Introduction

The STM32Cube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube™ covers the STM32 portfolio.

The STM32CubeF4 demonstration platform comes on top of the STM32Cube™ as a firmware package that offers a full set of software components based on a modules architecture allowing re-using them separately in standalone applications. All these modules are managed by the STM32CubeF4 demonstration kernel allowing to dynamically add new modules and access to common resources (storage, graphical components and widgets, memory management, Real-Time operating system).

The STM32CubeF4 demonstration platform is built around the powerful graphical library STemWin and the FreeRTOS real-time operating system and uses almost the whole STM32 capability to offer a large scope of usage based on the STM32Cube HAL BSP and several middleware components.

The architecture was defined with the goal of making from the STM32CubeF4 demonstration core an independent central component which can be used with several RTOS and third party firmware libraries through several abstraction layers inserted between the STM32CubeF4 demonstration core and the several modules and libraries working around.

The STM32CubeF4 demonstration supports STM32F4xx devices and runs on STM324x9I-EVAL, STM324xG-EVAL, STM32F429I-Discovery and STM32446E-EVAL boards.
Contents

1       STM32Cube™ overview ............................................. 7
2       Global architecture .............................................. 8
3       Kernel description ................................................ 9
    3.1  Overview .......................................................... 9
    3.2  Kernel initialization ............................................ 10
    3.3  Kernel processes and tasks .................................... 10
    3.4  Kernel graphical aspect ........................................ 11
    3.5  Kernel menu management ....................................... 13
    3.6  Modules manager .................................................. 16
    3.7  Direct open feature ............................................. 18
    3.8  Backup and settings configuration ............................ 19
    3.9  Storage units ..................................................... 21
    3.10 Clock and Date ................................................... 25
    3.11 Memory Management ............................................. 26
    3.12 Demonstration repository ..................................... 28
    3.13 Kernel components ............................................. 30
    3.14 Kernel core files ............................................... 30
    3.15 Hardware settings ............................................... 31

4       How to create a new module .................................... 32
    4.1  Creating the graphical aspect ................................. 32
    4.2  Graphics customization ......................................... 33
    4.3  Module implementation ......................................... 33
    4.4  Adding a module to the main desktop ....................... 34
    4.5  Module’s direct open ............................................ 35

5       Demonstration customization and configuration ............... 36
    5.1  LCD configuration ............................................... 36
    5.2  Layers management ............................................... 36
    5.3  Touchscreen calibration ......................................... 37
# Contents

5.4 BSP customization ................................................. 38  
5.4.1 SDRAM configuration ........................................ 38  
5.4.2 Touch screen configuration ................................. 39  

6 Performance ......................................................... 41  
6.1 Multi buffering features ...................................... 41  
6.2 Multi layers feature ............................................ 41  
6.3 Hardware acceleration ......................................... 42  

7 Footprint .............................................................. 44  
7.1 Kernel footprint ................................................. 44  
7.2 Module footprint ................................................ 44  
7.3 STemWin Features resources ................................. 45  
7.3.1 JPEG decoder ................................................. 45  
7.3.2 GUI Components ............................................. 45  

8 Demonstration functional description ......................... 48  
8.1 Kernel ............................................................. 48  
8.1.1 CPU Usage .................................................... 49  
8.1.2 Kernel Log .................................................... 49  
8.1.3 Process Viewer .............................................. 50  
8.2 Modules ........................................................... 50  
8.2.1 System ......................................................... 50  
8.2.2 File Browser .................................................. 53  
8.2.3 Game .......................................................... 55  
8.2.4 Benchmark .................................................... 56  
8.2.5 Audio .......................................................... 56  
8.2.6 Video .......................................................... 60  
8.2.7 USB Mass storage Device ................................. 66  
8.2.8 Camera ........................................................ 68  
8.2.9 Image viewer .................................................. 70  

9 Revision history ...................................................... 73
## List of tables

Table 1. File system interface: physical storage control functions ........................................... 21  
Table 2. File system interface APIs ......................................................................................... 22  
Table 3. APIs from the RTC module ......................................................................................... 25  
Table 4. APIs from the memory manager module ..................................................................... 28  
Table 5. Kernel components list ............................................................................................. 30  
Table 6. Kernel core files list .................................................................................................. 30  
Table 7. Jumpers for different demonstration boards ............................................................... 31  
Table 8. LCD frame buffer locations ......................................................................................... 39  
Table 9. Camera frame buffer locations .................................................................................... 39  
Table 10. Kernel files footprint ............................................................................................... 44  
Table 11. Modules footprint .................................................................................................... 44  
Table 12. RAM requirements for some JPEG resolutions ......................................................... 45  
Table 13. MemoSTemWin components memory requirements ............................................... 46  
Table 14. Widget memory requirements .................................................................................... 46  
Table 15. Available settings ..................................................................................................... 52  
Table 16. Data structure for audio ............................................................................................ 57  
Table 17. Audio module controls ............................................................................................ 59  
Table 18. Video module controls ............................................................................................. 61  
Table 19. Batch files description .............................................................................................. 64  
Table 20. Variables description ............................................................................................... 65  
Table 21. Parameters description ............................................................................................. 65  
Table 22. Data structure for USBD module ............................................................................... 67  
Table 23. USBD module controls ............................................................................................. 67  
Table 24. Camera module controls .......................................................................................... 70  
Table 25. Image viewer module controls ................................................................................. 72  
Table 26. Document revision history ....................................................................................... 73
List of figures

Figure 1. STM32Cube block diagram ................................................................. 7
Figure 2. STM32Cube architecture ................................................................. 8
Figure 3. Kernel components and services ..................................................... 9
Figure 4. Startup window .............................................................................. 11
Figure 5. Startup window for STM32446E-EVAL demonstration ..................... 12
Figure 6. Main desktop window .................................................................. 12
Figure 7. Main desktop window for STM32446E-EVAL demonstration ............ 13
Figure 8. Status bar ..................................................................................... 14
Figure 9. Status bar for STM32446E-EVAL demonstration ............................. 14
Figure 10. Icon view widget ........................................................................ 15
Figure 11. Icon view widget for STM32446E-EVAL demonstration ................. 15
Figure 12. Functionalities and properties of modules ..................................... 17
Figure 13. Starting file execution ................................................................ 18
Figure 14. Starting file execution for STM32446E-EVAL demonstration ......... 19
Figure 15. Available storage units ................................................................ 21
Figure 16. Software architecture ................................................................ 23
Figure 17. Detection of storage units ............................................................. 24
Figure 18. Detection of storage units for STM32446E-EVAL demonstration ... 24
Figure 19. Setting the time and the date ........................................................ 26
Figure 20. Setting the time and the date for STM32446E-EVAL demonstration 26
Figure 21. Memory heap for STM32CubeF4 demonstration ........................... 27
Figure 22. Folder structure ........................................................................... 29
Figure 23. STM32Cube demonstration boards .............................................. 31
Figure 24. GUI Builder overview ................................................................ 32
Figure 25. Graphics customization ............................................................... 33
Figure 26. Direct open from file browser ..................................................... 35
Figure 27. LCDConf location ....................................................................... 36
Figure 28. k_calibration.c location ............................................................... 37
Figure 29. Calibration steps ......................................................................... 38
Figure 30. SDRAM initialization .................................................................. 39
Figure 31. Touch screen initialization .............................................................. 40
Figure 32. Example of tearing effect ............................................................... 41
Figure 33. Independent layer management .................................................... 42
Figure 34. CPU usage display ..................................................................... 48
Figure 35. CPU usage ............................................................................... 49
Figure 36. Example of Log messages ............................................................ 49
Figure 37. Process viewer .......................................................................... 50
Figure 38. Demonstration global information ............................................... 51
Figure 39. Demonstration general settings .................................................... 51
Figure 40. Clock setting .............................................................................. 52
Figure 41. File browser ................................................................................ 53
Figure 42. File browser module architecture .............................................. 54
Figure 43. File opening from browser ............................................................ 54
Figure 44. File properties display ................................................................. 55
Figure 45. Reversi game ............................................................................. 55
Figure 46. Benchmarking ........................................................................... 56
Figure 47. Audio player module architecture ............................................. 57
Figure 48. Audio player module startup ....................................................... 58
List of figures

Figure 49. Video player module architecture .......................................................... 60
Figure 50. Video player module startup ................................................................. 61
Figure 51. EMF generation environment ................................................................. 63
Figure 52. JPEG2Movie overview ........................................................................... 63
Figure 53. EMF file generation ............................................................................... 64
Figure 54. USBD module architecture ................................................................. 66
Figure 55. USBD module startup ........................................................................... 67
Figure 56. Camera module architecture ................................................................. 69
Figure 57. Camera module startup ......................................................................... 69
Figure 58. Image viewer architecture ................................................................... 71
Figure 59. Image viewer startup ............................................................................ 72
1 STM32Cube™ overview

The STM32Cube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube™ covers the STM32 portfolio. STM32Cube™ Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for STM32F4 series)
  - The STM32CubeF4 HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

Figure 1. STM32Cube block diagram
2 Global architecture

The STM32CubeF4 demonstration is composed of a central kernel based on a set of firmware and hardware services offered by the STM32Cube middleware and the several evaluation and discovery boards and a set of modules mounted on the kernel and built in a modular architecture. Each module can be reused separately in a standalone application. The full set of modules is managed by the Kernel which provides access to all common resources and facilitates the addition of new modules as shown in Figure 2.

