#define GUI_FONT_8X16X2X2       &GUI_Font8x16x2x2
#define GUI_FONT_8X16X3X3       &GUI_Font8x16x3x3
#define GUI_FONT_8X16_ASCII     &GUI_Font8x16_ASCII
#define GUI_FONT_8X16_1         &GUI_Font8x16_1

//
// Digits
//
#define GUI_FONT_D24X32         &GUI_FontD24x32
#define GUI_FONT_D32            &GUI_FontD32
#define GUI_FONT_D36X48         &GUI_FontD36x48
#define GUI_FONT_D48            &GUI_FontD48
#define GUI_FONT_D48X64         &GUI_FontD48x64
#define GUI_FONT_D64            &GUI_FontD64
#define GUI_FONT_D60X80         &GUI_FontD60x80
#define GUI_FONT_D80            &GUI_FontD80

//
// Comic fonts
```

```
//
#define GUI_FONT_COMIC18B_ASCII &GUI_FontComic18B_ASCII
#define GUI_FONT_COMIC18B_1     &GUI_FontComic18B_1
#define GUI_FONT_COMIC24B_ASCII &GUI_FontComic24B_ASCII
#define GUI_FONT_COMIC24B_1     &GUI_FontComic24B_1
```

In most cases, those fonts are found sufficient. However, if needed, STemWin also supports several external font formats:

- System Independent Font (SIF) format
- External Bitmap Font (XBF) format
- TrueType Font (TTF) format

For those, STemWin Library includes a rich font API. See *Table 4*.

Table 4. Font API

Routine	Description
C file related font functions	
GUI_SetDefaultFont()	Sets the default font.
GUI_SetFont()	Sets the current font.
'SIF' file related font functions	
GUI_SIF_CreateFont()	Creates and selects a font by passing a pointer to system-independent font data.
GUI_SIF_DeleteFont()	Deletes a font previously created by GUI_SIF_CreateFont().
'TTF' file related font functions	
GUI_TTF_CreateFont()	Creates a GUI font from a TTF font file.
GUI_TTF_DestroyCache()	Destroys the cache of the TTF engine.
GUI_TTF_Done()	Frees all dynamically allocated memory of the TTF engine.
GUI_TTF_GetFamilyName()	Returns the family name of the font.
GUI_TTF_GetStyleName()	Returns the style name of the font.
GUI_TTF_SetCacheSize()	Can be used to set the default size of the TTF cache.
'XBF' file related font functions	
GUI_XBF_CreateFont()	Creates and selects a font by passing a pointer to a callback function, which then extracts data from the XBF font file.
GUI_XBF_DeleteFont()	Deletes a font previously created by GUI_XBF_CreateFont().
Common font-related functions	
GUI_GetCharDistX()	Returns the width in pixels (X-size) of a specified character in the current font.
GUI_GetFont()	Returns a pointer to the currently selected font.
GUI_GetFontDistY()	Returns the Y-spacing of the current font.
GUI_GetFontInfo()	Returns a structure containing font information.
GUI_GetFontSizeY()	Returns the height in pixels (Y-size) of the current font.

Table 4. Font API (continued)

Routine	Description
GUI_GetLeadingBlankCols()	Returns the number of leading blank pixel columns of the given character.
GUI_GetStringDistX()	Returns the X-size of a text using the current font.
GUI_GetTextExtend()	Evaluates the size of a text using the current font
GUI_GetTrailingBlankCols()	Returns the number of trailing blank pixel columns of the given character.
GUI_GetYDistOfFont()	Returns the Y-spacing of a particular font.
GUI_GetYSizeOfFont()	Returns the Y-size of a particular font.
GUI_IsInFont()	Evaluates whether a specified character belongs to a particular font.
GUI_SetDefaultFont()	Sets the default font to be used after GUI_Init().

3.10 GUIBuilder

The GUIBuilder is a tool for easily creating dialogs: instead of writing source code, the user can place and size widgets by drag and drop. Additional properties can be added via a pop-up menu. Fine tuning can be done by editing the properties of the widgets.

