

H8S/H8SX Series
GAPI Graphics API Version 1.0
User's Manual

Direct Drive Solution

User's Manual

Rev. 1.00
Dec 16, 2008

Renesas Technology America
america.renesas.com

Index

1. INTRODUCTION.....	2
1.1 GAPI OVERVIEW	2
1.1.1 <i>Philosophy</i>	2
1.1.2 <i>Capabilities</i>	2
1.2 BMP CREATION	3
1.2.1 <i>BMP Format</i>	3
1.2.2 <i>Color Information</i>	3
1.2.3 <i>RLE (Run Length Encoding) Compression</i>	4
1.2.4 <i>BMP Transparency</i>	5
2. SOFTWARE ARCHITECTURE.....	6
2.1 USAGE IN DEMO FRAMEWORK.....	6
2.1.1 <i>Memory Usage</i>	6
2.2 EVENT HANDLER	7
2.2.1 <i>Screen_type Structure</i>	7
2.2.2 <i>Screen Management</i>	8
2.2.3 <i>Screen Entry/Exit</i>	8
2.2.4 <i>Screen Tasks</i>	8
2.3 EXTERNAL BUS MANAGEMENT.....	10
2.3.1 <i>Standard Redefines</i>	11
2.3.2 <i>GAPI Data Types</i>	11
3. GAPI DEFINITION.....	12
3.1 BMP ACCESS AND CREATION.....	12
3.1.1 <i>LCDBMPInitFrame</i>	12
3.1.2 <i>LCDBMP24_16</i>	13
3.1.3 <i>BMP_Height</i>	14
3.1.4 <i>BMP_Width</i>	15
3.1.5 <i>BMP_Offset</i>	16
3.1.6 <i>BMP_FileSize</i>	17
3.1.7 <i>BMP_IndexColors</i>	18
3.2 BMP COPY ROUTINES.....	19
3.2.1 <i>LCDBMPCopy</i>	20
3.2.2 <i>LCDBMPColorCopy</i>	21
3.2.3 <i>LCDBMPIndex</i>	22
3.2.4 <i>LCDBMPScroll</i>	23
3.2.5 <i>LCDBMPCopySub</i>	24
3.3 FONT DRAW ROUTINES	26
3.3.1 <i>LCDBMPGPutS</i>	26
3.3.2 <i>LCDBMP_FontWidth</i>	28
3.4 FILL ROUTINES.....	29
3.4.1 <i>LCDBMPFill</i>	29
3.4.2 <i>LCDBMPFillGradient</i>	30
3.5 OTHER ROUTINES	31
3.5.1 <i>LCDBMPCalcGradientCT</i>	31
3.5.2 <i>LCDBMPCalcShadeCT</i>	32
3.5.3 <i>LCDBMPLabel</i>	33
3.5.4 <i>LCDBMPButton</i>	34

1. Introduction

1.1 GAPI Overview

The GAPI Graphics API is a set of routines to allow for the simple creation and manipulation of raster based images in RAM memory frames. In turn, these memory frames can be treated as input into the GAPI routines (allowing for the creation of complex composite images). Once the desired image has been created, the frame can be used as the display buffer for an LCD panel; this is the case in our Renesas LCD Direct Drive demonstration code.

1.1.1 Philosophy

The reason GAPI was developed was to provide a set of tools that could be used to illustrate the use of the Renesas Direct Drive solution capabilities. This solution is freely available to users as part of the Renesas Direct Drive Solution, and includes access to the source code for use on Renesas MCUs. While the solution is not as powerful as some of our 3rd party partners, it is usable for many types of user interfaces and as an evaluation platform during the MCU decision making process. Through re-use of source images and building of composite images at run-time, the GAPI is a very efficient tool for creating solutions that fit completely within the on chip flash memory resources of the MCU.

1.1.2 Capabilities

The current capabilities of the GAPI include:

- Ability to use most standard BMP formats as input.
- Ability to create and place text strings arbitrarily into a frame.
- Ability to fill regions of a frame with solid or gradient colors.
- Ability to transform colors of BMP inputs to suit application needs at runtime.
- Create 16bpp output frame for LCD display use.

Utilizing these relatively few functions allows for the creation of efficient, attractive user interfaces with relatively little coding overhead.

1.2 BMP Creation

Because the GAPI accepts most standard BMP formats, the creation of graphic content is a simple matter using widely available tools such as MS Paint, GIMP and PhotoShop. This section covers a few key points about BMP formats that allow for their optimal use with GAPI.

1.2.1 BMP Format

The file format of BMP files consists of two pieces; a header describing the image, and a raster containing the image. The header contains information on the height, width, color format of the raster, and optionally an index table to decode the raster color data.



1.2.2 Color Information

The BMP format allows for tradeoffs in raster size .vs color depth. The raster can be stored as 1, 4, 8, 16, 24 or 32 BPP (bits per pixel). Depending on the software tool, there are different methods to generate the various BMP formats, but often it will be found under a “save as” menu.

BPP	Number of Colors	Pixel Format	Size of 320x240 Image
1 bpp	2 (indexed)	8:8:8	9.6 Kbytes
4 bpp	16 (indexed)	8:8:8	38.4 Kbytes
8 bpp	256 (indexed)	8:8:8	76.8 Kbytes
16 bpp	64K	5:6:5	153.6 Kbytes
24 bpp	24M	8:8:8	230.4 Kbytes
32 bpp	24M (only 24 bit used for color)	8:8:8	307.2 Kbytes