Each module should provide the following functionalities and proprieties:
1. Icon and graphical aspect characteristics.
2. Method to startup the module.
3. Method to close down safely the module (example: Hot unplug for Unit Storage)
4. Method to manage low power mode
5. The module application core (main module process)
6. Specific configuration
7. Error management

---

**Figure 2. STM32Cube architecture**
3 Kernel description

3.1 Overview

The role of the demonstration kernel is mainly to provide a generic platform that control and monitor all the application processes, the kernel provides a set of friendly user APIs and services that allow to the user modules to have access to all the hardware and firmware resources and provide the following tasks and services:

- Hardware and modules initialization:
  - BSP initialization (LEDs, SDRAM, Touch screen, CRC, SRAM, RTC, QSPI and audio)
  - GUI initialization and Touch screen calibration
- Memory management
- Kernel log
- Graphical resources and main menu management.
- Storage managements (USB Disk flash and microSD)
- System monitoring and settings
- Time and date resources management
- File browsing and contextual menu
- CPU utilities (CPU usage, running tasks)

![Figure 3. Kernel components and services](image)
3.2 Kernel initialization

The first task of the kernel is to initialize the hardware and firmware resources to make them available to its internal processes and the modules around it. The kernel starts by initializing the HAL, system clocks and then the hardware resources needed during the middleware components:

- LEDs and Touchscreen
- SDRAM/SRAM
- Backup SRAM
- RTC
- QSPI Flash memory for STM32446E-EVAL demonstration board
- Audio Interface

Once the low level resources are initialized, the kernel performs the STemWin GUI library initialization and prepares the following common services:

- Memory manager
- Storage units
- Modules manager
- Kernel Log

Upon full initialization phase, the kernel adds and links the system and user modules to the demonstration core.

3.3 Kernel processes and tasks

The kernel is composed of two main tasks managed by FreeRTOS through the CMSIS-OS wrapping layer:

- GUI Thread: this task Initializes the demonstration main menu and then handles the graphical background task when requested by the STemWin;

```c
/* Brief Start task */
static void GUIThread(void const * argument) {
    /* Show the main menu */
    k_InitMenu();
    /* Gui background Task */
    while(1) {
        GUI_Thread();
        canDelay(10);
    }
}
```
• Timer Callback: this is the callback of the Timer managing periodically the touch screen state, the Timer callback is called periodically each 100 milliseconds.

3.4 Kernel graphical aspect

The STM32CubeF4 demonstration is built around the STemWin Graphical Library, based on SEGGER emWin one. 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.

The graphical aspect of the STM32CubeF4 demonstration is divided into two main graphical components:

• the startup window (Figure 4 and Figure 5): showing the progress of the hardware and software initialization;
• the main desktop (shown in Figure 6 and in Figure 7), that handles the main demonstration menu and the numerous kernel and modules control.

Figure 4. Startup window
Figure 5. Startup window for STM32446E-EVAL demonstration

Figure 6. Main desktop window
3.5 Kernel menu management

The main demonstration menu is initialized and launched by the GUI thread. Before the initialization of the menu the following actions are performed:

- Draw the background image.
- Create the status bar.
- Restore general settings from backup memory.
- Setup the main desktop callback to manage main window messages.

The main desktop is built around two main graphical components:

- The status bar (Figure 8 and Figure 9): indicates the storage units connection status, current time and date and a system button to allow to get system information like (running task, CPU load, and kernel log).
- The icon view widget (Figure 10 and Figure 11): contains the icons associated to added modules. User can launch a module by a simple click on the module icon.
Figure 8. Status bar

Figure 9. Status bar for STM32446E-EVAL demonstration
Figure 10. Icon view widget

Figure 11. Icon view widget for STM32446E-EVAL demonstration
A module is launched on simple click on the associated icon by calling to the startup function in the module structure; this is done when a WM_NOTIFICATION_RELEASED message arrives to the desktop callback with ID_ICONVIEW_MENU:

```c
/* Callback routine of desktop window. */
"param pMsg: pointer to data structure of type WM_MESSAGE

static void cbNotify(WM_MESSAGE *pMsg) {
  (...
  switch (pMsg->MsgId)
  { 
    case WM_NOTIFY_PARENT:
      Id = WM_GetId(pMsg->hWinSrc);
      NCode = pMsg->Data.v;
      switch (NCode)
      {
        case WM_NOTIFICATION_RELEASED:
          if (Id == ID_ICONVIEW_MENU)
          {
            if(sel < k_ModuleGetNumber())
              module_prop[sel].module->startup(pMsg->hMin, 0, 26);
            break;
          (...)
        default:
          break;
        break;
      }
  default:
    WM_DefaultProc(pMsg);
  }
```

### 3.6 Modules manager

The modules are managed by the kernel; the latter is responsible of initializing the modules, initializing hardware and GUI resources relative to the modules and initializing the common resources such as the storage Unit, the graphical widgets and the system menu.

Each module should provide the following functionalities and proprieties:

1. Icon and graphical component structure.
2. Method to startup the module.
3. Method to close down safely the module (example: Hot unplug for MS flash disk)
4. Method to manage low power mode (optional)
5. The Application task
6. The module background process (optional)
7. Remote control method (optional)
8. Specific configuration
9. Error management

Figure 12. Functionalities and properties of modules

The modules could be added in run time to the demonstration and can use the common kernel resources. The following code shows how to add a module to the demonstration:

```c
/* Add Modules*/
k_ModuleInit();

k_ModuleAdd(svideo_player);
k_ModuleAdd(simage_browser);
k_ModuleAdd(saudio_player);
k_ModuleAdd(scamera_capture);
k_ModuleAdd(isystem_info);
k_ModuleAdd(sinio_player);
k_ModuleAdd(afile_browser);
k_ModuleAdd(ccpu_bench);
k_ModuleAdd(legame_board);
k_ModuleAdd(susb_device);
```

A module is a set of function and data structures that are defined in a data structure that provides all the information and pointers to specific methods and functions to the kernel. This later checks the integrity and the validity of the module and inserts its structure into a module table.

Each module is identified by a unique ID. When two modules have the same UID, the Kernel rejects the second one. The module structure is defined as follows:

```c
typedef struct
{
  uint8_t id;
  const char *name;
  GUI_CONST_STORAGE GUI_BITMAP *icon;
  void (*startup) (WM_HWIN, uint16_t, uint16_t);
  void (*DirectOpen) (char *);
}
K_ModuleItem_TypeDef;
```
In this definition:

- **Id**: unique module identifier.
- **Name**: pointer to module name
- **Icon**: pointer to module icon (bitmap format)
- **Startup**: the function that create the module frame and control buttons
- **DirectOpen**: the function that create the module frame and launch the media associated to the file name selected in the file browser linked to a specific file extension.

### 3.7 Direct open feature

The direct open feature allows launching a media module directly from file browser when the extension file match with supported media type. The file extension should be previously associated to a module by using the following code:

```c
/* Add Modules*/
k_ModuleInit();

k_ModuleAdd(s_video_player);
k_ModuleOpenLink(s_video_player, "emf");
```

For STM32446E-EVAL we have:

```c
/* Add Modules*/
k_ModuleInit();

k_ModuleAdd(s_image_browser);
k_ModuleOpenLink(s_image_browser, "jpg" );
k_ModuleOpenLink(s_image_browser, "JPG" );
k_ModuleOpenLink(s_image_browser, "bmp" );
k_ModuleOpenLink(s_image_browser, "BMP" );
```

When the file browser is opened, a simple click on a file will open a contextual menu, that direct file open can be executed, as shown in Figure 13 and in Figure 14.

**Figure 13. Starting file execution**

![Figure 13. Starting file execution](image)

*Note:* The video player module is not supported in STM32446E-EVAL demonstration.
3.8 Backup and settings configuration

The STM32CubeF4 demonstration saves the kernel and modules settings in two different methods:

1. Using the RTC backup register (32 bits data width), in this method the data to be saved should be a 32 bits data and could be defined as a bitfield structure, example:

   ```c
   typedef union
   {
       struct
       {
           uint32_t repeat : 2;
           uint32_t pages : 2;
           uint32_t music : 1;
           uint32_t volume : 6;
           uint32_t reserved : 21;
       }
       b;
   } AudioSettingsTypeDef;
   ```

   The structure could be handled than, by using the two following kernel APIs to save or restore it from the RTC backup registers.

   ```c
   void _BackupSaveParameter(uint32_t address, uint32_t data);
   void _BackupRestoreParameter(uint32_t address);
   ```

2. Using the backup SRAM: the backup SRAM is a memory that the content is not lost when the board is powered down. When available, the backup SRAM is 4 Kbytes size.
and located at address: BKPSRAM_BASE (0x40024000). The backup SRAM could be used as normal RAM to save file paths or big structure example:

```c
#define CAMERA_SAVE_PATH  BKPSRAM_BASE + 0x100

/**
 * @brief Set default folder (saved in backup SRAM)
 * @param path : pointer to the default folder
 */
static void _Check_DefaultPath (uint8_t *path)
{
    if (*((char *)(CAMERA_SAVE_PATH)) == '0') || (*((char *)(CAMERA_SAVE_PATH)) == '1')
    {
        strncpy((char *)path, (char *)(CAMERA_SAVE_PATH), FILENAME_MAX_PATH_SIZE);
    }
    else
    {
        strcpy((char *)path, "0:");
    }
```
3.9 Storage units

The STM32CubeF4 demonstration kernel offers two storage units that can be used to retrieve audio, Image and Video media or to save captured images from the camera (Figure 15).