The GUIBuilder then generates some dialog C code that can be either customized or integrated as is in the project.

Figure 12. The GUIBuilder application

3.10.1 Basic usage of the GUIBuilder

- Start with the FRAMEWIN or WINDOW widget: only those widgets are able to serve as parent windows for a dialog.
- Place the widgets within the parent window: the widgets can be placed and sized by moving them with the mouse and/or by editing the properties in the property window.
- Configure the widgets: the pop-up menu shows the available options.
- Save the dialog: each dialog is saved in a separate file. The filenames are generated automatically, based on the name of the parent window.

3.10.2 Creation routine

The file generated using GUIBuilder contains a creation routine for the dialog. The routine name includes the name of the parent window: `WM_HWIN Create<WindowName>(void);`

Simply call the following routine to create the dialog:

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

3.10.3 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 GUIBuilder, the code must be between these comments.

Note: Adding code outside the user code comments makes the file unreadable for the GUIBuilder.

3.10.4 Callback routine

The 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

It 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.

4 Performance and footprint

4.1 LCD driver performance

Table 5 lists a set of tests used to measure the speed of the display driver.

Table 5. Speed test list

Test name	Description
Test 1: Filling	Measures the speed of filling. An area of 64 * 64 pixels is filled with different colors.
Test 2: Small fonts	Measures the speed of small character output. An area of 60 * 64 pixels is filled with small-character text.
Test 3: Big fonts	Measures the speed of big character output. An area of 65 * 48 pixels is filled with big-character text.
Test 4: Bitmap 1 bpp	Measures the speed of 1 bpp bitmaps. An area of 58 * 8 pixels is filled with a 1 bpp bitmap.
Test 5: Bitmap 2 bpp	Measures the speed of 2 bpp bitmaps. An area of 32 * 11 pixels is filled with a 2 bpp bitmap.
Test 6: Bitmap 4 bpp	Measures the speed of 4 bpp bitmaps. An area of 32 * 11 pixels is filled with a 4 bpp bitmap.
Test 7: Bitmap 8 bpp	Measures the speed of 8 bpp bitmaps. An area of 32 * 11 pixels is filled with an 8 bpp bitmap.
Test 8: Bitmap 16 bpp	Measures the speed of 16 bpp bitmaps. An area of 64 * 8 pixels is filled with an 8 bpp bitmap.

The tests were done on the STM324xG-EVAL and STM324x9I-EVAL boards using respectively FlexColor and Lin drivers.

The results are shown in *Table 6*.

Table 6. Speed test for the FlexColor and Lin drivers

Test name	FlexColor	Lin
Test 1: Filling	7.48 M	73.47 M
Test 2: Small fonts	1.57 M	4.16 M
Test 3: Big fonts	2.35 M	5.96 M
Test 4: Bitmap 1bpp	3.23 M	8.81 M
Test 5: Bitmap 2bpp	2.28 M	6.29 M
Test 6: Bitmap 4bpp	2.22 M	6.13 M
Test 7: Bitmap 8bpp	1.17 M	9.71 M
Test 8: Bitmap 16bpp	5.57 M	4.55 M

M=megapixels/second

4.2 STemWin footprint

The operation area of STemWin varies widely, depending primarily on the application and features used. In the following sections, memory requirements of various modules are listed, as well as the memory requirements of example applications.

The following table shows the memory requirements of the main components of STemWin. These values depend a lot on the compiler options, the compiler version and the used CPU. Note that the listed values are the requirements of the basic functions of each module.

Table 7. Module footprint

Component	ROM	RAM	Description
Windows Manager	6.2 Kbytes	2.5 Kbytes	Additional memory requirements of basic application when using the Window Manager.
Memory Devices	4.7 Kbytes	7 Kbytes	Additional memory requirements of a basic application when using memory devices.
Antialiasing	4.5 Kbytes	2 * LCD_XSIZE	Additional memory requirements for the antialiasing software item.
Driver	2 – 8 Kbytes	20 bytes	The memory requirements of the driver depend on the configured driver and whether a data cache is used or not. With a data cache, the driver requires more RAM.