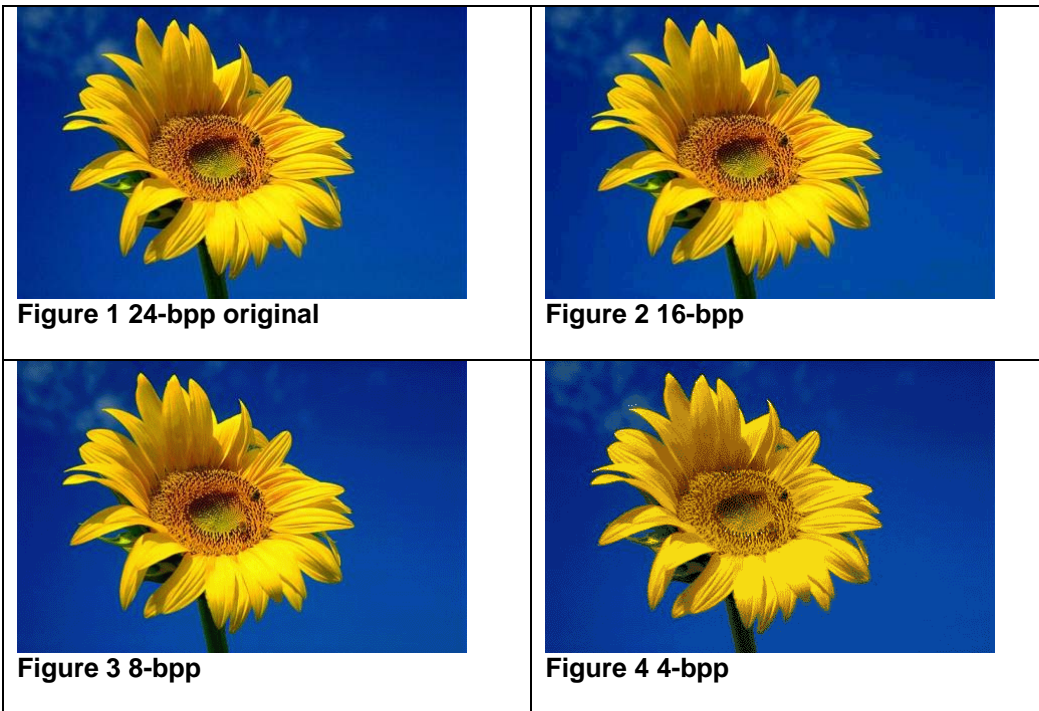
A few points from this data.

- The indexed formats do not contain the color data in the raster, they contain an index into a color lookup table (contained in the BMP header). So while these formats have a limited number of colors available, those colors can be any color within a 24-bit 8:8:8 (red:green:blue) color space.

For example in 1 BPP BMP, the two color choices could be  (173:44:201) and  (36:132:206).

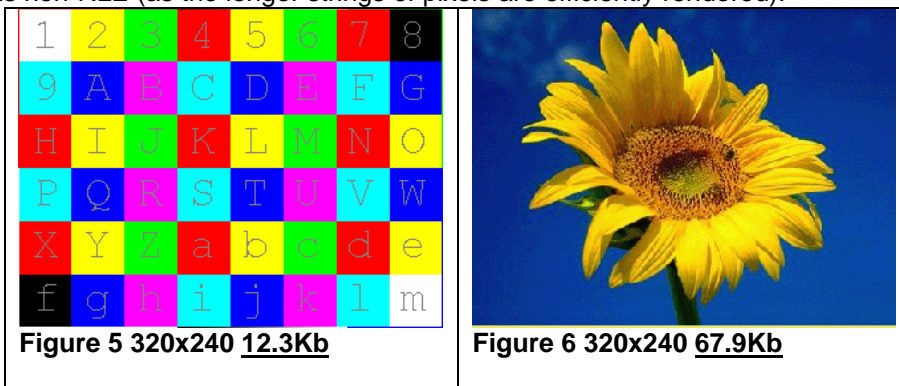
- As our Direct Drive solutions today only support 16bpp panels, all output data is converted to 16bpp for display. There is size and conversion overhead for associated with these formats that do not justify their use. The additional data in 24bpp and 32bpp BMP formats are just discarded, so there is no reason to use this formats with GAPI.
- There is overhead in the lookup process for indexed formats, however this process is optimized and often this overhead is justified for the size savings. Additionally, because the index tables can be manipulated at runtime, it makes the changing of the coloring of images in this format at runtime practical.
- Small images that are not going to have their colors changed are most efficiently saved in 16bpp formats. This avoids the size overhead of the index table, as well as the table lookup overhead.
- In general, most images do not need to be saved at 16bpp and indexed forms yield accurate coloring. When creating large background images, it is advised to create content that looks “good” in one of the indexed formats to conserve space. Alternatively, using solid or gradient backgrounds consume no space as they are generated at runtime.

The following illustrates the differences in some of the different formats. Often where you will see issues is in subtle transitions in shading. In non-photo images, often there is little to no appreciable loss of quality.




1.2.3 RLE (Run Length Encoding) Compression

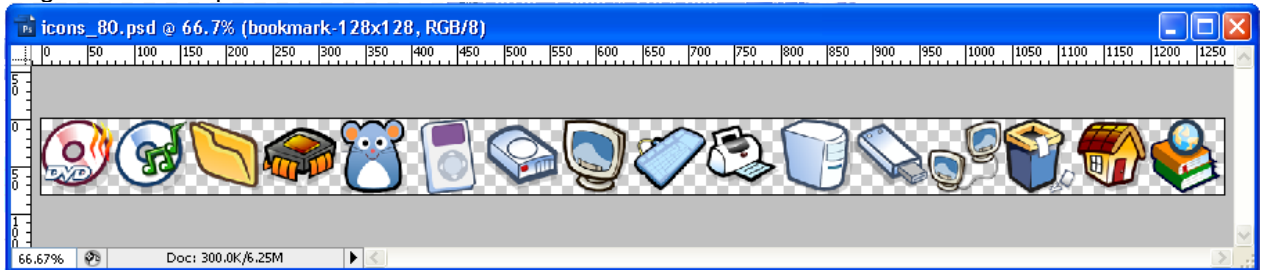
GAPI supports RLE compression on 8bpp index BMPs. RLE is a simple compression mechanism that depends on having “strings” of pixels with the same value. Because of this, the amount of compression is highly dependent on the source content. Because of this ensure that amount of compression justifies the overhead of decoding the RLE. For images where there is only a 10% space savings (as in Figure 6), you may incur a 50% increase in the time to render the image compared to non-RLE; however where the image has high compression (as in Figure 5), the amount if time to render the image can actually be the same or less as non-RLE (as the longer strings of pixels are efficiently rendered).



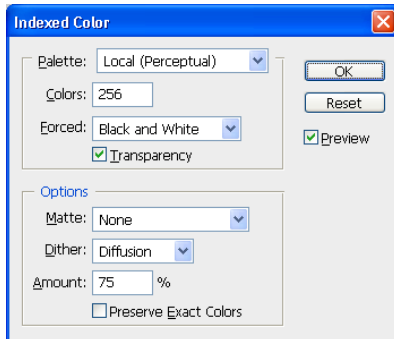
1.2.4 BMP Transparency

BMP files do not inherently support transparency, but with proper file construction, the GAPI tool can (optionally) interpret a color as being “transparent”. There are two ways to accomplish this.