![Available storage units](image)

**Figure 15. Available storage units**

The two units are initialized during the platform startup and thus they are available to all the modules during the STM32CubeF4 demonstration run time. These two units are accessible through the standard I/O operations offered by the FatFS used in the development platform. The USB Disk flash unit is identified as the Unit 0 and available only if a USB disk flash is connected on the USB FS connector, while the microSD flash is identified as the Unit 1 and available only if the microSD card is connected. The units are mounted automatically when the physical media are connected to the connector on the board.

The implemented functions in the file system interface to deal with the physical storage units are summarized in Table 1.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk_initialize</td>
<td>Initialize disk drive</td>
</tr>
<tr>
<td>disk_read</td>
<td>Interface function for a logical page read</td>
</tr>
<tr>
<td>disk_write</td>
<td>Interface function for a logical page write</td>
</tr>
<tr>
<td>disk_status</td>
<td>Interface function for testing if unit is ready</td>
</tr>
<tr>
<td>disk_ioctl</td>
<td>Control device dependent features</td>
</tr>
</tbody>
</table>
The full APIs functions set given by the file system interface are listed in Table 2:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_mount</td>
<td>Register/Unregister a work area</td>
</tr>
<tr>
<td>f_open</td>
<td>Open/Create a file</td>
</tr>
<tr>
<td>f_close</td>
<td>Close a file</td>
</tr>
<tr>
<td>f_read</td>
<td>Read file</td>
</tr>
<tr>
<td>f_write</td>
<td>Write file</td>
</tr>
<tr>
<td>f_lseek</td>
<td>Move read/write pointer, Expand file size</td>
</tr>
<tr>
<td>f_truncate</td>
<td>Truncate file size</td>
</tr>
<tr>
<td>f_sync</td>
<td>Flush cached data</td>
</tr>
<tr>
<td>f_opendir</td>
<td>Open a directory</td>
</tr>
<tr>
<td>f_readdir</td>
<td>Read a directory item</td>
</tr>
<tr>
<td>f_getfree</td>
<td>Get free clusters</td>
</tr>
<tr>
<td>f_stat</td>
<td>Get file status</td>
</tr>
<tr>
<td>f_mkdir</td>
<td>Create a directory</td>
</tr>
<tr>
<td>f_unlink</td>
<td>Remove a file or directory</td>
</tr>
<tr>
<td>f_chmod</td>
<td>Change attribute</td>
</tr>
<tr>
<td>f_utime</td>
<td>Change timestamp</td>
</tr>
<tr>
<td>f_rename</td>
<td>Rename/Move a file or directory</td>
</tr>
<tr>
<td>f_mkfs</td>
<td>Create a file system on the drive</td>
</tr>
<tr>
<td>f_forward</td>
<td>Forward file data to the stream directly</td>
</tr>
<tr>
<td>f_chdir</td>
<td>Change current directory</td>
</tr>
<tr>
<td>f_chdrive</td>
<td>Change current drive</td>
</tr>
<tr>
<td>f_getcwd</td>
<td>Retrieve the current directory</td>
</tr>
<tr>
<td>f_gets</td>
<td>Read a string</td>
</tr>
<tr>
<td>f_putchar</td>
<td>Write a character</td>
</tr>
<tr>
<td>f_puts</td>
<td>Write a string</td>
</tr>
<tr>
<td>f_printf</td>
<td>Write a formatted string</td>
</tr>
</tbody>
</table>

For the FAT FS file system, the page size is fixed to 512 bytes. USB disk flashes with higher page size are not supported.
The Storage units are built around the USB host library in high speed and the microSD BSP drivers; the software architecture is shown in Figure 16.

**Figure 16. Software architecture**

The FatFS is mounted upon the USB Host mass storage class and the SD BSP driver to allow an abstract access to the physical media through standard I/O methods.
The storage units' presence detection is handled internally by the kernel and the status bar shows the icons of the available media, as shown in Figure 17 and in Figure 18.

Figure 17. Detection of storage units

Figure 18. Detection of storage units for STM32446E-EVAL demonstration
3.10 Clock and Date

The clock and date are managed by the RTC HAL driver, the RTC module initializes the LSE source clock and provides a set of methods to retrieve date and clock in addition to backup save and restore ones. Table 3 shows the different APIs offered by the RTC module:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_calendarBkupinit</td>
<td>Initialize RTC peripheral (clock and backup registers)</td>
</tr>
<tr>
<td>k_BkupSaveParameter</td>
<td>Save a 32bits word into backup registers</td>
</tr>
<tr>
<td>k_BkupRestoreParameter</td>
<td>Retrieve a saved 32bits word from backup registers</td>
</tr>
<tr>
<td>k_SetTime</td>
<td>Change system time through the RTC_TimeTypeDef</td>
</tr>
<tr>
<td>k_GetTime</td>
<td>Get system time into the RTC_TimeTypeDef structure</td>
</tr>
<tr>
<td>k_SetDate</td>
<td>Change system date through the RTC_DateTypeDef</td>
</tr>
<tr>
<td>k_GetDate</td>
<td>Get system date into the RTC_DateTypeDef structure</td>
</tr>
</tbody>
</table>

The following code shows an example of how to retrieve the system data:

```c
259 #ifdef CAMERA_SaveToFile
260   // Save the data to specified file.
261   // Get a path pointer to the saving path
262   /*
263   CAM_SAVEToFile(path)  
264   
265   // Initialize RTC peripheral (clock and backup registers)
266   TimerTypeDef  
267   DateTypeDef  
268   /* Create a filename */
269   k_GetTime(TIMedata);
270   k_GetDate(sDate);
271   sprintf(filename, "Camera_%02d%02d%02d_%02d%02d%02d.bmc", 
272       Date.Year + 2014,
273       Data.Month,
274       Date.Year + 2014,
275       Data.Month,
276       Data.Year,
277       Data.Month,
278       Data.Year,
279   );
```

The kernel uses the RTC for modules settings saving and getting the time and date, displayed in the status bar of the main desktop. Time and date could be changed through the system module, as shown in Figure 19 and in Figure 20.
3.11 Memory Management

A huge amount of system RAM is allocated to the GUI internal heap, the kernel memory manager is used as a standalone memory allocator for some specific data blocks, like file lists and kernel log buffer.

The kernel memory manager is based on a single memory pool that could be placed anywhere in the additional internal or external memory resources. The memory heap is built
on a contiguous memory blocks managed by the mem_TypeDef structure through a pages table that gather the block status after each memory allocation or de-allocation operations.

For the STM32CubeF4 demonstration, the memory heap is located in the CCM data RAM.

Figure 21. Memory heap for STM32CubeF4 demonstration

The memory manager offers a set of standard high level APIs to allocate and free memory block from the predefined pool. The granularity of the memory allocation is defined by the SIZE_OF_PAGE define, set to 1024 bytes by default and the total number of available blocks depending on the heap size, in the k_mem.h file as shown in the code below.

```
#define MEM_BASE 0x10000000
#define SIZE_OF_PAGE 1024 /* 1 KBytes pages */
#define MAX_PAGE_NUMBER 64 /* Maximum of 64 pages */
```

For STM32446E-EVAL demonstration, the memory heap is located in the external SDRAM memory.
Table 4 shows the different APIs offered by the memory manager module.

Table 4. APIs from the memory manager module

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void k_MemInit(void)</td>
<td>Initialize the memory heap (base address)</td>
</tr>
<tr>
<td>void * k_malloc(size_t s)</td>
<td>Allocate an amount of contiguous memory blocks</td>
</tr>
<tr>
<td>void k_free (void * p)</td>
<td>Free an already allocated amount of RAM blocks</td>
</tr>
</tbody>
</table>

For STM32446E-EVAL demonstration, the different icons of the applications are stored in the external memory QSPI, configured in memory-mapped mode to the STM32 address space, and seen by the system as if it were an internal memory.

This mode provides a direct interface to access data from external SPI memory and thus simplify Software requirements.

