User's Manual

Renesas GAPI Graphics API Revision 2.01

User's Manual

Revision 2.01 July 18, 2012

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

Table of Contents

1. Introduction 5
1.1 GAPI Overview 5
1.2 Philosophy 5
1.3 Capabilities 5
1.4 BMP Creation 6
1.5 BMP Format 6
1.6 Color Information 6
1.7 RLE (Run Length Encoding) Compression 7
1.8 BMP Transparency 7

2. API Information 10
2.1.1 Header Files 10
2.1.2 Configuration 10
2.1.3 Integer Types 10
2.1.4 GAPI Data Types 10
2.1.5 Macro Definitions 10
2.1.6 Dependencies 11
2.1.7 Data Sections 11

3. GAPI Definition 12
3.1 BMP Access and Creation 12
3.1.1 R_BMP_Init 12
3.1.2 R_GAPI_24_Disp 13
3.1.3 R_BMP_Height 14
3.1.4 R_BMP_Width 15
3.1.5 R_BMP_Offset 16
3.1.6 R_BMP_FileSize 17
3.1.7 R_BMP_IndexColors 18
3.2 BMP Copy Routines 19
3.2.1 R_GAPI_CopySub 20
3.2.2 R_GAPI_Copy 23
3.2.3 R_GAPI_CopyTransparent 24
3.2.4 R_GAPI_CopyMulK 25
3.2.5 R_GAPI_CopyMulC 26
3.2.6 R_GAPI_CopyMulT 27
3.3 Font Draw Routines 28
3.3.1 R_GAPI_PutS 28
3.3.2 Font Generation Utilities 31
3.3.3 R_GAPI_FontWidthString 32
3.3.4 LCDBMP_FontHeight 33
3.4 Fill and Translate Routines 34
3.4.1 R_GAPI_Fill 34
3.4.2 R_GAPI_FillMulK 35
3.4.3 R_GAPI_FillGradient 36
3.4.4 R_GAPI_CopyXlate 37
3.5 Other Routines 39
3.5.1 R_GAPI_Index 39
3.5.2 R_GAPI_ColorCopy 40
3.5.3 R_GAPI_Scroll 41
3.5.4 R_GAPI_CalcGradientCTN 42
3.5.5 R_GAPI_CalcShadeCT 43

3.5.6 R_GAPI_Label 44
3.5.7 R_GAPI_Button 45
3.5.8 R_GAPI_PoolMemoryGet 46
3.5.9 R_GAPI_PoolMemoryFree 47

Renesas MCU

GAPI Graphics API Revision 2.01

Revision 2.01
July 18, 2012

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.

1.2 Philosophy

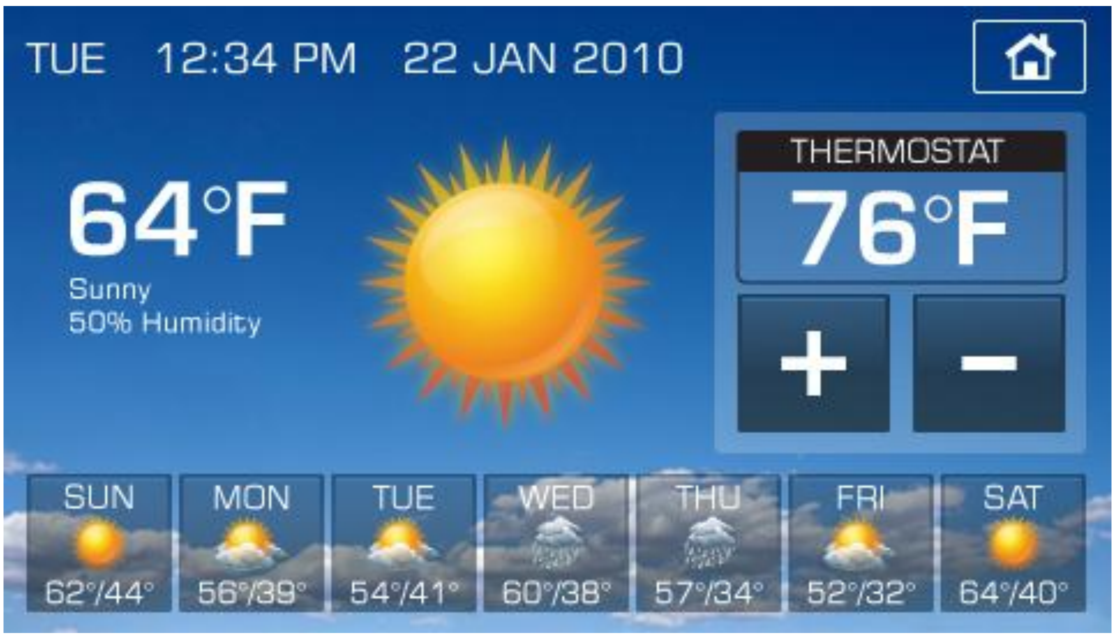
GAPI was developed to provide a set of tools that could be used to illustrate the use of the Renesas LCD graphic solution capabilities. GAPI is freely available to users as part of the Renesas LCD graphic solutions, and includes access to the source code for use on Renesas MCUs. While the solution is 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.

1.3 Capabilities

The current capabilities of the GAPI include:

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

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



1.4 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.

1.5 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.

1.6 Color Information

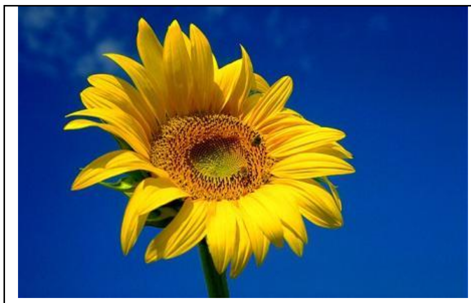
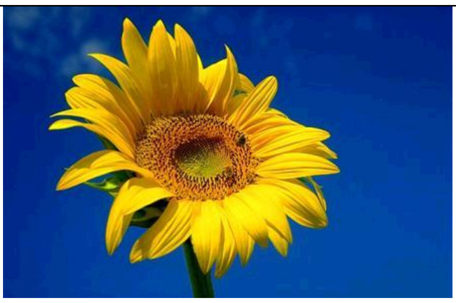
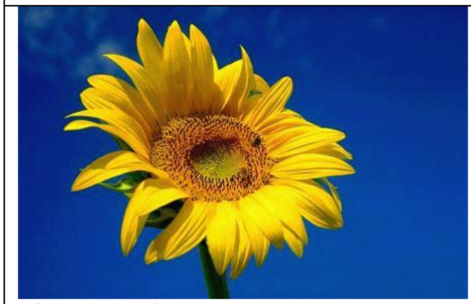
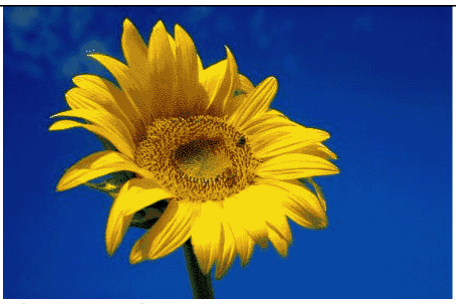
The BMP format allows for tradeoffs in raster size .vs color depth. The raster can be stored as 1, 4, 8, 16, 24 or 32 BPP (bits per pixel). Depending on the software tool, there are different methods to generate the various BMP formats, but often it will be found under a "save as" menu.