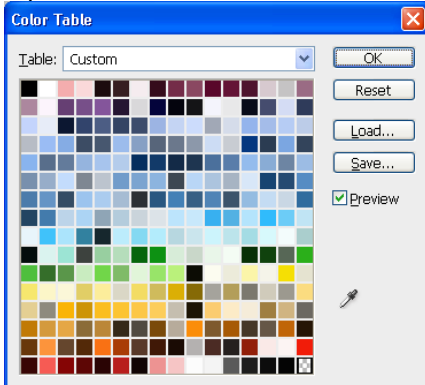
- With 16bpp source images, when calling the “copy” API routines, you can specify a designated 16bpp pixel value to be treated as transparent. During the copy process, this “transparent” color is not copied to the destination raster. On the source BMP fill any “transparent” areas with this color you are designating. It is often easiest to always use the same color and keep it in tool palette. Here the green color is used to specify the transparent color and the value is passed as the transparent argument to the copy routines. 
- With indexed BMP formats and using some tools (notably Photoshop). You can save transparent areas of the source image into the BMP by designating a special index (last one used) as part of the index table. When transparency is specified during the creation of the index table, this index value is automatically used for any transparent areas. When calling the BMP copy routines, specifying the last index value of the table as transparent will prevent those pixels from being copied to the destination BMP.
- Original Photoshop Document.



- Convert to indexed mode.



- Resultant color table with transparency. Save this file as an 8Bpp BMP format. Note that when opened with other programs or even Photoshop after the save, the “transparency” color will be represented as white. But the “transparent” area will still correspond to this index value.

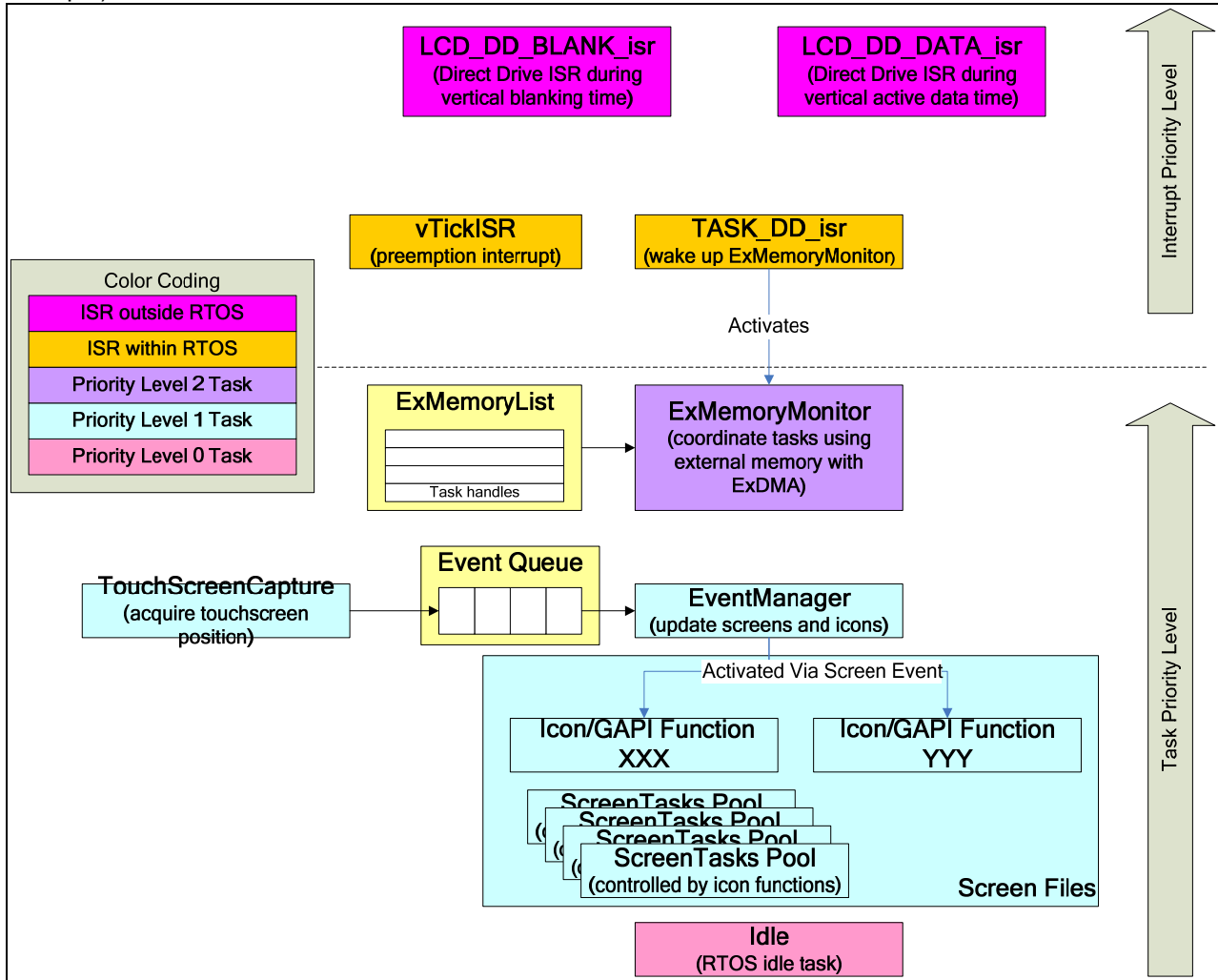


2. Software Architecture

This section will show a practical example of how the GAPI is integrated into an overall system.