### 3.12 Demonstration repository

The STM32Cube is a component in the STM32Cube package. Figure 22 shows the demonstration folder organization:
The demonstration sources are located in the projects folder of the STM32Cube package for each supported board. The sources are divided into five groups described as follows:

1. **Core**: contains the kernel files
2. **Modules**: contains the system and user modules including the graphical aspect and the modules functionalities.
3. **Binary**: demonstration binary file in Hex format
4. **Config**: all middleware's components and HAL configuration files
5. **Project settings**: a folder per tool chain containing the project settings and the linker files.
3.13 Kernel components

Table 5. Kernel components list

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel core</td>
<td>Kernel core and utilities</td>
</tr>
<tr>
<td>Modules</td>
<td>User and system modules</td>
</tr>
<tr>
<td>STM32 HAL Drivers</td>
<td>STM32Cube HAL driver relative to the STM32 device under use</td>
</tr>
<tr>
<td>BSP Drivers</td>
<td>Evaluation board (or discovery kit) BSP drivers</td>
</tr>
<tr>
<td>CMSIS</td>
<td>CMSIS Cortex®-M3/4 Device Peripheral Access Layer System</td>
</tr>
<tr>
<td>FatFS</td>
<td>FATFS File system</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS Real Time Operating System</td>
</tr>
<tr>
<td>STemWin</td>
<td>STemWin Graphical Library</td>
</tr>
<tr>
<td>USB_H_LIBRARY</td>
<td>USB Device Library (Mass Storage Class)</td>
</tr>
<tr>
<td>USBH_LIBRARY</td>
<td>USB Host Library (Mass Storage Class)</td>
</tr>
</tbody>
</table>

3.14 Kernel core files

Table 6. Kernel core files list

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Main program file</td>
</tr>
<tr>
<td>stm32fxxx_it.c</td>
<td>Interrupt handlers for the application</td>
</tr>
<tr>
<td>k_bsp.c</td>
<td>Provides the kernel BSP functions</td>
</tr>
<tr>
<td>k_calibration.c</td>
<td>Touch screen calibration processes</td>
</tr>
<tr>
<td>k_log.c</td>
<td>Kernel Log manager</td>
</tr>
<tr>
<td>k_mem.c</td>
<td>Kernel memory heap manager</td>
</tr>
<tr>
<td>k_menu.c</td>
<td>Kernel menu and desktop manager</td>
</tr>
<tr>
<td>k_module.c</td>
<td>Modules manager</td>
</tr>
<tr>
<td>k_modules_res.c</td>
<td>Common modules resources</td>
</tr>
<tr>
<td>k_RTC.c</td>
<td>RTC and backup manager</td>
</tr>
<tr>
<td>k_startup.c</td>
<td>Demonstration startup windowing process</td>
</tr>
<tr>
<td>k_storage</td>
<td>Storage units manager</td>
</tr>
<tr>
<td>startup_stm32fyyyxx.s</td>
<td>Startup file</td>
</tr>
<tr>
<td>cpu_utils.c</td>
<td>CPU load calculation utility</td>
</tr>
</tbody>
</table>
3.15 Hardware settings

The STM32CubeF4 demonstration supports STM32F4xx devices and runs on the following demonstration boards from STMicroelectronics:

- STM324x9I-EVAL
- STM324xG-EVAL
- STM32F429I-Discovery
- STM32446E-EVAL.

![Figure 23. STM32Cube demonstration boards](image)

<table>
<thead>
<tr>
<th>Board</th>
<th>Jumper</th>
<th>Position description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM324x9I-EVAL</td>
<td>JP16</td>
<td>Not fitted (used for USB device module)</td>
</tr>
<tr>
<td></td>
<td>JP4/JP5</td>
<td>&lt;2-3&gt; (used for Audio demonstration)</td>
</tr>
<tr>
<td></td>
<td>JP8</td>
<td>&lt;2-3&gt; (used for backup domain on battery)</td>
</tr>
<tr>
<td>STM324xG-EVAL</td>
<td>JP16</td>
<td>&lt;2-3&gt; (used for Audio demonstration)</td>
</tr>
<tr>
<td></td>
<td>JP19</td>
<td>&lt;2-3&gt; (used for backup domain on battery)</td>
</tr>
<tr>
<td></td>
<td>JP31</td>
<td>&lt;2-3&gt; (used for USB device module)</td>
</tr>
<tr>
<td>STM32F429I-Discovery</td>
<td>JP3</td>
<td>ON (Power on MCU)</td>
</tr>
<tr>
<td></td>
<td>CN4</td>
<td>ON (Discovery mode)</td>
</tr>
<tr>
<td>STM324446E-EVAL</td>
<td>JP4</td>
<td>&lt;2-3&gt; (used for USB device module)</td>
</tr>
<tr>
<td></td>
<td>JP19</td>
<td>&lt;1-2&gt; (used for audio player module)</td>
</tr>
</tbody>
</table>
4 How to create a new module

A module is composed of two main parts:
- Graphical aspect: the main window frame and module's controls
- Functionalities: module functions and internal processes

4.1 Creating the graphical aspect

The graphical aspect consists of the main frame window in addition to the set of the visual elements and controls (buttons, check boxes, progress bars...) used to control and monitor the module's functionalities.

The STM32CubeF4 demonstration package provides a PC tool; the GUIBuilder (Figure 24) that allows easily and quickly creating the module frame window and all its components in few steps. For more information about the GUI Builder, refer to the emwin User and reference guide (UM03001).

The GUI Builder needs only a few minutes to totally design the module appearances using “drag and drop” commands and then generate the source code file to be included into the application.

The file generated is composed of the following main parts:
- A resource table: it’s a table of type GUI_WIDGET_CREATE_INFO, which specifies all the widgets to be included in the dialog and also their respective positions and sizes.
- A dialog callback routine: described more in detail in section 4.3 (it is referred to as “main module callback routine”).
4.2 Graphics customization

After the basic module graphical appearance is created, it is then possible to customize some graphical elements, such as the buttons, by replacing the standard aspect by the user defined image. To do this, a new element drawing callback should be created and used instead of the original one.

Below an example of a custom callback for the Play button:

```c
363  /**
364  * @brief callback for play button
365  * @param pMsg: pointer to data structure of type WM_MESSAGE
366  * @retval None
367  */
368 static void _cbButton_play(WM_MESSAGE * pMsg) {
369     switch (pMsg->MsgId) {
370         case WM_PAINT:
371             _OnPaint_play(pMsg->hWin);
372             break;
373         default:
374             /* The original callback */
375             BUTTON_Callback(pMsg);
376             break;
377         }
378     }
```

On the code portion above, the _OnPaint_play routine contains just the new button drawing command.

Note that the new callback should be associated to the graphical element at the moment of its creation, as shown below:

![Figure 25. Graphics customization](image-url)

4.3 Module implementation

Once the graphical part of the module is finalized, the module functionalities and processes could be added then. It begins with the creation of the main module structure as defined in Section 3.6: Modules manager.
Then, each module has its own Startup function which simply consists of the graphical module creation, initialization and link to the main callback:

```c
/**
 * @brief Module window Startup
 * @param hWnd: pointer to the parent handle.
 * @param xpos: X position
 * @param ypos: Y position
 * @retval None
 */
static void Startup(HWND hWnd, int xpos, int ypos);
```

In the example above cbDialog refers to the main module callback routine. Its general skeleton is structured like the following:

```c
/**
 * @brief Callback routine of the dialog
 * @param pMsg: pointer to data structure of type WM_MESSAGE
 * @retval None
 */
static void cbDialog(WM_MESSAGE *pMsg)
{
    switch(pMsg->msgId)
    {
    case WM_INIT_DIALOG:
        // Initialize graphical elements and restore backup parameters if any */
    case WM_NOTIFY_PARENT:
    ...
    }
```

The list of windows messages presented in the code portion above (WM_INIT_DIALOG and WM_NOTIFY_PARENT) is not exhaustive, but represents the essential message IDs used:

- "WM_INIT_DIALOG": allows initializing the graphical elements with their respective initial values. It is also possible here to restore the backup parameters (if any) that will be used during the dialog procedure.
- "WM_NOTIFY_PARENT": describes the dialog procedure, for example: define the behavior of each button.

The full list of window messages can be found in the WM.h file.

### 4.4 Adding a module to the main desktop

Once the module appearance and functionality are defined and created, it still only to add the module to the main desktop view, this is done by adding it to the list (structure) of menu items: module_prop[], defined into k_module.h.

To do this, k_ModuleAdd() function should be called just after the module initialization into the main.c file.

Note that the maximum modules number in the demonstration package is limited to 15; this value can be changed by updating MAX_MODULES_NUM defined into k_module.c.
4.5 Module’s direct open

If there is a need to launch the module directly from the file browser contextual menu, an additional method should be added in the module structure for the direct open feature. This callback is often named `_ModuleName_DirectOpen`.

*Figure 26* is an example of how to open a file using the adequate module from the file browser.

**Figure 26. Direct open from file browser**

In the STM32CubeF4 demonstration, there are three modules linked to the file browser contextual menu:

- The **video player**\(^1\), supporting the format:
  - emf
- The **image browser**, supporting the formats:
  - jpg
  - bmp
- The **audio player**, supporting the format:
  - wav.

Then, to link the module to the file browser open menu, the command `k_ModuleOpenLink()` is called after the module is added.

---

1. The video player is not supported by STM32446E-EVAL demonstration.
5 Demonstration customization and configuration

5.1 LCD configuration

The LCD is configured through the LCDConf.c file, see Figure 27. The main configuration items are listed below:

- **Multiple layers:**
  - The number of layers to be used defined using GUI_NUM_LAYERS.
- **Multiple buffering:**
  - If NUM_BUFFERS is set to a value "n" greater than 1, it means that "n" frame buffers will be used for drawing operation (see section 7.1 for impact of multiple buffering on performance).
- **Virtual screens:**
  - If the display area is greater than the physical size of the LCD, NUM_VSCREENS should be set to a value greater than 1. Note that virtual screens and multi buffers are not allowed together.
- **Frame buffers locations:**

The physical location of frame buffer is defined through LCD_LAYERX_FRAME_BUFFER.