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
4 bpp RLE	16 (indexed)	8:8:8	varies
8 bpp	256 (indexed)	8:8:8	76.8 Kbytes
8 bpp RLE	256 (indexed)	8:8:8	varies
16 bpp	64K	5:6:5	153.6 Kbytes
24 bpp	24M	8:8:8	230.4 Kbytes
32 bpp	24M with transparency	8:8:8 + 8 alpha	307.2 Kbytes

A few points from this data.

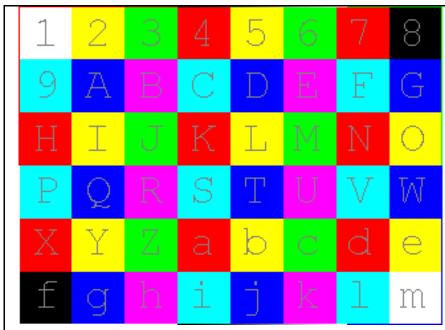
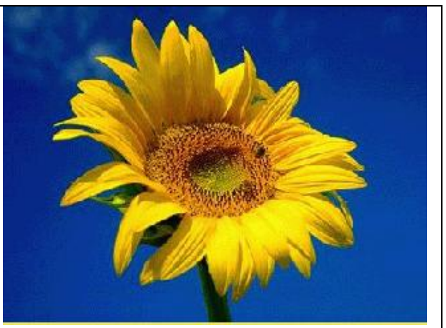
- The indexed formats do not contain the color data within the raster, they contain an index into a color lookup table (contained in the BMP header). So while these formats have a limited number of colors available, those colors can be <u>any</u> color within a 24-bit 8:8:8 (red:green:blue) color space. For example in 1 BPP BMP, the two color choices could be: 🟪 (173:44:201) and 🟦 (36:132:206).
- All output data is converted to 16bpp for display. When using 8:8:8 color formats, there is associated size and conversion overhead that may not justify their use. The additional color data in 24bpp and 32bpp BMP formats are just discarded, so there is no reason to use this formats with GAPI (unless the transparency information is used).
- 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 execution 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 can consume no space as they are generated at runtime by the GAPI.

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.

 **Figure 1 24-bpp original**	 **Figure 2 16-bpp**
 **Figure 3 8-bpp**	 **Figure 4 4-bpp**

1.7 RLE (Run Length Encoding) Compression

GAPI supports RLE compression on 4Bpp and 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).

 **Figure 5 320x240 <u>12.3Kb</u>**	 **Figure 6 320x240 <u>67.9Kb</u>**

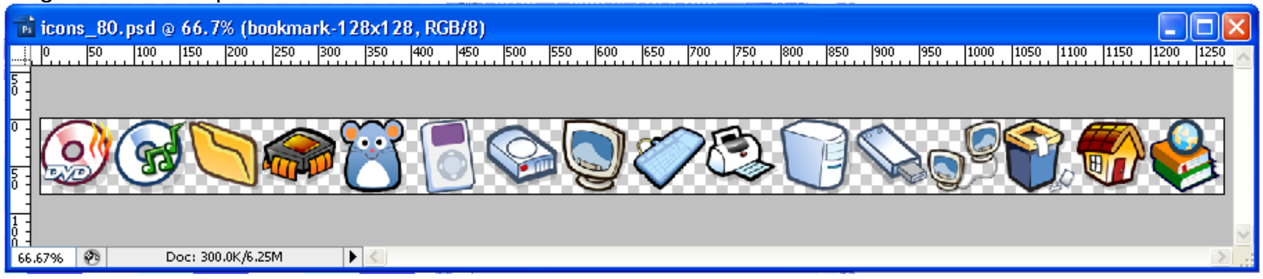
1.8 BMP Transparency

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

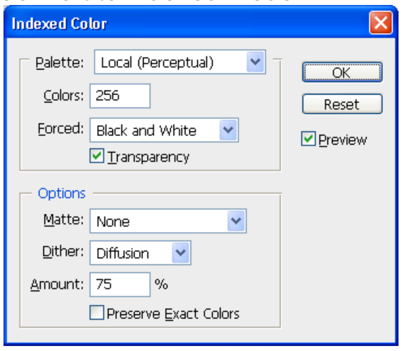
- With 16bpp source images, when calling the "copy" API routines, you can specify a designated 16bpp pixel value to be treated as transparent "magic color". During the copy process, this "transparent" color is <u>not</u> copied to the destination raster. On the source BMP fill any "transparent" areas 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 "magic color" index as part of the index table

(this "magic color" is the last used index in the 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.



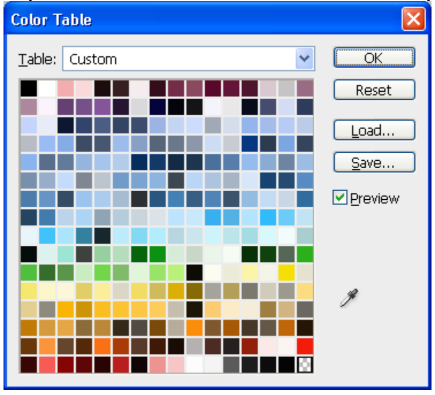
Indexed Color

Palette: Local (Perceptual)
Colors: 256
Forced: Black and White
☑ Transparency

Options
Matte: None
Dither: Diffusion
Amount: 75 %
☐ Preserve Exact Colors

OK
Reset
☑ Preview

- 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.