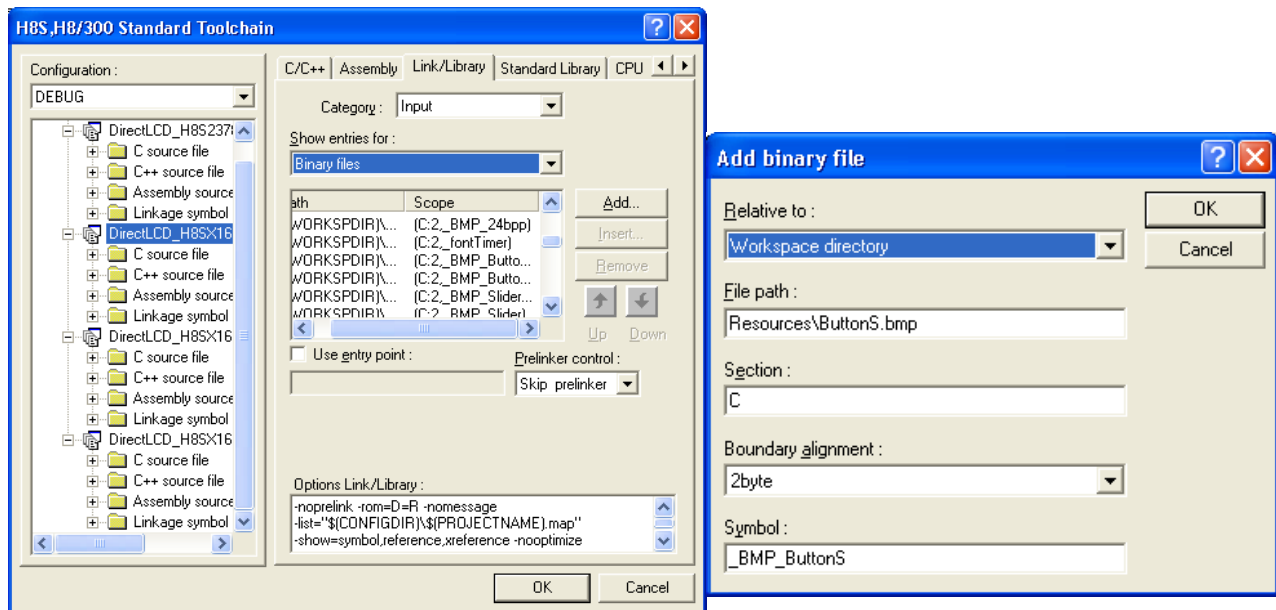
2.1 Usage in demo framework.

The Renesas LCD Direct Drive demonstration software makes use of the GAPI to provide on screen graphics support. In some areas the demo framework is wrapping the GAPI in higher level abstractions to provide touchscreen button/widget support. In other areas it is directly accessed (to paint a background for example).



2.1.1 Memory Usage

There are many ways to store the source images for access by the GAPI at runtime (SD card, external flash, internal flash, etc). In the Direct Drive demo we are storing them in internal flash by linking the standard BMP files at compile time with the rest of the firmware. This is accomplished through the HEW linker dialog. To add a file, you select the "link/library" tab, "input" category, show entries for "binary files" and "Add". This dialog allows you to specify the source file, the section to link the file into, and a symbolic label to allow access to the file from the source code. For the dialog shown, this generates a linker command of "-binary=\$(WORKSPDIR)\Resources\ButtonS.bmp"(C:2,_BMP_ButtonS)", more information on this linker command can be found in the "H8S, H8/300 Series C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual".



At runtime, these BMP files are accessed by the GAPI under user program control and copied to external RAM to provide a user interface.

2.2 Event Handler

The demo code provides that an “event handler” that processes generated events (notably touchscreen interaction) and routes them to a “ScreenUpdate” handler. At any given time, one “Screen” is in scope of the event handler, and all events are passed the screen for handling. These event and screen handlers are not part of GAPI, but utilize GAPI to render the user interface.

The screens in the demo are instantiated each in a single “.c” file named “ScreenXXX.c”. The interface between the event handler and the screen is in the form of a “SCREEN_type” data structure, each screen file contains one of these data structures. The intent is to allow screens to be independent objects that can easily be inserted, moved or removed from the user interface.

The “root” screen that needs to be present in the system is the “ScreenHomeData” structure. The event handler initializes the system to use this structure for the first screen. Subsequently, messages can be passed to the event handler to switch to other Screen_type structures.

2.2.1 Screen_type Structure

The Screen_type structure consists of Constructor and Destructor function pointers (see Screen Entry/Exit) as well as a pointer to a list of Icon_type objects. This list of icons is used by event handler to allow the screen to process external events.

These Icon_type includes pointer to a handler function to process the event. There are other records in this object type (BMP pointer, Scheme pointer, X/Y positions), but these records are not used by the event handler. The event handler calls each icon handler within the screen with the event information as well as a pointer its own Icon_type structure. This additional information (BMP pointer, Scheme pointer, X/Y positions), can than be used icon handler to display and process the button (or whatever object type the icon handler is configured for).

The following is an example of screen structure and associated icon list. The third field is the icon function pointer; the other fields are available for use by the icon function pointer. By maintaining this information in the list, it makes modifying button BMP, coloring and position very convenient.

```

/* MUST INCLUDE NULL TERMINATED RECORD */
static const ICON_type Icons[]=
{
    { &BMP_ButtonR, T_SchemeBlue,   ScreenCountdown,   SX(0.063), SY(0.783) },
    { &BMP_ButtonR, T_SchemeGreen,  ScreenAnimate,     SX(0.375), SY(0.783) },
    { &BMP_ButtonR, T_SchemeRed,    ScreenData,        SX(0.688), SY(0.783) },
    { &BMP_ButtonR, T_SchemeRed,    ScreenSlider,      SX(0.063), SY(0.566) },
    { &BMP_ButtonR, T_SchemeBlue,   ScreenWrites,      SX(0.375), SY(0.566) },
    { &BMP_ButtonR, T_SchemeGreen,  ScreenIO,          SX(0.688), SY(0.566) },
    { NULL,         NULL,           NULL,              0, 0 },
};

```



```

SCREEN_type ScreenHomeData=
{
    Constructor, Destructor, Icons
};

```

The framework provides several generic handler routines that allow for very simple (but limited) behavior by the icon handler. The following icon handler uses a generic “ButtonHandler” to process much of this handlers functionality. This icon handler will draw a button on the screen (colored with centered text) at “Constructor” time, highlight the button when pressed, and when the button is released while active, this function will then pass a message to the event handler to switch to a new screen.

```

void ScreenAnimate(ICON_type const *pS, EVENT_MSG const *pMsg)
{
    static sI16 state=0;
    if(0 != (state = ButtonHandler(pS, pMsg, (uI08 *)"Animate", state)))
        if (MSG_RELEASE == pMsg->id)
        {
            // send a screen change command to the handler
            msg.type = DISPLAY_TYPE;
            msg.id = MSG_SCREEN_PUSH;
            msg.param.pS = &ScreenAnimateData;
            RLCD_QueueSend(retVal,EventQueue, msg)
        }
}

```

Alternatively, the icon handler can choose to process the event without using any generic handlers involved to completely customize the behavior.

2.2.2 Screen Management

Messages can be generated within the firmware to request changes to which screen is active. The screen changes can be tracked to allow nesting into “screens” and then backing out. This done through a stack structure with maximum depth of 8 levels deep.

Message Id	Description
MSG_SCREEN_CHANGE	Change screen without maintaining information about how to return to the previous screen.
MSG_SCREEN_PUSH	Change screen while maintaining information about how to return to the previous screen.
MSG_SCREEN_POP	Return to a previously saved screen.