Figure 27. LCDConf location

5.2 Layers management

In the STM32CubeF4 demonstration package with the STM324x9I-EVAL and Discovery Kit, GUI_NUM_LAYERS is set to 2 (both layers are used):

- "Layer 0 is dedicated to background display"
- "Layer 1 is used for the main desktop display"

Dedicated layers usage will lighten the CPU load during the refresh tasks.
5.3 Touchscreen calibration

When the demonstration is launched for the first time, the touchscreen needs to be calibrated. A full set of dedicated routines is included in the demonstration package and regrouped into k_calibration.c file (Figure 28).

To do this, after the startup screen is displayed, the user has to follow the displayed calibration instructions by touching the screen at the indicated positions (Figure 29). This will allow getting the physical Touch screen values that will be used to calibrate the screen.
Once this runtime calibration is done, the touch screen calibration parameters are saved to the RTC Backup data registers: RTC_BKP_DR0 and RTC_BKP_DR1, so the next time the application is restarted, these parameters are automatically restored and there is no need to re-calibrate the touchscreen.

5.4 BSP customization

5.4.1 SDRAM configuration

The SDRAM capacity is 1 Mbyte x 32 bits x 4 banks. The BSP SDRAM driver offers a set of functions to initialize, read/write in polling or DMA mode.
The SDRAM external memory must be initialized before the GUI initialization to allow its use as LCD layers frame buffer.

Table 8. LCD frame buffer locations

<table>
<thead>
<tr>
<th>Layer</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD Layer0</td>
<td>0xC0200000</td>
</tr>
<tr>
<td>LCD Layer1</td>
<td>0xC0400000</td>
</tr>
</tbody>
</table>

The SDRAM is used also as DCMI output for camera module. The camera output is stored in camera frame buffer address as 16bpp (RGB565) and converted to 24bpp in the Camera converted frame before its stocking in the selected storage unit.

Table 9. Camera frame buffer locations

<table>
<thead>
<tr>
<th>Camera</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera frame buffer</td>
<td>0xC0000000</td>
</tr>
<tr>
<td>Camera converted frame</td>
<td>0xC0025800</td>
</tr>
</tbody>
</table>

5.4.2 Touch screen configuration

The touch screen is controlled by the BSP TS driver which uses the BSP IO driver in case of STM32429-EVAL board and STM32446E-EVAL board, and by the TS3510 component in case of STM32439-EVAL board.
Figure 31. Touch screen initialization

The touch screen is initialized in 'k_BspInit' following the used screen resolution as shown in the code below.

```c
52  **/
53  * @brief Initializes LEDs, SDRAM, touch screen, CRC and SRAM.
54  * @param None
55  * @retval None
56  */
57  void k_BspInit(void)
58  {
59      (...)
60  
61  
62  
63  
64  
65  
66  /**
67  * @brief Read the coordinate of the point touched and assign their
68  * value to the variables u82_TSPCoordinate and u82_TSPCoordinate
69  * @param None
70  * @retval None
71  */
72  void k_TouchUpdate(void)
73  {
74      GUI_PID_STATE TS_State;
75    
76  
77  
78  
79  
80  
```

4074 DocID026161 Rev 1
6 Performance

Note: This section is only available for STM329I-EVAL demonstration.

6.1 Multi buffering features

Multiple buffering is the use of more than one frame buffer, so that the display ever shows a screen which is already completely rendered, even if a drawing operation is in process. When starting the process of drawing the current content of the front buffer is copied into a back buffer. After that all drawing operations take effect only on this back buffer. After the drawing operation has been completed the back buffer becomes the front buffer. Making the back buffer the visible front buffer normally only requires the modification of the frame buffer start address register of the display controller.

Now it should be considered that a display is refreshed by the display controller approximately 60 times per second. After each period there is a vertical synchronization signal, known as VSYNC signal. The best moment to make the back buffer the new front buffer is this signal. If not considering the VSYNC signal tearing effects can occur, as shown in Figure 32.

Figure 32. Example of tearing effect

6.2 Multi layers feature

Windows can be placed in any layer or display, drawing operations can be used on any layer or display. Since there are really only smaller differences from this point of view, multiple layers and multiple displays are handled the same way (Using the same API routines) and are simply referred to as multiple layers, even if the particular embedded system uses multiple displays.

In the STM32CubeF4 demonstration, the layer 0 is dedicated for the background while the layer 1 with transparency activated is dedicated for the main desktop, this will allow to the kernel to keep the background unchanged during the desktop visual changes without refreshing the background image.
6.3 Hardware acceleration

With the STM324x9I-EVAL and Discovery Kit demonstration, the hardware acceleration capabilities of the STM32F429/STM32F439 cores are used. STemWin offers a set of customization callbacks to change the default behavior based on the hardware capabilities, the optimized processes are implemented in the LCDConf.c file and implement the following features:

a) Color conversion
   Internally STemWin works with logical colors (ABGR). To be able to translate these values into index values for the hardware and vice versa, the color conversion routines automatically use the DMA2D for that operation if the layer works with direct color mode.
   This low level implementation makes sure that in each case where multiple colors or index values need to be converted the DMA2D is used.

b) Drawing of index based bitmaps
   When drawing index based bitmaps STemWin first loads the palette of the bitmap into the DMA2Ds LUT instead of directly translating the palette into index values for the hardware. The drawing operation then is done by only one function call of the DMA2D.

c) Drawing of high color bitmaps
   If the layer works in the same mode as the high color bitmap has its pixel data available, these bitmaps can be drawn by one function call of the DMA2D. The following function is used to set up such a function:

   \[ \text{LCD\_SetDevFunc(LayerIndex, LCD\_DEVFUNC\_DRAWBMP\_16BPP, pFunc);} \]

d) Filling operations
   Setting up the function for filling operations:

   \[ \text{LCD\_SetDevFunc(LayerIndex, LCD\_DEVFUNC\_FILLRECT, pFunc);} \]

e) Copy operations
   Setting up the functions for copy operations used by the function GUI\_CopyRect():

   \[ \text{LCD\_SetDevFunc(LayerIndex, LCD\_DEVFUNC\_COPYRECT, pFunc);} \]
f) Copy buffers
Setting up the function for transferring the front- to the back buffer when using multiple buffers:

```c
LCD_SetDevFunc(LayerIndex, LCD_DEVFUNC_COPYBUFFER, pFunc);
```

g) Fading operations
Setting up the function for mixing up a background and a foreground buffer used for fading memory devices:

```c
GUI_SetFuncMixColorsBulk(pFunc);
```

h) General alpha blending
The following function replaces the function which is used internally for alpha blending operations during image drawing (PNG or true color bitmaps) or semitransparent memory devices:

```c
GUI_SetFuncAlphaBlending(pFunc);
```

i) Drawing antialiased fonts
Setting up the function for mixing single foreground and background colors used when drawing transparent ant aliased text:

```c
GUI_SetFuncMixColors(pFunc).
```
7 Footprint

The purpose of the following sections is to provide the memory requirements for all the demonstration modules, including jpeg decoder and STemWin’s main GUI components. The aim is to have an estimation of memory requirement in case of suppression or addition of a module or feature.

The footprint data are provided for the following environment:

- Tool chain: IAR 6.70.1
- Optimization: high size
- Board: STM32F429-EVAL.

7.1 Kernel footprint

*Table 10* shows the code memory, data memory and the constant memory used for each kernel file.

<table>
<thead>
<tr>
<th>File</th>
<th>code [byte]</th>
<th>data [byte]</th>
<th>const [byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_bsp</td>
<td>260</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>K_calibration</td>
<td>972</td>
<td>28</td>
<td>48</td>
</tr>
<tr>
<td>k_Log</td>
<td>100</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>k_mem</td>
<td>266</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>k_menu</td>
<td>3496</td>
<td>900</td>
<td>412089</td>
</tr>
<tr>
<td>k_module</td>
<td>214</td>
<td>244</td>
<td>0</td>
</tr>
<tr>
<td>k_module_res</td>
<td>98</td>
<td>0</td>
<td>207692</td>
</tr>
<tr>
<td>k_rtc</td>
<td>196</td>
<td>32</td>
<td>195529</td>
</tr>
<tr>
<td>k_startup</td>
<td>316</td>
<td>4</td>
<td>300064</td>
</tr>
<tr>
<td>k_storage</td>
<td>954</td>
<td>2844</td>
<td>24</td>
</tr>
<tr>
<td>main</td>
<td>614</td>
<td>4</td>
<td>44</td>
</tr>
</tbody>
</table>

1. The memory is allocated dynamically in some structures of this file.

7.2 Module footprint

*Table 11* shows the code memory, data memory and the constant memory used for each kernel file.