Multilayer	2 – 8 Kbytes	-	If working with a multi layer or a multi display configuration, additional memory is required for each additional layer, because each requires its own driver.
Core	5.2 Kbytes	80 bytes	Memory requirements of a typical application without using additional software items.
JPEG	12 Kbytes	38 Kbytes	Basic routines for drawing JPEG files.
GIF	3.3 Kbytes	17 Kbytes	Basic routines for drawing GIF files.
Sprites	4.7 Kbytes	16 bytes	Routines for drawing sprites and cursors.
Font	1 – 4 Kbytes	-	Depends on the font size to be used.

Table 8. Widget footprint

Component	ROM	RAM	Description
BUTTON	1 Kbyte	40 bytes	*1
CHECKBOX	1 Kbyte	52 bytes	*1
DROPDOWN	1.8 Kbytes	52 bytes	*1
EDIT	2.2 Kbytes	28 bytes	*1
FRAMEWIN	2.2 Kbytes	12 bytes	*1
GRAPH	2.9 Kbytes	48 bytes	*1
GRAPH_DATA_XY	0.7 Kbytes	-	*1
GRAPH_DATA_YT	0.6 Kbytes	-	*1
HEADER	2.8 Kbytes	32 bytes	*1
LISTBOX	3.7 Kbytes	56 bytes	*1
LISTVIEW	3.6 Kbytes	44 bytes	*1
MENU	5.7 Kbytes	52 bytes	*1
MULTIEDIT	7.1 Kbytes	16 bytes	*1
MULTIPAGE	3.9 Kbytes	32 bytes	*1
PROGBAR	1.3 Kbytes	20 bytes	*1
RADIOBUTTON	1.4 Kbytes	32 bytes	*1
SCROLLBAR	2 Kbytes	14 bytes	*1
SLIDER	1.3 Kbytes	16 bytes	*1
TEXT	1 Kbyte	16 bytes	*1
CALENDAR	0.6 Kbyte	32 bytes	*1

5 FAQs (Frequently Asked Questions)

This section gathers some of the most frequent questions STemWin Library package users may ask, and provides some solutions and tips.

Table 9. FAQs

No.	Question	Answer/solution
1	Are all the STemWin features included in the package?	Yes. The delivered locked binaries where compiled with all the features enabled.
2	What is the STemWin Library configuration (during the binary generation)?	The file GUIConf.h (located under Libraries\STemWinLibraryXYZ\Config\) was used to generate the STemWin binaries. The content of that file is as follows: #define GUI_NUM_LAYERS (2) #define GUI_DEFAULT_FONT &GUI_Font6x8 #define GUI_SUPPORT_TOUCH (1) #define GUI_SUPPORT_MOUSE (1) #define GUI_SUPPORT_UNICODE (1) #define GUI_WINSUPPORT (1) #define GUI_SUPPORT_MEMDEV (1) #define GUI_SUPPORT_AA (1) #define WM_SUPPORT_STATIC_MEMDEV (1)
3	Isn't the delivered binary too large?	No. It depends on the application. The compiler considers only called parts from the external functions; thus, non-used resources are not included in the final application size.
4	How can a new LCD controller be supported?	To support any kind of LCD controller, the user should implement two configuration files: LCDConf.c/.h GUIConf.c/.h *Section 3.1* describes in detail the content of those files.
5	Is it mandatory to use the FreeRTOS operating system?	No. Any other operating system can be used. But then a corresponding GUI_X_OS.c file is needed (see *Section 3.1.3*).
6	The project is compiled without errors but, when running the application, the display does not work.	This issue may be caused by one of the following: Stack size is too low. Wrong initialization of the display controller. Wrong configuration of the display interface.

6 Revision history

Table 10. Document revision history

Date	Revision	Changes
19-Jul-2013	1	Initial release.
07-Feb-2014	2	- The use of STemWin generic version (XYZ). - The support of STM324x9I-EVAL board.
20-Mar-2014	3	- Added reference to STM32CubeF2 and STM32CubeF4
25-Jun-2014	4	Added STM32CubeF3 in the list of applicable software.