2.2.3 Screen Entry/Exit

To provide housekeeping capabilities, each screen structure defines a “constructor” and “destructor” function that is called when the screen is entered and exited respectively.

- The constructor provides an opportunity to draw the screen background and buttons and initialize and screen specific variables. A “BasicConstructor” function is provided as a convenience that can be used to draw all of the buttons enumerated in the screen structure.
- The destructor function is called by the event handler prior to switching to new screen to allow an opportunity for screen cleanup. A “BasicDestructor” function is provided that shuts down all screen tasks.

2.2.4 Screen Tasks

The event manager only calls the screen handlers when an event occurs; the question then is how to handle periodic updates without blocking the event handler?

To minimize dynamic memory allocation (and possible fragmentation), the demo does not dynamically generate and kill tasks. However, it is often desirable to have temporary tasks running to handle aspects of a screen (lengthy operations, animations, count downs, etc). Rather than having each screen permanently allocate a task for these temporary operations, the demo include a pool of pre-allocated tasks that are available for temporary screen use.

The following functions can be used to access these screen tasks. Usage examples can be found in the demo code.

Function Call	Description
ScreenTaskStart	Requests the use of one of the pool of task. Takes a function pointer as an

	argument and returns 0 on successful allocation of a task.
ScreenTaskStop	Releases the use of the task for others. Takes a function pointer as an argument and returns 0 on successful allocation of a task.
ScreenTasksStop	Releases all screen tasks back to the pool. Useful on exit of a screen and is part of the BasicDestructor call.

The following is an example of usage of the task pool.

This event driven "button" function is starting and stopping the "AnimateDemo" task.

```
static void AnimateStart(ICON_type const *pS, EVENT_MSG const *pMsg)
{
    static sI16 state=0;
    static uI08 running=0;
    if(0 != (state =IconHandler(pS, pMsg, 0, state)))
        if (MSG_RELEASE == pMsg->id)
            {
                if(0 == running)
                    {
                        // start timer Scrolling Demo
                        (void)ScreenTaskStart(AnimateDemo);
                        running=1;
                    }
                else
                    {
                        // start timer Scrolling Demo
                        (void)ScreenTaskStop(AnimateDemo);
                        running=0;
                    }
            }
}
```

This task is now executing every 50mS to update graphics on the screen independent of other events being processed.

Note that this task function does exit after each time it is called and that the return value is the timeout being used until the next call.

```
static TickType AnimateDemo(void)
{
    static sI16 index = 0;
    TickType xDelay = (TickType)(50/TICK_RATE_MS);
    ICON_type *pI = &ScreenAnimateData.icons[ANIMATE_ICON];

    if (++index > (BMP_Width(pI->pBmp)/BMP_Height(pI->pBmp)))
        index = 0;

    /* Let system know we're accessing External Memory */
    ExMemoryAcquire(RLCD_GetTaskHandle());
    {
        (void) LCDBMPIndex(pI->pBmp,displayFrameBuf,pI->PosX,pI->PosY, index);
    }
    ExMemoryRelease(RLCD_GetTaskHandle());

    return (xDelay);
}
```

2.3 External Bus Management

The external bus is under control of the External DMA unit for updating images to the LCD panel. During LCD update, the external bus is not available for the MCU core to access it. To provide this coordination (allowing external access to the bus during vertical blanking time), the demo software provides a mechanism for tasks to register and un-register their use of the external bus.

The consequences of accessing the bus with registering usage are screen artifacts and stalling of the MCU core as it awaits the ExDMA block transfer to complete.

Note that tasks that are not registered as currently using the external bus will continue to execute during the ExDMA transfer.

Function Call	Description
ExMemoryAcquire	Informs the system that this task is currently using the external bus. Accepts a task handle as an argument.
ExMemoryRelease	Informs the system that this task no longer is using the external bus. Accepts a task handle as an argument.

2.3.1 Standard Redefines

These following type have been redefined in order to make the code easier for formatting.

```
typedef unsigned char  uI08;    // Unsigned Integer 8-bits
typedef signed char    sI08;    // Signed Integer 8-bits
typedef unsigned short uI16;    // Unsigned Integer 16-bits
typedef signed short   sI16;    // Signed Integer 16-bits
typedef unsigned long  uI32;    // Unsigned Integer 32-bits
typedef signed long    sI32;    // Signed Integer 32-bits
```

2.3.2 GAPI Data Types

These data types are used within the API to support graphics data structures.

```
/* Data structure of BMP color table entry */
typedef struct ColorTable_type;

/* BMP header structure */
typedef struct BMP_type;

/* 16bpp data structure for 5:6:5 (this may vary by MCU endian/compiler) */
typedef uI16 PIXEL_16bpp_type;

/* 24bpp data structure for 8:8:8 formatted pixels */
typedef struct P24bpp_type;

/* union of pixel types that can be within raster */
typedef union pRaster_type;

/* structure definition for font information */
typedef struct GPUT_type;

/* structure definition for higher level color/font schemes */
typedef struct GScheme_type;
```

3. GAPI Definition

3.1 BMP Access and Creation

Internally, the GAPI wraps all raster images in a BMP wrapper to maintain information about the content of the raster data. This wrapper is in the form of an industry standard BMP format. The following functions return information about a given BMP or provide for initialization of an internally allocated BMP wrapper for a raster.

All BMP coordinate information follows the Window's standard (0,0 is in the lower left corner)

3.1.1 LCDInitFrame

Initialize an allocated BMP_type structure.

Format

```
gapiResp_type LCDInitFrame
(BMP_type *const d, const uI08 bpp, const sI16 Width, const sI16 Height);
```

Parameters

d

Pointer to the user allocated structure to be initialized.

bpp

Bits per pixel of raster. Currently only accepted value for internal BMPs is **16**.

Width

Width of the raster associated with this raster.

Height

Height of the raster associated with this raster.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_bmp.c"

Description

This function is called to initialize a user allocated BMP structure for a given raster to allow its use internally to other GAPI functions. The allocated raster must immediately follow the header in memory.

Example

```
BMP_type wrapper;
uI16 raster[160L*120L]

{
    LCDInitFrame(&wrapper, 16, 160, 120);
}
```

3.1.2 LCDBMP24_16

Conversion routine from an 8:8:8 pixel data to a 5:6:5 pixel data.

Format

```
uI16 LCDBMP24_16( ColorTable_type const *const p24 );
```

Parameters

p24

Pointer to the 8:8:8 pixel data.