<table>
<thead>
<tr>
<th>File</th>
<th>code [byte]</th>
<th>data [byte]</th>
<th>const [byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>6764</td>
<td>501</td>
<td>33067</td>
</tr>
<tr>
<td>Benchmark</td>
<td>1320</td>
<td>36</td>
<td>32693</td>
</tr>
</tbody>
</table>
7.3 STemWin Features resources

7.3.1 JPEG decoder

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

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

<table>
<thead>
<tr>
<th>Resolution</th>
<th>RAM usage [kbyte]</th>
<th>RAM usage, size dependent [kbyte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>160x120</td>
<td>45.5</td>
<td>12.5</td>
</tr>
<tr>
<td>320x340</td>
<td>58.0</td>
<td>25.0</td>
</tr>
<tr>
<td>480x272</td>
<td>70.5</td>
<td>37.5</td>
</tr>
<tr>
<td>640x480</td>
<td>83.0</td>
<td>50.0</td>
</tr>
</tbody>
</table>

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

7.3.2 GUI Components

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.

Table 13 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 13. MemoSTemWin components memory requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>ROM</th>
<th>RAM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Manager</td>
<td>6.2 Kbytes</td>
<td>2.5 Kbytes</td>
<td>Additional memory requirements of basic application when using the Windows Manager</td>
</tr>
<tr>
<td>Memory Devices</td>
<td>4.7 Kbytes</td>
<td>7 Kbytes</td>
<td>Additional memory requirements of basic application when using memory devices</td>
</tr>
<tr>
<td>Antialiasing</td>
<td>4.5 Kbytes</td>
<td>2 * LCD_XSIZE</td>
<td>Additional memory requirements for the antialiasing software item</td>
</tr>
<tr>
<td>Driver</td>
<td>2-8 Kbytes</td>
<td>20 bytes</td>
<td>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</td>
</tr>
<tr>
<td>Multilayer</td>
<td>2-8 Kbytes</td>
<td>-</td>
<td>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</td>
</tr>
<tr>
<td>Core</td>
<td>5.2 Kbytes</td>
<td>80 bytes</td>
<td>Memory requirements of a typical application without using additional software items</td>
</tr>
<tr>
<td>JPEG</td>
<td>12 Kbytes</td>
<td>36 Kbytes</td>
<td>Basic routines for drawing JPEG files</td>
</tr>
<tr>
<td>GIF</td>
<td>3.3 Kbytes</td>
<td>17 Kbytes</td>
<td>Basic routines for drawing GIF files</td>
</tr>
<tr>
<td>Sprites</td>
<td>4.7 Kbytes</td>
<td>16 bytes</td>
<td>Routines for drawing sprites and cursors</td>
</tr>
<tr>
<td>Font</td>
<td>1-4 Kbytes</td>
<td>-</td>
<td>Depends on the font size to be used</td>
</tr>
</tbody>
</table>

Table 14. Widget memory requirements

| Component       | ROM   | RAM  | Description | | |
|-----------------|-------|------|-------------|---|
| BUTTON          | 1.0 Kbytes | 40 bytes | (1) |
| CHECKBOX        | 1.0 Kbytes | 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) |
| 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) |
1. The listed memory requirements of the widgets contain the basic routines required for creating and drawing the widget. Depending on the specific widget there are several additional functions available which are not listed in the table.

<table>
<thead>
<tr>
<th>Component</th>
<th>ROM</th>
<th>RAM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPAGE</td>
<td>3.9 Kbytes</td>
<td>32 bytes</td>
<td>(1)</td>
</tr>
<tr>
<td>PROGBAR</td>
<td>1.3 Kbytes</td>
<td>20 bytes</td>
<td>(1)</td>
</tr>
<tr>
<td>RADIOBUTTON</td>
<td>1.4 Kbytes</td>
<td>32 bytes</td>
<td>(1)</td>
</tr>
<tr>
<td>SCROLLBAR</td>
<td>2.0 Kbytes</td>
<td>14 bytes</td>
<td>(1)</td>
</tr>
<tr>
<td>SLIDER</td>
<td>1.3 Kbytes</td>
<td>16 bytes</td>
<td>(1)</td>
</tr>
<tr>
<td>TEXT</td>
<td>1.0 Kbytes</td>
<td>16 bytes</td>
<td>(1)</td>
</tr>
<tr>
<td>CALENDAR</td>
<td>0.6 Kbytes</td>
<td>32 bytes</td>
<td>(1)</td>
</tr>
</tbody>
</table>
8 Demonstration functional description

8.1 Kernel

The main desktop is built around two main graphical components:

- The status bar: indicates the storage units’ connection status, current time and date and a system utilities button to allow getting system information like (running task, CPU usage, and kernel log).
- The icon view widget: contains the icons associated to added modules. User can launch a module by a simple click on the module icon (see Figure 34).

![Figure 34. CPU usage display](image)

The system utilities are accessible during the STM32CubeF4 demonstration running time, using the system button (ST Logo) in top left of the main desktop. The system utilities button offers the following services:

- CPU Usage history
- Kernel log messages
- Current running processes viewer.
8.1.1 CPU Usage

The CPU Usage utility provides a graphical representation of the CPU usage evolution (Figure 35) during the demonstration run time starting for the first time it was launched. Note that once launched the CPU usage utilities keep running in background and can be restored in any time.

![Figure 35. CPU usage](image)

8.1.2 Kernel Log

The kernel log utility gather all the kernel and module messages and save them into a dedicated internal buffer. The Log messages can be visualized at any time during the demonstration run time, as shown in Figure 36.

![Figure 36. Example of Log messages](image)
8.1.3 Process Viewer

The process viewer (Figure 37) allows to check and to display the status of the currently running tasks (FreeRTOS) at any time during the demonstration run time. It shows the following information:

2. Current running tasks priorities
3. Running tasks states (FreeRTOS statics information).

![Figure 37. Process viewer](image)

8.2 Modules

8.2.1 System

Overview

The system module provides three control tabs: system information, general settings and clock settings to set the global demonstration settings. The system module retrieves demonstration information from internal kernel settings data structures and acts on the several kernel services to changes settings.

Functional description

The system module provides three graphical views:

a) Demonstration global Information (Figure 38)
   This first page shows the main demonstration information such as: Used board, STM32 core part number, and current CPU clock and demonstration revision.

b) General settings (Figure 39)
   The general settings tab permits to change the global demonstration configuration. Note that the new settings are not applied immediately; new settings take effect after restarting the demonstration.
Figure 38. Demonstration global information

Figure 39. Demonstration general settings
Table 15 shows the different settings that can be changed.

<table>
<thead>
<tr>
<th>Configuration item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable sprites</td>
<td>Checking this box allows the sprites to move on the background desktop</td>
</tr>
<tr>
<td>Enable background mode</td>
<td>Not used (reserved for future use)</td>
</tr>
<tr>
<td>Run CPU at 180 MHz</td>
<td>Allow to run the demonstration at maximum speed. Note that the device USB clock is not at compliant clock with mode. To use the USB device mass storage module, it is recommended to use the default 168 MHz CPU clock</td>
</tr>
<tr>
<td>Disable Flex skin</td>
<td>Unchecking this box, classical GUI skin is used.</td>
</tr>
</tbody>
</table>

### Table 15. Available settings

<table>
<thead>
<tr>
<th>Configuration item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable sprites</td>
<td>Checking this box allows the sprites to move on the background desktop</td>
</tr>
<tr>
<td>Enable background mode</td>
<td>Not used (reserved for future use)</td>
</tr>
<tr>
<td>Run CPU at 180 MHz</td>
<td>Allow to run the demonstration at maximum speed. Note that the device USB clock is not at compliant clock with mode. To use the USB device mass storage module, it is recommended to use the default 168 MHz CPU clock</td>
</tr>
<tr>
<td>Disable Flex skin</td>
<td>Unchecking this box, classical GUI skin is used.</td>
</tr>
</tbody>
</table>

C) Clock settings

The clock setting tab ([Figure 40](#)) allows to adjust the demonstration time and date by changing the RTC configuration of the kernel.

---

**Table 15. Available settings**

<table>
<thead>
<tr>
<th>Configuration item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable sprites</td>
<td>Checking this box allows the sprites to move on the background desktop</td>
</tr>
<tr>
<td>Enable background mode</td>
<td>Not used (reserved for future use)</td>
</tr>
<tr>
<td>Run CPU at 180 MHz</td>
<td>Allow to run the demonstration at maximum speed. Note that the device USB clock is not at compliant clock with mode. To use the USB device mass storage module, it is recommended to use the default 168 MHz CPU clock</td>
</tr>
<tr>
<td>Disable Flex skin</td>
<td>Unchecking this box, classical GUI skin is used.</td>
</tr>
</tbody>
</table>

---

**Figure 40. Clock setting**

![Figure 40. Clock setting](image-url)
8.2.2 File Browser

Overview

The File browser module is a system module that allows to explore the connected storage unit(s), to delete or to open a selected file. The file list structure is built during the media connection and updated after a connection status change of one of the used media.