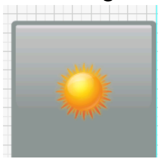
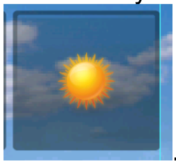
Color Table

Table: Custom

OK
Reset
Load...
Save...
☑ Preview

- This "magic color" technique can yield effective results with small easy to construct BMPs. However, because the a pixel is either "copied" or "not copied" (transparent), the image edges can appear jagged and there is no capability to "blend" an edge pixel into the background pixels.

- These issues can be avoided by use of BMPs that contain an "alpha" (transparency) channel. When rendering images with the alpha channel, the image is seamlessly "blended" into the background.

 can be blended into a frame to produce .

- Also, grayscale images such as this 8bpp RLE can be used as an "alpha mask" and transparently rendered onto frame to produce . Note that the "solid" white color in the source image can be changed at runtime to any color, any pixel that is not "solid" white has the coloring applied and is blended into the destination image on a pixel by pixel basis.

- The detailed API calls in this manual will provide examples of how this is achieved.

2. API Information

The GAPI API follows the Renesas API naming standards.

Additionally, it follows the Renesas Integrated Firmware Development conventions.

2.1.1 Header Files

All API calls are accessed by including a single file “r_GAPI.h” located in the common r_Packages include directory. This header file in turn references several header files buried in the GAPI package. These “sub-header” files are all named “r_GAPI_xxx.h” where “xxx” denotes a class of functions. These files can be located within the HEW “dependencies” tree.

2.1.2 Configuration

The GAPI package configuration is located in “config_r_GAPI.h”. After any configuration changes, the package will need to be rebuilt.

Attribute	Purpose
Raster_t	The Raster_t typdef specifies the data type of the raster memory. Typically this will be set to “uint16_t”.
GAPI_MEMORY_POOLS	This determines the number of 256 and 1024 byte memory pools that are allocated by GAPI for use by the r_GAPI_PoolMemory routines. This value must be at least 2 when using these routines with GAPI (assumes no application usage and a single thread using the GAPI). Setting this value to 0 will build the package without integrated pool support. The user code is then responsible for supplying the R_GAPI_PoolMemory routines.

2.1.3 Integer Types

The GAPI uses ANSI C 99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in “stdint.h”.

2.1.4 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 BMP_ColorTable_t;
/* BMP header structure */
typedef struct BMP_t;
/* Data type of the output raster (user configuration) */
typedef uint32_t Raster_t;
/* structure definition for font information */
typedef struct GAPI_PUT_t;
/* structure definition for higher level color/font schemes */
typedef struct GAPI_Scheme_t;
```

2.1.5 Macro Definitions

These macros are defined as a convenience to the user in specifying common colors.

```
#define CT_WHITE   {0xFF, 0xFF, 0xFF, 0x00}
#define CT_BLACK   {0x00, 0x00, 0x00, 0x00}
#define CT_RED     {0x00, 0x00, 0xFF, 0x00}
#define CT_GREEN   {0x00, 0xFF, 0x00, 0x00}
#define CT_BLUE    {0xFF, 0x00, 0x00, 0x00}
#define CT_CYAN    {0xFF, 0xFF, 0x00, 0x00}
#define CT_MAGENTA {0xFF, 0x00, 0xFF, 0x00}
#define CT_YELLOW  {0x00, 0xFF, 0xFF, 0x00}
#define CT_GRAY_25 {0x40, 0x40, 0x40, 0x00}
#define CT_GRAY_50 {0x80, 0x80, 0x80, 0x00}
#define CT_GRAY_75 {0xC0, 0xC0, 0xC0, 0x00}
```

2.1.6 Dependencies

The Graphics API requires the FreeRTOS package if GAPI_MEMORY_POOLS is not zero.

2.1.7 Data Sections

In addition to the standard compiler sections, the Graphics API uses the BGAPI_POOL* sections to provide temporary pool memory if GAPI_MEMORY_POOLS is not zero. The user project linker options must properly map this section to the appropriate memory location (internal RAM).

3. GAPI Definition

3.1 BMP Access and Creation

Internally, the GAPI wraps all raster images with a BMP header 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 R_BMP_Init

Initialize an allocated BMP_t structure.

Format

```
   GAPI_Resp_t R_BMP_Init
   (BMP_t *const d, uint32_t r, const uint8_t bpp,
              const int16_t Width, const int16_t Height);
```

Parameters

d

Pointer to the user allocated structure to be initialized.

r

Pointer to the user allocated raster that this BMP will be using (as 32bit value…not pointer).

bpp

Bits per pixel of raster, the following are acceptable values:
24, 16, 8, 4, 1: The BMP_t structure will be initialized appropriately.
BMP_DISP_BPP: This is a special value that should be used for creating display frames.

Width

Width of the frame associated with this raster.

Height

Height of the frame associated with this raster.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "r_GAPI.h"

Description

This function is called to initialize a user allocated BMP structure for a given raster to allow its use by other GAPI functions.

Example

```
BMP_t wrapper;
Raster_t raster[160L*120L]
   {
     R_BMP_Init (&wrapper, (uint32_t)raster, BMP_DISP_BPP, 160, 120);
   }
```

3.1.2 R_GAPI_24_Disp

Conversion routine from an 8:8:8 pixel data to a display pixel data.

Format

```
   uint16_t R_GAPI_24_Disp (BMP_ColorTable_t const *const p24 );
```

Parameters

p24

Pointer to the 8:8:8 pixel data.

Return Values

5:6:5 pixel data.

Properties

Prototyped in file ”r_GAPI.h”

Description

Converts 24bpp RGB data to 16bpp 5:6:5 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

```
BMP_ColorTable_t CT;
   {
     uint16_t pixel= LCDBMP24_16( &CT );
   }
```

3.1.3 R_BMP_Height

Extract height information from BMP_t structure.

Format

```
   int16_t R_BMP_Height(BMP_t 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 ”r_GAPI.h”

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_t wrapper;
   {
     int16_t height= R_BMP_Height(&wrapper);
   }
```

3.1.4 R_BMP_Width

Extract width information from BMP_t structure.

Format

```
    int16_t R_BMP_Width(BMP_t 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 "r_GAPI.h"

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_t wrapper;
    {
      int16_t width= R_BMP_Width(&wrapper);
    }
```

3.1.5 R_BMP_Offset

Extract offset to raster information from BMP_t structure.

Format

```
   int32_t R_BMP_Offset(BMP_t 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 ”r_GAPI.h”

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_t wrapper;
{
  Raster_t *pRaster = (pRaster *)((uint8_t *)&wrapper + R_BMP_Offset (&wrapper));
}
```

3.1.6 R_BMP_FileSize

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

Format