Return Values

5:6:5 pixel data.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi.h"

Description

Converts 24bpp RGB data to 16bpp RGB data. Note that HW_BYTE_SWAPPED_BUS can be defined to configure the code to compensate for the connection of the MCU to the LCD panel. In the case of the big endian H8S family connected R5:G6:B5 to D15:D0 **do not** define HW_BYTE_SWAPPED_BUS, in the case of connection as G3:B5:R5:G3 to D15:D0 **do** define HW_BYTE_SWAPPED_BUS.

Example

```
pRaster_type psr;  
  
{  
    uI16 pixel= LCDBMP24_16( psr.pCT );  
}
```

3.1.3 BMP_Height

Extract height information from BMP_type structure.

Format

```
sI16 BMP_Height(BMP_type const *const pBmp);
```

Parameters

pBmp

Pointer to the BMP structure.

Return Values

Height of the raster as defined in the BMP structure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_bmp.c"

Description

Request the height field of the BMP structure. While the BMP format allocates 4 bytes to this field, this API uses at most a 16 bit size.

Example

```
BMP_type wrapper;  
  
{  
    sI16 height=BMP_Height(&wrapper);  
}
```

3.1.4 BMP_Width

Extract width information from BMP_type structure.

Format

```
sI16 BMP_Width(BMP_type const *const pBmp);
```

Parameters

pBmp

Pointer to the BMP structure.

Return Values

Width of the raster as defined in the BMP structure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_bmp.c"

Description

Request the width field of the BMP structure. While the BMP format allocates 4 bytes to this field, this API uses at most a 16 bit size.

Example

```
BMP_type wrapper;  
  
{  
    sI16 width=BMP_Width(&wrapper);  
}
```

3.1.5 BMP_Offset

Extract offset to raster information from BMP_type structure.

Format

```
sI32 BMP_Offset(BMP_type const *const pBmp);
```

Parameters

pBmp

Pointer to the BMP structure.

Return Values

Offset of the raster as defined in the BMP structure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_bmp.c"

Description

Request the offset to raster field of the BMP structure. Normally the raster immediately follows the variable length BMP wrapper information. This offset is the distance (in bytes) to the raster from the start of the BMP structure

Example

```
BMP_type wrapper;  
  
{  
    pRaster_type pRaster.p08 = (uI08 *)&wrapper + BMP_Width(&wrapper);  
}
```

3.1.6 **BMP_FileSize**

Extract size of BMP file (wrapper + raster) information from BMP_type structure.

Format

```
uI32 BMP_FileSize (BMP_type const *const pBmp);
```

Parameters

pBmp

Pointer to the BMP structure.

Return Values

File size of BMP.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_bmp.c"

Description

Returns total size of the BMP file (wrapper + raster)

Example

```
extern BMP_type *picture;

{
    uI32 size = BMP_Width(picture);
}
```

3.1.7 **BMP_IndexColors**

Determine number of significant colors in the BMP color table.

Format

```
uI32 BMP_IndexColors (BMP_type const *const pBmp);
```

Parameters

pBmp

Pointer to the BMP structure.

Return Values

Number of significant colors in the index table, 0 if not an indexed format.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_bmp.c"

Description

BMP index tables can have a variable amount of defined colors (up to 256). This function returns that number active in this BMP.

Example

```
extern BMP_type *picture;  
  
{  
    uI32 colors = BMP_IndexColors (picture);  
}
```

3.2 BMP Copy Routines

Several routines are provided to manipulate and copy rasters as defined in this section. Unless noted otherwise, these routines are capable accepting the following raster formats. Currently the output (destination format) for all of these routines is 16bpp 5:6:5 format.

Raster Type (bits per pixel)	Max Index Table Entries	Pixel Format	Notes
1 bpp	2	8:8:8	
4 bpp	16	8:8:8	
8 bpp	256	8:8:8	
8 bpp RLE	256	8:8:8	Run Length Encoded
16 bpp	0	5:6:5	
24 bpp	0	8:8:8	
32 bpp	0	8:8:8	

3.2.1 LCDBMPCopy

Copies source BMP to destination BMP.

Format

```
gapiResp_type LCDBMPCopy
( BMP_type const *const s, BMP_type *d, sI16 PosX, sI16 PosY );
```

Parameters

s

Pointer to the source BMP structure.

d

Pointer to the destination BMP structure.

PosX

X axis offset within destination raster to place source.

PosY

Y axis offset within destination raster to place source.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi.h"

Description

This routine will copy the raster data of a source BMP to the raster data of a destination BMP. Translation of pixel type will automatically be performed to the type specified in the destination BMP. Clipping will be performed based on the destination limits (no data will be copied outside of the raster area).

Example

```
extern BMP_type const *picture;
extern BMP_type *frame;

{
    LCDBMPCopy(picture, frame, 20, 40);
}
```

3.2.2 LCDBMPColorCopy

Copies source BMP to destination BMP utilizing color directions from scheme.

Format

```
gapiResp_type LCDBMPColorCopy
(BMP_type const *const s, BMP_type *d, sI16 PosX, sI16 PosY,
GScheme_type const *pScheme);
```

Parameters

- s*
Pointer to the source BMP structure.
- d*
Pointer to the destination BMP structure.
- PosX*
X axis offset within destination raster to place source.
- PosY*
Y axis offset within destination raster to place source.
- pScheme*
pointer to Scheme structure defining color handling.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"
Implemented in file "gapi_button.c"

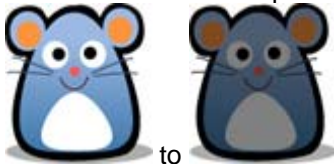
Description

This routine will copy the raster data of a source BMP to the raster data of a destination BMP. Input BMP must be an 8bpp indexed format. The scheme structure contains an option record with a "colorize" field.

If "colorize" is set to 1, this routine will generate a color table that is a gradient determined by the schemes two color entries and use this as the color table for the copy. For example, if the color entries are (0x00, 0x00, 0x00), (0x00, 0xFF, 0x00) a gradient from full black to full green is generated with 256 entries for the copy color table. This is an appropriate method to colorize a grayscale BMP.



If "colorize" is set to 2, this routine will generate a color table that is derived from the source BMP color table. This table is determined by the schemes first color entry, each color channel is scaled in the source table by this value/256. For example, if the scheme color entry is (0x80, 0x80, 0x80) all source color table R:G:B values are multiplied by 50% (128/256). This is an appropriate method to "dim" a color BMP.



Example

See demo code.

3.2.3 LCDBMPIndex

Copies indexed source BMP to destination BMP.