![Figure 41. File browser](image)

Functional description

The file browser is mainly used for standard file operations: explore folder, file information, file deletion and opening supported extension file when a file type is linked to the direct open file feature of the kernel (Figure 42). Note that Read-Only file cannot be deleted physically from media.
To open the contextual file menu, user has to select a file (selecting a folder has no effect). The following actions are accessible through the contextual menu:

a) **Open file**: if a file extension is linked to the direct open file feature of the kernel, the associated application with this extension is launched and the file is opened automatically (*Figure 43*).

b) **Delete file**: selecting a file for deletion will display a confirmation message box to confirm the deletion operation. Note that Read-Only file cannot be deleted physically from media.

c) **Properties**: the File browser can be used to check file properties such as current location, size, and creation date.
Note: The File browser can explore up to four levels, the maximum explorer level is defined in the kernel files (k_storage.h).

8.2.3 Game

The game coming in the STM32CubeF4 demonstration is based on the Reversi game. It is a strategy board game for two players, played on an 8×8 board. The goal of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

In this STM32CubeF4 demonstration STM32 MCU is one of the two players. The GUI will ask the user to start a new game when the ongoing one is over.
8.2.4 Benchmark

Overview
The Benchmark module is a system module that allows measure the graphical performance by measuring the time needed to draw several colored rectangles in random position with random size during a specific period. The result is given in pixel per second.

Functional description
The benchmark starts immediately once the start speed benchmark button is pressed. After few seconds the result is displayed in red below the CPU Usage graphical window and result is logged in the right list box with date and time stamp (Figure 46).

Figure 46. Benchmarking

8.2.5 Audio

Overview
The audio player module provides a complete audio solution based on the STM32F4xx and delivers a high-quality music experience. It supports playing music in WAV format but may be extended to support other compressed formats such as MP3 and WMA audio formats.

Architecture
Figure 47 shows the different audio player parts and their connections and interactions with the external components.
**Data structure used**

*Table 16* contains the different data structure used in audio player module and a brief description of each of them.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAV_InfoTypedef</td>
<td>Contains the wave file information extracted from wave file header</td>
</tr>
<tr>
<td>AUDIOPLAYER_ProcessTypedef</td>
<td>Contains the audio player state, the speaker state, the volume value and the pointer to the audio buffer.</td>
</tr>
<tr>
<td>AUDIOPLAYER_StateTypedef</td>
<td>Contains the different audio player state:</td>
</tr>
<tr>
<td></td>
<td>– AUDIOPLAYER_STOP</td>
</tr>
<tr>
<td></td>
<td>– AUDIOPLAYER_START</td>
</tr>
<tr>
<td></td>
<td>– AUDIOPLAYER_PLAY</td>
</tr>
<tr>
<td></td>
<td>– AUDIOPLAYER_PAUSE</td>
</tr>
<tr>
<td></td>
<td>– AUDIOPLAYER_EOF</td>
</tr>
<tr>
<td></td>
<td>– AUDIOPLAYER_ERROR</td>
</tr>
</tbody>
</table>
Functional description

The audio player initialization is done in startup step. In this step all the audio player states, the speaker and the volume value are initialized and only when the play button in the audio player interface is pressed to start the process.

There are two ways to start audio player module:
- From main desktop menu as shown in Figure 48
- Through the file browser contextual menu: direct open feature.

When the audio player is started, the following actions are executed:
- The graphical components are initialized:
  - The audio frame
  - The control buttons
  - The list box field

An additional memory is allocated to keep the audio list (pWavList) and the audio file information (pFileInfo).
Table 17. Audio module controls

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play button</td>
<td></td>
<td>Changes the audio player state to &quot;AUDIOPLAYER_PLAY&quot;&lt;br&gt;Reads the wave file from storage unit&lt;br&gt;Sets the frequency&lt;br&gt;Starts or resumes the audio task&lt;br&gt;Starts playing audio stream from a data buffer using &quot;BSP_AUDIO_OUT_Play&quot; function in BSP audio driver.&lt;br&gt;Replaces play button by pause button</td>
</tr>
<tr>
<td>Pause button</td>
<td></td>
<td>Suspends the audio task&lt;br&gt;Pauses the audio file stream&lt;br&gt;Replaces pause button by play button</td>
</tr>
<tr>
<td>Stop button</td>
<td></td>
<td>Close the wave file from storage unit&lt;br&gt;Suspends the audio task&lt;br&gt;Stops audio playing&lt;br&gt;Changes the audio player state to &quot;AUDIOPLAYER_STOP&quot;</td>
</tr>
<tr>
<td>Previous button</td>
<td></td>
<td>Point to the previous wave file&lt;br&gt;Stops audio playing&lt;br&gt;Starts playing the previous wave file if play button is pressed</td>
</tr>
<tr>
<td>Next button</td>
<td></td>
<td>Point to the next wave file&lt;br&gt;Stops audio playing&lt;br&gt;Starts playing the next wave file if play button is pressed</td>
</tr>
<tr>
<td>Add file to playlist</td>
<td></td>
<td>Open file browser window and choose wave file to be added to playlist</td>
</tr>
<tr>
<td>Add folders</td>
<td></td>
<td>Open file browser window and choose entire folder to be added to playlist</td>
</tr>
<tr>
<td>Repeat buttons</td>
<td></td>
<td>At the end of file:&lt;br&gt;- If repeat all is selected next wave file is selected and played&lt;br&gt;- If repeat once is selected the played wave file is repeated&lt;br&gt;- If repeat off is selected the audio player stop</td>
</tr>
<tr>
<td>Speaker button</td>
<td></td>
<td>Sets the volume at mute (first press)&lt;br&gt;Sets the volume at value displayed in volume slider (second press)</td>
</tr>
<tr>
<td>Volume slider</td>
<td></td>
<td>Sets the volume value</td>
</tr>
<tr>
<td>Progress slider</td>
<td></td>
<td>Sets the desired position in the wave file</td>
</tr>
<tr>
<td>Close button</td>
<td></td>
<td>Close audio player module</td>
</tr>
</tbody>
</table>
8.2.6 Video

Overview

The video player module provides a video solution based on the STM32F4xx and STemWin movie API. It supports playing movie in emf format.

Architecture

*Figure 49* shows the different video player modules and their connections and interactions with the external components.

Functional description

There are two ways to start Video player module:

- Either by touching the video player icon: *Figure 50*
- Or by using the file browser contextual menu: direct open feature.
When the video player is started, the following actions are executed:

- The graphical components are initialized:
  - The video frame
  - The control buttons
  - The list box field
- Memory is allocated to save the video list (pVideoList) and the file information (pFileInfo).

Figure 50. Video player module startup

Table 18 summarizes the different actions behind each control button:

### Table 18. Video module controls

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
</table>
| Play button  | ![Play icon](play_icon.png) | Checks if the video size is not supported  
Supported video size: 0 < xSize < 1024 and 0 < ySize < 768  
Changes the video player state to “VIDEO_PLAY”  
Reads the video file from storage unit  
Replaces play button by pause button |
| Pause button | ![Pause icon](pause_icon.png) | Pauses the video file stream  
Changes the video player state to “VIDEO_PAUSE”  
Replaces pause button by play button |
| Stop button  | ![Stop icon](stop_icon.png) | Closes the video file from storage unit  
Stops video playing  
Changes the video player state to “VIDEO_IDLE” |
| Previous button | ![Previous icon](previous_icon.png) | Points to the previous video file  
Stops video playing  
Changes the video player state to “VIDEO_IDLE” |
To be able to play movies with the STemWin API functions it is required to create files of the STemWin specific EmWin movie file format. There are two steps to generate an emf file:

a) Convert files of any MPEG file format into a folder of single JPEG files for each frame (Figure 51). The free FFmpeg available at ffmpeg website can be used.

---

**Table 18. Video module controls (continued)**

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next button</td>
<td>![Play]</td>
<td>Points to the next video file&lt;br&gt;Stops video playing&lt;br&gt;Starts playing the next video file if play button is pressed</td>
</tr>
<tr>
<td>Add file to playlist</td>
<td>![Plus]</td>
<td>Opens file browser window and choose emf file from available storage unit to be added to playlist</td>
</tr>
<tr>
<td>Add folder</td>
<td>![Folder]</td>
<td>Opens file browser window and choose entire folder from available storage unit to be added to playlist</td>
</tr>
<tr>
<td>Repeat buttons</td>
<td>![Repeat]</td>
<td>At the end of file:&lt;br&gt;- If repeat all is selected next video file is selected and played&lt;br&gt;- If repeat once is selected the played video file is repeated&lt;br&gt;- If repeat off is selected the video player stops</td>
</tr>
<tr>
<td>Progress slider</td>
<td>![SeekBar]</td>
<td>Sets the desired position in the emf file</td>
</tr>
<tr>
<td>Full screen button</td>
<td>![FullScreen]</td>
<td>Scales the image to be showed on full screen mode</td>
</tr>
<tr>
<td>Close button</td>
<td>![Close]</td>
<td>Closes video player module</td>
</tr>
</tbody>
</table>

---

**Video file creation (emf)**

To be able to play movies with the STemWin API functions it is required to create files of the STemWin specific EmWin movie file format. There are two steps to generate an emf file:

a) Convert files of any MPEG file format into a folder of single JPEG files for each frame (Figure 51). The free FFmpeg available at ffmpeg website can be used.
b) Create an emf file from JPEG file using JPEG2Movie tool available in STemWin package (see Figure 52).

Figure 52. JPEG2Movie overview
The above steps could be done once using a predefined batch (included in the STemWin package) as shown in Figure 53.