```
   uint32_t R_BMP_FileSize(BMP_t const *const pBmp);
```

Parameters

pBmp

Pointer to the BMP structure.

Return Values

File size of BMP.

Properties

Prototyped in file ”r_GAPI.h”

Description

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

Example

```
extern BMP_t *picture;
   {
     uint32_t size = R_BMP_FileSize(picture);
   }
```

3.1.7 R_BMP_IndexColors

Determine number of significant colors in the BMP color table.

Format

```
    uint32_t R_BMP_IndexColors(BMP_t 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 ”r_GAPI.h”

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_t *picture;
    {
      uint32_t colors = R_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. The output (destination format) for all of these routines is Raster_t in 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	
4 bpp RLE	16	8:8:8	Run Length Encoded. Files with only one index color may not be able to be processed when RLE compressed, if this is the case, please force 2 colors and use 2nd color for fill.
8 bpp	256	8:8:8	
8 bpp RLE	256	8:8:8	Run Length Encoded. Files with only one index color may not be able to be processed when RLE compressed, if this is the case, please force 2 colors and use 2nd color for fill.
16 bpp	0	5:6:5	
24 bpp	0	8:8:8	
32 bpp	0	8:8:8:8	Alpha channel processed as transparency value.

3.2.1 R_GAPI_CopySub

Copies source BMP to destination BMP. This is the underlying function used by the following wrapper BMP copy functions and offers much control over the copy.

Generally, the simplified wrapper copy routines will yield more readable code and should be used unless they don't meet the application need.

Format

```
   GAPI_Resp_t R_GAPI_CopySub( BMP_t const *const s, BMP_t *const d,
   int16_t dPosX, int16_t dPosY, int16_t sPosX, int16_t sPosY,
   int16_t Width, int16_t Height, BMP_ColorTable_t const *const ct,
   uint32_t 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
: Utilize the TRANSPARENCY(mode,value) macro to generate an appropriate value.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h” (details in r_gapic_core.h).

Description

This routine will copy/merge the raster data of a source BMP to the raster data of a destination BMP.

Translation of the source pixel Bpp type will automatically be performed to the type specified in the destination BMP Bpp type (currently destination must be the display raster type).

Clipping will be performed based on the supplied Width/Height parameters. No data will be copied from outside the boundaries of source raster nor will any data be written outside the boundaries of the destination raster.

The ability to specify a color table allows runtime recoloring of images. A runtime color table can be generated using GAPI calls such as R_GAPI_CalcGradientCTN and R_GAPI_CalcShadeCT.

A key feature of this function is the ability to specify transparency transformation. The value to pass for the tr argument is generated by the TRANSPARENCY(mode, value) macro as follows:

mode	**value**	**Operation**
GAPI_T_NONE GAPI_TI_NONE	Not used	Source pixel is copied to destination
GAPI_T_MAGIC GAPI_TI_MAGIC	Magic index	For indexed BMP sources, the index value corresponding to the magic color. For 16bpp BMP sources, the 5:6:5 magic color
GAPI_T_MUL GAPI_TI_MUL	Not used	Dest=Dest * Src (5:6:5 channel by channel multiply) This is used to create a color "filter" from the source image.
GAPI_T_MULK GAPI_TI_MULK	5:6:5 color to use as Multiplier	Dest=Src * Value (5:6:5 channel by channel multiply) This can be used to darken a source image
GAPI_T_MULC GAPI_TI_MULC	5:6:5 color to use as "white" replacement	Dest=(Src * Value) + ((1-Src)*Dest) (5:6:5 channel by channel multiply) Treat source image as an alpha mask (white solid, black transparent, intermediate blended)
GAPI_T_MULT GAPI_TI_MULT	5:6:5 color to use as alpha multiplier	Dest=(Value*Src) + (1-Value)*Dest (5:6:5 channel by channel multiply) Blend source into destination by ratio determined by value.
GAPI_T_MULS GAPI_TI_MULS	5:6:5 color to use as alpha multiplier	Dest=(Value*Src*Dest) + (1-Src)*Dest (5:6:5 channel by channel multiply) Source is mask to apply multiplication of value. This is useful for shadowing effects.
GAPI_T_FILL GAPI_TI_FILL	5:6:5 fill color	Dest=Value

Note: The above modes labeled "_TI_" invert the colors of the source image prior to the operation. This is useful in eliminating redundant source images to achieve a desired effect.

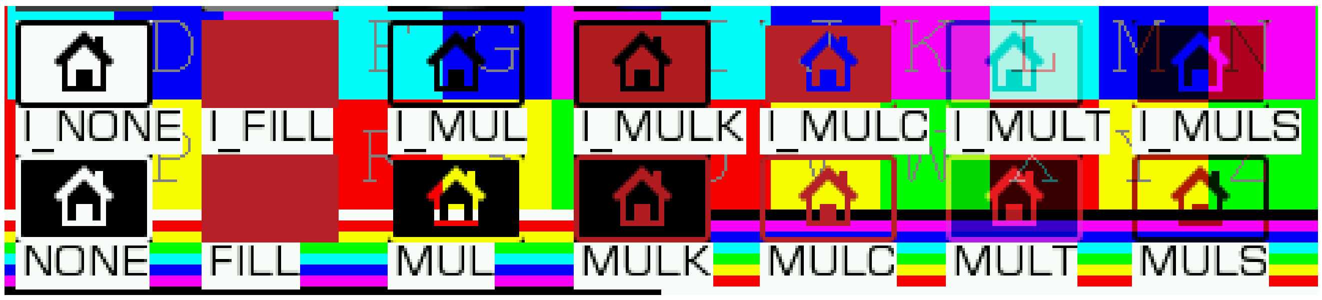
Note: The above operations are implemented completely in software. Great effort was spent to make them as efficient as possible, but the more complex operations may not be appropriate for large source images at high redraw rates (dragging a large icon across the screen for example). Please evaluate the sample applications and your own needs).

Note: In the case of an 8:8:8:8 32bpp source image, the TRANSPARENCY "value" is always supplied by the source image alpha channel (converted to an appropriate 5:6:5 grayscale). By setting the TRANSPARENCY "mode" to GAPI_T_MULT, the alpha channel will provide expected transparency, but other modes may be used to provide other effects.

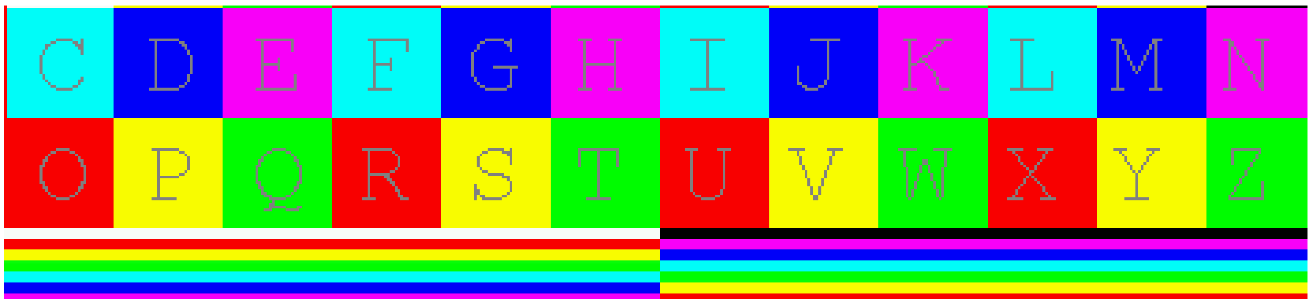
Example

```
extern BMP_t const *picture;
extern BMP_t *frame;
   {
     R_GAPI_CopySub (picture, frame, 20, 40, 0, 0, R_BMP_Width(picture),
       R_BMP_Height(picture), picture->biColorTable,
       TRANSPARENCY(GAPI_T_MULK, CT_GRAY_50 ));
   }
```

The below image illustrates the modes as applied to the "home" source bitmap, showing both the inverted source (top row) and non-inverted source (bottom row) using a shade of "red" for the "value" field. The bottom row "NONE" image represents the unaltered source "Src" bitmap.



The below image is the background data that the above operations were rendered against and was used as the "Dest" bitmap.



3.2.2 R_GAPI_Copy

Copies source BMP to destination BMP wit h no transparency.

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_Copy
   ( BMP_t const *const s, BMP_t *d, int16_t PosX, int16_t 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 ”r_GAPI.h”

Description

This routine will copy the raster data of a source BMP to the raster data of a destination BMP (without any use of transparency).

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_t const *picture;
extern BMP_t *frame;
   {
     R_GAPI_Copy (picture, frame, 20, 40);
   }
```

3.2.3 R_GAPI_CopyTransparent

Copies source indexed BMP to destination BMP (with magic color transparency).

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_CopyTransparent
   ( BMP_t const *const s, BMP_t *d, int16_t PosX, int16_t 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 ”r_GAPI.h”

Description

This routine will copy the raster data of a source BMP to the raster data of a destination. Transparency of the indexed source BMP will be utilized (Transparency color being the last indexed 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_t const *picture;
extern BMP_t *frame;
   {
     R_GAPI_CopyTransparent(picture, frame, 20, 40);
   }
```

3.2.4 R_GAPI_CopyMulK

Copies source indexed BMP to destination BMP (with MulK transformation).

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_CopyMulK
   ( BMP_t const *const s, BMP_t *d, int16_t PosX, int16_t PosY,
     BMP_ColorTable_t const *const ct );
```

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.

ct

Color table pointer to specify the 8:8:8 color to be used for the TRANSPARENCY macro value.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

This routine will copy the raster data of a source BMP to the raster data of a destination. The source image will be rendered darker by multiplying by the color specified by the CT parameter.

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_t const *picture;
extern BMP_t *frame;
const BMP_ColorTable_t myCT= CT_GRAY_75;
   {
     R_GAPI_CopyMulK(picture, frame, 20, 40, &myCT);
   }
```

3.2.5 R_GAPI_CopyMulC

Copies source indexed BMP to destination BMP (with MulC transformation).

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_CopyMulC
   ( BMP_t const *const s, BMP_t *d, int16_t PosX, int16_t PosY,
     BMP_ColorTable_t const *const ct );
```

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.
ct
    Color table pointer to specify the 8:8:8 color to be used for the TRANSPARENCY macro value.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file "r_GAPI.h"

Description

This routine will copy the raster data of a source BMP to the raster data of a destination. The source image will be rendered as an alpha mask (White as solid color, Black as transparent, intermediate values blended). The rendered solid color is specified by the CT parameter.

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_t const *picture;
extern BMP_t *frame;
const BMP_ColorTable_t myCT= CT_RED;
   {
     R_GAPI_CopyMulC(picture, frame, 20, 40, &myCT);
   }
```

3.2.6 R_GAPI_CopyMulT

Copies source indexed BMP to destination BMP (with MulT transformation).

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_CopyMulC
   ( BMP_t const *const s, BMP_t *d, int16_t PosX, int16_t PosY,
     BMP_ColorTable_t const *const ct );
```

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.

ct

Color table pointer to specify the 8:8:8 color to be used for the TRANSPARENCY macro value.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

This routine will copy the raster data of a source BMP to the raster data of a destination. The source image will be rendered with opacity as specified by the CT parameter.

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_t const *picture;
extern BMP_t *frame;
const BMP_ColorTable_t myCT= CT_GRAY_25;
   {
     R_GAPI_CopyMulT(picture, frame, 20, 40, &myCT);
   }
```

3.3 Font Draw Routines

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

3.3.1 R_GAPI_PutS

Places a text string into a BMP raster.

Format

```
    int16_t R_GAPI_PutS ( uint8_t const * s, GAPI_PUT_t const *Gdata,
    BMP_t *d, int16_t PosX, int16_t PosY);
```

Parameters

s

Pointer to the source UTF-8 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 ”r_GAPI.h”

Description

This routine uses text from a null terminated UTF-8 string to put corresponding font BMP data into a destination raster. The GAPI_PUT_t structures controls the presentation of the font file (refer to following).

- Refer to the GAPI_F_xxx Macros for help in specifying the following
- **GAPI_Font_t const **ppFont**: Pointer to pointer to the font file used to extract character images.
- **BMP_ColorTable_t ct[2]**: Color table used to define font coloring. [0] is for background color, [1] is for foreground color.
- **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 “-1” value will force the font to use the “bounding box” character width for horizontal spacing. A positive value will force a displacement equal to “SpacingX” from character to character (used when fonts are oriented horizonatally).
- **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 “-1” value will force the font to use the “bounding box” character width for vertical spacing. A positive value will force a displacement equal to “SpacingY” from character to character (used when fonts are oriented vertically).
- **Align_H**: Specifies the horizontal alignment of the string relative to the insertion point.
  0: Left Justify
  1: Center Justify
  2: Right Justify
- **Align_V**: Specifies the vertical alignment of the string relative to the insertion point
  0: Top Justify
  1: Center Justify
  2: Bottom Justify
- **Bgnd_transparent**: Flag to set background transparency behavior.
  0: Solid
  1: Transparent
- **Fgnd_transparent**: Flag to set foreground transparency behavior.
  For 1bpp fonts, 0: Solid, 1 = Transparent.

For 4bpp and 8bpp fonts, refer to GAPI_T_xxxx behavior (GAPI_T_MULC yielding normal solid, GAPI_T_MULS yielding shadows, etc)

Example

```
/* create font with black fore and white background */
const GAPI_PUT_t T1 = { &fontTerminal, {CT_WHITE,CT_BLACK},
                        GAPI_F_SPACE_PROPORTIONAL, GAPI_F_SPACE_PROPORTIONAL,
                       { GAPI_F_ALIGN_H_CENTER, GAPI_F_ALIGN_V_CENTER,0,0,
                         GAPI_F_BACK_SOLID, GAPI_F_FORE_SOLID}};
extern BMP_t *frame;

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

3.3.2 Font Generation Utilities

Two PC utilities are provided to assist in the creation of the font files.

gapi_font_gen takes the following arguments to create a suitable font file.

Usage: gapi_font_gen.exe [options] font ...

- **B** Render font as bold (pseudo bold via font embolden).
- **I** Render font as italic (pseudo italic via font oblique).
- **R res** Render font at selected Bit/Pixel res=1, 4, 8.
  - 1BPP: smallest size for most fonts, lowest quality, no anti-aliasing will be done.
  - 4BPP: smallest size for anti-aliased fonts, good quality, most processing.
  - 8BPP: best quality (default).
- **e** This option will output file for little-endian target (default is big-endian).
- **d name** Dump information contained within a generated font file.
- **i name** Specify file name to input glyph list from (ascii hex number/glyph).
  If this is not specified, glyph codes 0x20 to 0x7F will be used.
- **o name** Specify file name to output (.efnt extension will be appended).
  If this is not specified, FontName_FontStyle_[BIM]Size.efnt will be used.
- **p size** The pixel size to generate the font at. This specifies the 'EM' square (default=16).
- **r** Disable use of Run Length Encoding (default is use RLE if smaller file would be generated).
- **Font** The source font file to render glyphs from.

gapi_font_gen is based on FreeType 2 and supports the following font formats.

- TrueType fonts (and collections)
- Type 1 fonts
- CID-keyed Type 1 fonts
- CFF fonts
- OpenType fonts (both TrueType and CFF variants)
- SFNT-based bitmap fonts
- X11 PCF fonts
- Windows FNT fonts
- BDF fonts (including anti-aliased ones)
- PFR fonts
- Type 42 fonts (limited support)

utf creates a list of UTF-32 characters that are contained within a source file. This list can then be used by the “gapi_font_gen” “-i” option to create a font file that contains these specific glyphs (without using the “-i” option, glyphs 0x20 to 0x7F will be included).

- <in> Source file to obtains list from.
- <out> Name of output file to write list to.

3.3.3 R_GAPI_FontWidthString

Request width of a character string as would be rendered.

Format

```
   int16_t R_GAPI_FontWidthString (uint8_t const * s, GAPI_PUT_t const *Gdata);
```

Parameters

s

Pointer to the source UTF-8 text string.

Gdata

Pointer to data structure controlling the font formatting.

Return Values

Width of the string in pixels.

Properties

Prototyped in file ”r_GAPI.h”

Description

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

Example

```
/* create font with black fore and white background */
const GAPI_PUT_t T1 = { &fontTerminal, {CT_WHITE,CT_BLACK},
                        GAPI_F_SPACE_PROPORTIONAL, GAPI_F_SPACE_PROPORTIONAL,
                       { GAPI_F_ALIGN_H_CENTER, GAPI_F_ALIGN_V_CENTER,0,0,
                         GAPI_F_BACK_SOLID, GAPI_F_FORE_SOLID}};
{
  int16_t Width = R_GAPI_FontWidthString(“Hello World”, &T1);
}
```

3.3.4 LCDBMP_FontHeight

Request height of a given font.

Format

```
   int16_t R_GAPI_FontHeight(GAPI_PUT_t const *Gdata);
```

Parameters

s

Pointer to the source UTF-8 text string.

Gdata

Pointer to data structure controlling the font formatting.

Return Values

Height of the string in pixels.

Properties

Prototyped in file ”r_GAPI.h”

Description

This routine returns the height of the fonts “bounding box” (height of lowest part of any glyph to highest part of any glyph).

Example

```
/* create font with black fore and white background */
const GAPI_PUT_t T1 = { &fontTerminal, {CT_WHITE,CT_BLACK},
                        GAPI_F_SPACE_PROPORTIONAL, GAPI_F_SPACE_PROPORTIONAL,
                       { GAPI_F_ALIGN_H_CENTER, GAPI_F_ALIGN_V_CENTER,0,0,
                         GAPI_F_BACK_SOLID, GAPI_F_FORE_SOLID}};
{
  int16_t Height = R_GAPI_FontHeight (&T1);
}
```

3.4 Fill and Translate Routines

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

3.4.1 R_GAPI_Fill

Fills an area of the destination BMP with a specified color.

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_Fill
   (BMP_t *d, int16_t PosX, int16_t PosY, int16_t Width, int16_t Height,
     BMP_ColorTable_t const *const ct );
```

Parameters

s

Pointer to the source BMP structure.

d

Pointer to the destination BMP structure.

PosX

X axis offset within destination raster to fill.

PosY

Y axis offset within destination raster to fill.

Width

Width of destination raster area to fill.

Height

Height of destination raster area to fill.

ct

Color table pointer to specify the 8:8:8 color to be used for the TRANSPARENCY macro value.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

This routine will fill the specified raster data of the destination with a color specified by the CT parameter.

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_t const *picture;
extern BMP_t *frame;
const BMP_ColorTable_t myCT= CT_MAGENTA;
   {
     R_GAPI_Fill(picture, frame, 20, 40, &myCT);
   }
```

3.4.2 R_GAPI_FillMulK

Darken a specified area of the destination by specified color value.

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
   GAPI_Resp_t R_GAPI_FillMulK
   (BMP_t *d, int16_t PosX, int16_t PosY, int16_t Width, int16_t Height,
     BMP_ColorTable_t const *const ct );
```

Parameters

s
    Pointer to the source BMP structure.
d
    Pointer to the destination BMP structure.
PosX
    X axis offset within destination raster to fill.
PosY
    Y axis offset within destination raster to fill.
Width
    Width of destination raster area to fill.
Height
    Height of destination raster area to fill.
ct
    Color table pointer to specify the 8:8:8 color to be used for the TRANSPARENCY macro value.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

This routine will darken a region of the destination with a color specified by the CT parameter.

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_t const *picture;
extern BMP_t *frame;
const BMP_ColorTable_t myCT= CT_GRAY_50;
   {
     R_GAPI_FillMulK (picture, frame, 20, 40, &myCT);
   }
```

3.4.3 R_GAPI_FillGradient

Fills destination BMP area with a linear gradient of color.

Format

```
   GAPI_Resp_t R_GAPI_FillGradient(BMP_t *const d, const int16_t dPosX, const
int16_t dPosY, int16_t Width, int16_t Height, BMP_ColorTable_t const *const ct,
int16_t 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

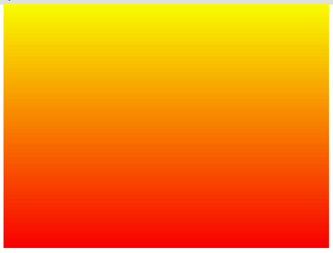
Prototyped in file ”r_GAPI.h”

Description

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

Example

```
extern BMP_t *frame;
{
  const BMP_ColorTable_t local_ct[]={CT_RED, CT_YELLOW};
  // Fill the background vertically with gradient red to yellow
  (void) R_GAPI_FillGradient(frame, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, local_ct, 90);
}
```



3.4.4 R_GAPI_CopyXlate

Translate a 16Bpp native region.

Format

```
   GAPI_Resp_t R_GAPI_CopyXlate
   ( BMP_t const *const s, BMP_t *const d, int16_t dPosX, int16_t dPosY,
     int16_t sPosX, int16_t sPosY, int16_t Width, int16_t Height, uint16_t Mode
);
```

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.

Mode

Flags to control behavior of the translation.

GAPI_ CX_ROT_0	No rotation.
GAPI_ CX_ROT_90	Rotate 90 degrees.
GAPI_ CX_ROT_180	Rotate 180 degrees.
GAPI_ CX_ROT_270	Rotate 270 degrees
GAPI_ CX_FLIPX	Flip region left to right
GAPI_ CX_FLIPY	Flip region top to bottom.

Return Values

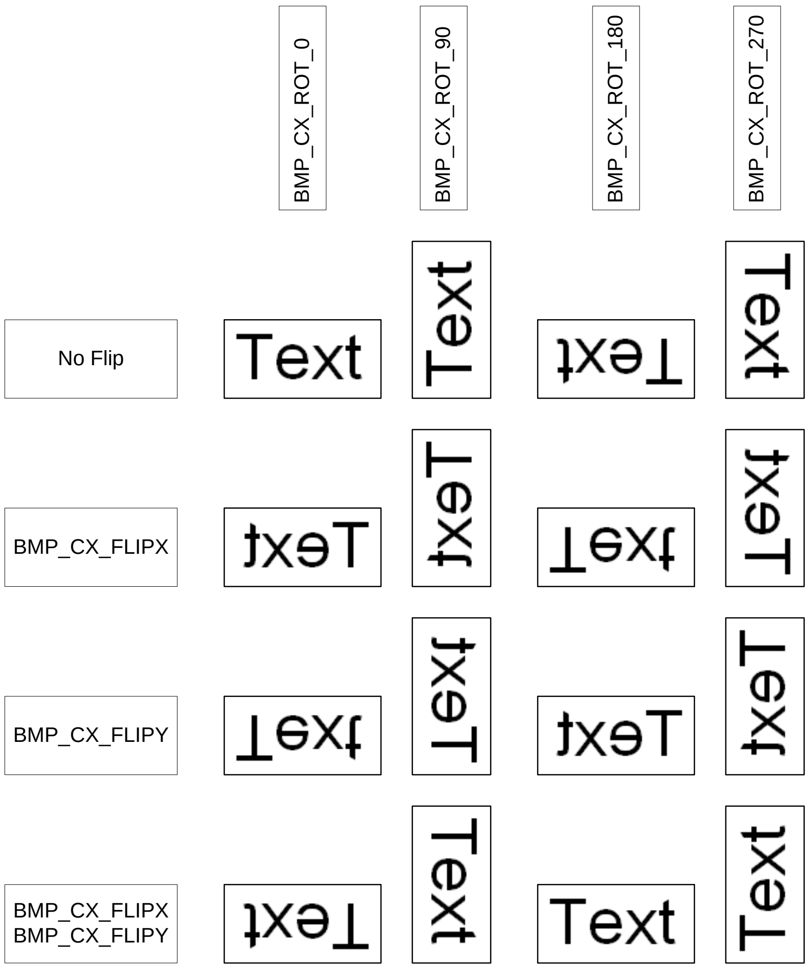
0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

Move and translate a 16bpp region controlled by the Mode flags as follows.



Example

```
extern BMP_t const *picture;
extern BMP_t *frame;
   {
     /* Translate region of source picture to destination frame */
     R_GAPI_CopyXlate(picture, frame, 20, 40, 0, 0, R_BMP_Width(picture),
       R_BMP_Height(picture), (GAPI_ CX_ROT_90 | GAPI_ CX_FLIPX) );
   }
```

3.5 Other Routines

The routines in this section are miscellaneous support routines.

3.5.1 R_GAPI_Index

Copies one instance of indexed source BMP to destination BMP.

This function is implemented as inline wrapper function for R_GAPI_CopySub.

Format

```
GAPI_Resp_t R_GAPI_Index
( BMP_t const *const s, BMP_t *const d, int16_t PosX, int16_t PosY,
int16_t 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

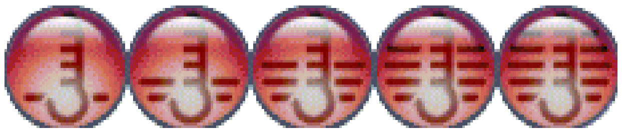
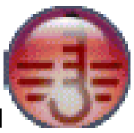
0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

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

 specifying index = 2 would yield 

Example

```
See demo code.
```

3.5.2 R_GAPI_ColorCopy

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

Format

```
GAPI_Resp_t R_GAPI_ColorCopy
(BMP_t const *const s, BMP_t *d, int16_t PosX, int16_t PosY,
GAPI_Scheme_t 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

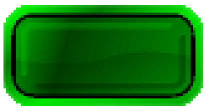
Prototyped in file ”r_GAPI.h”

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 “CT_COLOR_NO_CHANGE”, no color manipulation is performed on the index table.

If “colorize” is set to “CT_COLOR_LINEAR_CHANGE”, 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 CT_BLACK, CT_GREEN 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.

to 

If “colorize” is set to “CT_COLOR_MUL_CHANGE”, 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 CT_GRAY_50 all source color table R:G:B values are multiplied by 50% (128/256). This is an appropriate method to “dim” a color BMP.

to

Example

```
See demo code.
```

3.5.3 R_GAPI_Scroll

Creates a scrolling banner effect when updated multiple times.

Format

```
   void R_GAPI_Scroll(BMP_t const *s, BMP_t *d,
   int16_t dPosX, int16_t dPosY, int16_t split, uint16_t 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

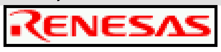
Prototyped in file "r_GAPI.h"

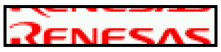
Description

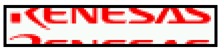
Draws a source BMP in two pieces separated at the split point, the first half is placed second, and the second have is placed first. Note that the value of the split is modulo bounded to the axis being split to keep it within the region of the source.

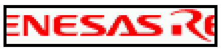
Example

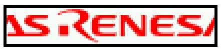
```
extern BMP_t const *picture;
extern BMP_t *frame;
   {
     R_GAPI_Scroll(picture, frame, 20, 40, scroll_split, 1);
   }
```


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

3.5.4 R_GAPI_CalcGradientCTN

Generate an index table of color gradients.

Format

```
    GAPI_Resp_t R_GAPI_CalcGradientCTN(BMP_ColorTable_t const *const ct,
    uint8_t mode, BMP_ColorTable_t * ct_out, int16_t N);
```

Parameters

ct

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

mode

0=ct[0] to ct[1] gradient
1=ct[1] to ct[0] gradient
2=0:0:0 to ct[0] gradient

ct_out

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

N

Number of entries to generate in the color table…typically 256 or 16.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

Generates an N 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 indexed grayscale BMP source data.

Example

```
See sample code
```

3.5.5 R_GAPI_CalcShadeCT

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

Format

```
   GAPI_Resp_t R_GAPI_CalcShadeCT(BMP_ColorTable_t const * pS,
   BMP_ColorTable_t const *const ct, BMP_ColorTable_t * ct_out);
```

Parameters

pS

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

ct

Pointer to color table with information for manipulation 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 ”r_GAPI.h”

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
```

3.5.6 R_GAPI_Label

Positions a text string within a BMP.

Format

```
   GAPI_Resp_t R_GAPI_Label(BMP_t const *const s, BMP_t *d,
   int16_t PosX, int16_t PosY, GAPI_Scheme_t const *pScheme, uint8_t 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 the text label.

PosY

Y axis offset within destination raster to place text label.

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 ”r_GAPI.h”

Description

Positions a text string on the output raster based on the dimensions of the source BMP button. The CT_TEXT_xxx macros can be used to position the text relative to the dimensions of the source bitmap.

The source is just used to obtain dimensions and not drawn.

Example

```
See sample code
```

3.5.7 R_GAPI_Button

Draws a BMP with a label on the destination raster.

Format

```
   GAPI_Resp_t R_GAPI_Button(BMP_t const *const s, BMP_t *d,
   int16_t PosX, int16_t PosY, GAPI_Scheme_t const *pScheme, uint8_t 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 ”r_GAPI.h”

Description

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

This function is a wrapper for “R_GAPI_ColorCopy” and “R_GAPI_Label” GAPI calls.

Example

```
See sample code
```

3.5.8 R_GAPI_PoolMemoryGet

Request a block of memory from the GAPI pool.

Format

```
    uint8_t * R_GAPI_PoolMemoryGet(uint16_t size);
```

Parameters

size

Size of requested memory block (256 or 1024).

Return Values

NULL if unable to allocate, pointer to allocated memory if successful.

Properties

Prototyped in file ”r_GAPI.h”

Description

Requests a block of memory from the GAPI pool.

The number of available blocks of each size is configured with the GAPI_MEMORY_POOLS macro. This memory is allocated in the GAPI_POOL section, which must be located in the project by the linker options.

If the user wishes to use another memory allocation mechanism for GAPI, set “GAPI_MEMORY_POOLS” to zero in the configuration file and create a local function to supply this behavior. If this function is not overridden, the FreeRTOS package is also required in the system.

Example

```
  pLocalCT16 = (void *)R_GAPI_PoolMemoryGet(1024);
```

3.5.9 R_GAPI_PoolMemoryFree

Release a block of memory from the GAPI pool.

Format

```
    uint8_t R_GAPI_PoolMemoryFree(uint8_t *pPool);
```

Parameters

pPool

Pointer to the previously allocated block.

Return Values

0 if successful, non-zero if failure.

Properties

Prototyped in file ”r_GAPI.h”

Description

Releases a block of memory from the GAPI pool.

If the user wishes to use another memory allocation mechanism for GAPI, set “GAPI_MEMORY_POOLS” to zero in the configuration file and create a local function to supply this behavior. If this function is not overridden, the FreeRTOS package is also required in the system.

Example

```
  (void)R_GAPI_PoolMemoryFree((void *)pLocalCT);
```

REVISION HISTORY	Renesas GAPI Graphics API Revision 2.01 User's Manual

Rev.	Date	Description	
		Page	**Summary**
1.00	Jan.28.09	—	First edition issued
1.10	Mar.17.09		Updated to latest API format.
1.20	Sep.25.09		Added proportional font support. Added “LCDBMPCopyXlate” API Call Added mode to “LCDBMPCalcGradientCTN” API Call Added “LCDBMP_FontHeight” API Call Added “LCDBMP_FontWidthString” API Call (replacing LCDBMP_FontWidth API Call) Added “LCDBMPCopyTransparent” API Call
1.21	Oct.7.09		Latest gapi_font_gen options
2.00	Sep.25.10		Conversion to RAPI format Added GAPI blending modes Documentation made more generic to support multiple MCU families.
2.01	Jul.18.12		Additional T_MODE selections documented

Renesas GAPI Graphics API Revision 2.01
User's Manual

Publication Date:	Rev.2.01 July 18, 2012
Published by:	Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Renesas MCU
GAPI Graphics API Revision 2.00

Renesas Electronics Corporation

R01UHxxxxEJxxxx
(Previous Number:REJ09Bxxxx)