Format

```
gapiResp_type LCDBMPIndex  
( BMP_type const *const s, BMP_type *const d, sI16 PosX, sI16 PosY,  
sI16 Index);
```

Parameters

s

Pointer to the source BMP structure.

d

Pointer to the destination BMP structure.

PosX

X axis offset within destination raster to place source.

PosY

Y axis offset within destination raster to place source.

Index

Index of image within source to copy.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gap.h"

Description

This routine will copy the raster data of a source BMP to the raster data of a destination BMP. The Index field copies a sub-region of the source. The intent here is to allow a simple method to create animated GIF output. Currently the sub-region extracted is a square (based on image height). For example stepping through the following 5 images at a fixed time base produces an animated icon.



Example

See demo code.

3.2.4 LCDBMPScroll

Creates a scrolling banner effect when updated multiple times.

Format

```
void LCDBMPScroll(BMP_type const *s, BMP_type *d,
                 sI16 dPosX, sI16 dPosY, sI16 split, uI16 type);
```

Parameters

- s* Pointer to the source BMP image.
- d* Pointer to the destination BMP.
- dPosX* X axis offset within destination raster to place source image.
- dPosY* Y axis offset within destination raster to place source image.
- split* Point within source image that act as base for drawing. Area before split is placed into destination image after end of source image.
- type* 0= split X axis, 1= split Y axis.

Return Values

none.

Properties

Prototyped in file "gapi.h"
Implemented in file "gapi_effects.c"

Description

Draws a source BMP in two pieces separated at the split point, the first half is placed second, and the second have is placed first. Note that the value of the split is modulo bounded to the axis being split to keep it within the region of the source.

Example

```
extern BMP_type const *picture;
extern BMP_type *frame;

{
    LCDBMPScroll(picture, frame, 20, 40, scroll_split, 1);
}
```



is an 18x104 pixel BMP.



split type 1 at 6 pixels.



split type 1 at 12 pixels



split type 0 at 24 pixels



split type 0 at 80 pixels

3.2.5 LCDBMPCopySub

Copies source BMP to destination BMP. This is the underlying function for many of the other BMP copy functions and offers much control over the copy.

Format

```
gapiResp_type LCDBMPCopy
( BMP_type const *const s, BMP_type *const d, sI16 dPosX, sI16 dPosY,
  sI16 sPosX, sI16 sPosY, sI16 Width, sI16 Height,
  ColorTable_type const *const ct, uI16 tr );
```

Parameters

s

Pointer to the source BMP structure.

d

Pointer to the destination BMP structure.

dPosX

X axis offset within destination raster to place source.

dPosY

Y axis offset within destination raster to place source.

sPosX

X axis offset within source raster to copy from.

sPosY

Y axis offset within source raster to copy from.

Width

Width of source raster area to copy.

Height

Height of source raster area to copy.

ct

pointer to color table to use for pixel conversion/copy.

tr

pixel color to use as transparency color. If value is specified as NO_TRANSPARENCY_COLOR, no transparency handling is performed.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_copy.c"

Description

This routine will copy the raster data of a source BMP to the raster data of a destination BMP. Translation of pixel type will automatically be performed to the type specified in the destination BMP. Clipping will be performed based on the destination limits (no data will be copied outside of the raster area). The ability to specify the color table to use (rather than using one defined in BMP wrapper) allows for ability to colorize grayscale BMPs at runtime.

Example

```
extern BMP_type const *picture;
extern BMP_type *frame;

{
  /* example equivalent to LCDBMPCopy */
  LCDBMPCopySub(picture, frame, 20, 40, 0, 0, BMP_Width(picture),
  BMP_Height(picture), picture->biColorTable, NO_TRANSPARENCY_COLOR);
```

}

3.3 Font Draw Routines

The routines in this section support the placement of text strings into a destination raster.

3.3.1 LCDBMPGPutS

Places a text string into a BMP raster.

Format

```
sI16 LCDBMPGPutS( uI08 const * s, GPUt_type const *Gdata,
  BMP_type *d, sI16 PosX, sI16 PosY);
```

Parameters

s

Pointer to the source text string.

Gdata

Pointer to data structure controlling the font formatting.

d

Pointer to the destination BMP structure.

PosX

X axis offset within destination raster to start placing the text.

PosY

Y axis offset within destination raster to start placing the text.

Return Values

Position of the dominant axis after text output.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_font.c"

Description

This routine uses text from a null terminated ASCII string to put corresponding font BMP data into a destination raster. The GPUt_type structures controls the presentation of the font BMP (refer to following).

- **BMP_type const *pFont:** Points to the BMP used to extract character images.
- **ColorTable_type ct[2]:** Color table used to define 1bpp font coloring. [0] is for background color, [1] is for foreground color.
- **sI16 SpacingX:** This value controls the spacing between characters of the font. A value of "0" will use the fixed width of the font to place the following character. A non-"0" value will force a displacement equal to "SpacingX" from character to character (used when fonts are oriented horizontally).
- **sI16 SpacingY:** This value controls the spacing between characters of the font. A value of "0" will use the fixed width of the font to place the following character. A non-"0" value will force a displacement equal to "SpacingY" from character to character (used when fonts are oriented vertically).
- **sI16 FontOffset:** This value should be set to the value of the first character in the font bitmap. This is useful for using a reduced font set for large font sizes. For example, set to '0' in a font bitmap that only has the images for '0' to '9'.
- **sI16 FontCount:** The number of character images in the font. Set to 0 to let the software automatically detect this value based on BMP height/width ratio (must be 64, 128 or 256 entries in the font to use 0).
- **uI16 Orientation:** Controls the direction to copy the images out. 0=Right/Left, 1: Left/Right, 2: Top/Bottom, 3: Bottom/Top.
- **uI16 Bgnd_transparent:** Flag to set background transparent on the font.
- **uI16 Fgnd_transparent:** Flag to set foreground transparent on the font.

The structure of the font BMP is simply images of characters on a fixed offset from each other. For

example, this BMP with a FontOffset='0' and FontCount=43 would be valid for the shown characters. Note that any character that is not in the table will simply cause the space equal to the character width to be skipped.

0123456789 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Example

```
/* create font with black fore and white background */
const GPUT_type T1 = { &fontTerminal, {{0xFF,0xFF,0xFF,0x00},
{0x00,0x00,0x00,0x00}}, 0, 0, 0, 0, {0,0,0,0,0,0}};

extern BMP_type *frame;

{
  PosX=LCDBMPGPutS("Hello World!", &T1, frame, 0, 0);
}
```

3.3.2 LCDBMP_FontWidth

Request width of individual character within the font BMP.

Format

```
sI16 LCDBMP_FontWidth(GPUT_type const *Gdata);
```

Parameters

Gdata

Pointer to data structure controlling the font formatting.

Return Values

Width of the font character in pixels.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_font.c"

Description

This routine is useful to position text prior to rendering by determining overall string width.

Example

```
{  
  sI16 Width = sprintf(text_buffer, "%3.0d", value) * LCDBMP_FontWidth(&FontCountdown);  
}
```

3.4 Fill Routines

The routines in this section support the filling of raster areas.

3.4.1 LCDBMPFill

Fills destination BMP area with a solid color.

Format

```
gapiResp_type LCDBMPFill(BMP_type *const d, const sI16 dPosX, const sI16 dPosY, sI16 Width, sI16 Height, ColorTable_type const *const ct);
```

Parameters

d

Pointer to the destination BMP structure.

dPosX

X axis offset within destination raster to place fill.

dPosY

Y axis offset within destination raster to place fill.

Width

Width of fill area.

Height

Height of fill area.

ct

Pointer to color table with information for fill color (ct[0] defines fill color).

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_fill.c"

Description

Fills a region of the destination with the color defined by ct[0].

Example

```
extern BMP_type *frame;

{
    const ColorTable_type local_ct[]={0x00, 0x00, 0xFF, 0x00},
                                   {0x00, 0xFF, 0xFF, 0x00}};

    // Fill the background with red
    (void)LCDBMPFillGradient(frame, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, local_ct);
}
```

3.4.2 LCDBMPFillGradient

Fills destination BMP area with a linear gradient of color.

Format

```
gapiResp_type LCDBMPFillGradient(BMP_type *const d, const sI16 dPosX, const
sI16 dPosY, sI16 Width, sI16 Height, ColorTable_type const *const ct, sI16
angle);
```

Parameters

d

Pointer to the destination BMP structure.

dPosX

X axis offset within destination raster to place fill.

dPosY

Y axis offset within destination raster to place fill.

Width

Width of fill area.

Height

Height of fill area.

ct

Pointer to color table with information for fill color (ct[0] and ct[1] define fill gradient).

angle

angle in degrees from ct[0] to ct[1] (currently only 0, 90, 180 and 270 supported).

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_fill.c"

Description

Fills a region of the destination with a gradient of color defined by ct[0] and ct[1].

Example

```
extern BMP_type *frame;
{
    const ColorTable_type local_ct[]={0x00, 0x00, 0xFF, 0x00},
                                {0x00, 0xFF, 0xFF, 0x00}};

    // Fill the background vertically with gradient red to orange
    (void)LCDBMPFillGradient(frame, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, local_ct, 90);
}
```



3.5 Other Routines

The routines in this section are miscellaneous support routines.

3.5.1 LCDBMPCalcGradientCT

Generate an 256 entry index table of color gradients.

Format

```
gapiResp_type LCDBMPCalcGradientCT(ColorTable_type const *const ct,  
uI08 inv, ColorTable_type *const ct_out);
```

Parameters

ct

Pointer to color table with information for fill color (ct[0] and ct[1] define fill gradient).

inv

0=ct[0] to ct[1] gradient, 1=ct[1] to ct[0] gradient.

ct_out

Pointer to user allocated destination color table...note that a color table requires 1Kbyte of RAM.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_fill.c"

Description

Generates a 256 entry color table with the colors linearly varying from ct[0] to ct[1]. This function is useful to generate an index table to colorize 8bpp grayscale BMP source data.

Example

See sample code

3.5.2 LCDBMPCalcShadeCT

Generate a 256 entry index table derived from a source color table.

Format

```
gapiResp_type LCDBMPCalcShadeCT(ColorTable_type const * pS,  
ColorTable_type const *const ct, ColorTable_type * ct_out);
```

Parameters

pS

Pointer to source color table that will be “dimmed”.

ct

Pointer to color table with information for fill color (ct[0] is used to set scaling).

ct_out

Pointer to user allocated destination color table...note that a color table requires 1Kbyte of RAM.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file “gapi.h”

Implemented in file “gapi_fill.c”

Description

Generates a 256 entry color table with the colors linearly scaled by the values in ct[0]. The translation function is:

```
ct_out[N].red = (ct[0].red/256) * pS[N].red ;  
ct_out[N].green = (ct[0].green/256) * pS[N].green ;  
ct_out[N].blue = (ct[0].blue/256) * pS[N].blue ;
```

Note that each color entry (red, green, blue) in ct[0] is used to scale each channel in the source color table...the values of the ct scalar to do not have to be the same. This function is useful to generate an index table to dim an 8bpp BMP.

Example

See sample code

3.5.3 LCDBMPLabel

Centers a text string on BMP/button.

Format

```
gapiResp_type LCDBMPLabel(BMP_type const *const s, BMP_type *d,  
sI16 PosX, sI16 PosY, GScheme_type const *pScheme, uI08 const * label);
```

Parameters

s

Pointer to the source BMP button.

d

Pointer to the destination BMP structure.

PosX

X axis offset within destination raster to place BMP button.

PosY

Y axis offset within destination raster to place button.

pScheme

Pointer to source structure containing font, color and control information.

label

Pointer to text string for label.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_button.c"

Description

Centers a text string on the output raster based on the dimensions of the source BMP button. The source is just used to obtain dimensions and not drawn.

Example

See sample code

3.5.4 LCDBMPButton

Draws a button with text on the destination raster.

Format

```
gapiResp_type LCDBMPButton(BMP_type const *const s, BMP_type *d,  
sI16 PosX, sI16 PosY, GScheme_type const *pScheme, uI08 const * label);
```

Parameters

s

Pointer to the source BMP button.

d

Pointer to the destination BMP structure.

PosX

X axis offset within destination raster to place BMP button.

PosY

Y axis offset within destination raster to place BMP button.

pScheme

Pointer to source structure containing font, color and control information.

label

Pointer to text string for label.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "gapi.h"

Implemented in file "gapi_button.c"

Description

Draws a BMP using information in pScheme structure in the destination raster, then centers the label text string on the button BMP in the destination raster.

Example

See sample code

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (<http://www.renesas.com>)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guarantees regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
 - (1) artificial life support devices or systems
 - (2) surgical implantations
 - (3) healthcare intervention (e.g., excision, administration of medication, etc.)
 - (4) any other purposes that pose a direct threat to human lifeRenesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.