**Figure 53. EMF file generation**

For more information about how to use the emf generation batches, refer to the STemWin User and Reference Guide (UM3001).

<table>
<thead>
<tr>
<th>File</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.bat</td>
<td>Sets some defaults to be used. Needs to be adapted as explained in Prep.bat.</td>
</tr>
<tr>
<td>MakeMovie.bat</td>
<td>Main conversion file. Not to be adapted normally.</td>
</tr>
<tr>
<td>&lt;X_SIZE&gt;x&lt;Y_SIZE&gt;.bat</td>
<td>Some helper files for different resolutions. Detailed explanation in &lt;X_SIZE&gt;x&lt;Y_SIZE&gt;.bat</td>
</tr>
</tbody>
</table>
Prep.bat

The Prep.bat is required to prepare the environment for the actual process. Calling it directly will not have any effect. It is called by the MakeMovie.bat. To be able to use the batch files it is required to adapt this file at first. This file sets variables used by the file MakeMovie.bat, they are listed in Table 20.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%OUTPUT%</td>
<td>Destination folder for the JPEG files. Will be cleared automatically when starting the conversion with MakeMovie.bat.</td>
</tr>
<tr>
<td>%FFMPEG%</td>
<td>Access variable for the FFmpeg tool. Should contain the complete path required to call FFmpeg.exe.</td>
</tr>
<tr>
<td>%JPEG2MOVIE%</td>
<td>Access variable for the JPEG2MOVIE tool. Should contain the complete path required to call JPEG2Movie.exe.</td>
</tr>
<tr>
<td>%DEFAULT_SIZE%</td>
<td>Default movie resolution to be used. Can be ignored if one of the &lt;X-SIZE&gt;x&lt;Y-SIZE&gt;.bat files are used.</td>
</tr>
<tr>
<td>%DEFAULT_QUALITY%</td>
<td>Default quality to be used by FFmpeg.exe for creating the JPEG files. The lower the number the better the quality. Value 1 indicates that a very good quality should be achieved, value 31 indicates the worst quality. For more details please refer to the FFmpeg documentation.</td>
</tr>
<tr>
<td>%DEFAULT_FRAMERATE%</td>
<td>Frame rate in frames/second to be used by FFmpeg. It defines the number of JPEG files to be generated by FFmpeg.exe for each second of the movie. For more details please refer to the FFmpeg documentation.</td>
</tr>
</tbody>
</table>

MakeMovie.bat

This is the main batch file used for the conversion process. Normally it is not required to be change this file, but it is required to adapt Prep.bat first. It could be called with the parameters listed in Table 21:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%1</td>
<td>Movie file to be converted</td>
</tr>
<tr>
<td>%2 (optional)</td>
<td>Size to be used. If not given %DEFAULT_SIZE% of Prep.bat is used.</td>
</tr>
<tr>
<td>%3 (optional)</td>
<td>Quality to be used. If not given %DEFAULT_QUALITY% of Prep.bat is used.</td>
</tr>
<tr>
<td>%4 (optional)</td>
<td>Frame rate to be used. If not given %DEFAULT_FRAMERATE% of Prep.bat is used.</td>
</tr>
</tbody>
</table>
Since the FFmpeg output can differ strongly from the output of previous actions, the MakeMovie.bat deletes all output files in the first place. The output folder is defined by in the environmental variable %OUTPUT% in Prep.bat. After that it uses FFmpeg.exe to create the required JPEG files for each frame. Afterwards it calls JPEG2Movie to create a single EMF file which can be used by STemWin directly. After the conversion operation the result can be found in the conversion folder under FFmpeg.emf. It also creates a copy of that file into the source file folder. It will have the same name as the source file with a size-postfix and .emf extension.

<\x_SIZE>x<\y_SIZE>.bat

These files are small but useful helpers if several movie resolutions are required. The filenames of the batch files itself are used as parameter '-s' for FFmpeg.exe. You can simply drag-and-drop the file to be converted to one of these helper files. After that an .emf file with the corresponding size-fix can be found in the source file folder.

8.2.7 USB Mass storage Device

Overview

The USB device module includes mass storage device application using the MicroSD memory. It uses the USB OTG FS peripheral as the USB OTG HS is used for the USB disk Flash storage unit.

Architecture

Figure 54 shows the different USBD module components and their connections and interactions with the external components.

Figure 54. USBD module architecture
Data structure used

Table 22. Data structure for USBD module

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBDSettingsTypeDef</td>
<td>sd_mounted: connection status</td>
</tr>
</tbody>
</table>

Functional description

Run USB Device demonstration by clicking USB device icon in the main desktop, as in Figure 55.

**Figure 55. USBD module startup**

Table 23. USBD module controls

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect USB</td>
<td><img src="image" alt="Connect USB Preview" /></td>
<td>Changes the USB logo as follows: Changes the USBD status as CONNECTED</td>
</tr>
<tr>
<td>Disconnect USB</td>
<td><img src="image" alt="Disconnect USB Preview" /></td>
<td>Changes the USB logo as follows: Changes the USBD status as DISCONNECTED</td>
</tr>
</tbody>
</table>
8.2.8 Camera

Overview

The camera application allows to directly and permanently display on the LCD the image captured using the camera module. It is also possible to take a snapshot and save it to a customizable location in the storage unit.

In addition to brightness and contrast which are adjustable, several effects can be applied to the output image: black and white, negative, antique...etc. Note that all these effects can be applied in runtime.

Architecture

*Figure 56* shows the different camera module parts and their respective connections and interactions with the external components.

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert microSD card</td>
<td>NA</td>
<td>Changes the microSD logo as follows:</td>
</tr>
<tr>
<td>Remove microSD card</td>
<td>NA</td>
<td>Changes the microSD logo as follows:</td>
</tr>
<tr>
<td>Close</td>
<td>✗</td>
<td>Closes USBD module</td>
</tr>
</tbody>
</table>

Table 23. USBD module controls (continued)
Functional description

To start the camera module click on the Camera icon, as indicated in Figure 57.

When the camera module is started, the following actions are executed:

- The graphical components are initialized.
- Memory is allocated to save the capture folder location (pFileInfo).
- The saved parameters (brightness and contrast) are restored from the RTC backup register.

*Table 24* summarizes the different actions behind each control button:
8.2.9 Image viewer

Overview

The Image viewer module allows displaying bmp and jpg pictures. It is possible to load the full images list from a folder or to add the images manually to the playlist. Once the playlist is created, navigation between pictures can be done either via Next and previous buttons or by enabling the slide show mode. The slide show timer can be changed on the fly (there is no need to restart the module).

Architecture

*Figure 58* shows the different image viewer parts and their respective connections and interactions with the external components.
Functional description

There are two ways to start Image viewer module:

- Either by touching the Image viewer icon (Figure 59);
- Or by using the file browser contextual menu: direct open feature.

When the image viewer is started, the following actions are executed:

- The graphical components are initialized:
  - The image frame
  - The control buttons
  - The list box field
- Memory is allocated to save the image list (pImageList) and the file information (pFileInfo).
- The saved parameters are restored from the RTC backup register.
Table 25 summarizes the different actions behind each control button.

<table>
<thead>
<tr>
<th>Button</th>
<th>Preview</th>
<th>Brief description</th>
</tr>
</thead>
</table>
| Close              | ![Close icon] | Frees allocated memory  
                                Ends the module dialog                                           |
| Previous           | ![Previous icon] | Closes the current image  
                                Opens the previous image  
                                Refreshes the image frame  
                                Updates the selection in the playlist                           |
| Start slideshow    | ![Start slideshow icon] | Closes the current image  
                                Opens the next image  
                                Refreshes the image frame  
                                Creates the slideshow timer                                     |
| Next               | ![Next icon] | Closes the current image  
                                Opens the next image  
                                Refreshes the image frame  
                                Updates the selection in the playlist                           |
| Settings           | ![Settings icon] | Creates and shows the settings dialog                                              |
| Add folder         | ![Add folder icon] | Opens the directory chooser to allow selection of an entire folder and then adds all the images included in this folder to the playlist |
| Add file           | ![Add file icon] | Opens the file chooser to allow selection of an image which will be added to the playlist |
### 9 Revision history

#### Table 26. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-Apr-2014</td>
<td>1</td>
<td>Initial release. Introduced STM32446E-EVAL demonstration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 5: Startup window for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 7: Main desktop window for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 9: Status bar for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 11: Icon view widget for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 14: Starting file execution for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 18: Detection of storage units for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 21: Memory heap for STM32CubeE4 demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- footnote 1 in Section 4.5: Module’s direct open;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- note in Section 6: Performance.</td>
</tr>
<tr>
<td>20-Mar-2015</td>
<td>2</td>
<td>Updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Introduction and figure on Cover page;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 3.1: Overview;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 3.2: Kernel initialization;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 3.3: Kernel processes and tasks;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 3.7: Direct open feature;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 5: Startup window for STM32446E-EVAL demonstration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 23: STM32Cube demonstration boards;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 3.11: Memory Management;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 3.15: Hardware settings;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Table 7: Jumpers for different demonstration boards;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Section 5.4.2: Touch screen configuration;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Figure 53: EMF file generation.</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved