GP32 GPSDK API REFERENCE Ver 2.1.5

Created on : April1, 2002

Created by : A-Chi Chung

Ji-Hyeon Lim

Source : Game Park Inc. R&D Center

Contents

Table of API Contents

GpNetParseIp	GpSockCreate
GpSockConnect	GpSockSend
GpSockSendto	GpSockRecv
GpSockRecvfrom	GpSockClose
GpSockShutdown	GpPPPShutdown
GpSockPeerGet	GpSockNameGet
GpSockOptGet	GpSockOptSet
GpSockConnected	

GP OS API

GpTimerOptSet	GpKernelOptSet
GpTimerSet	GpKernelLock
GpTimerPause	GpKernelUnlock
GpTimerResume	GpThreadSleep
GpTimerKill	GpThreadOptSet
	GpThreadHandleGet
GpMain	GpNetThreadAct
GpKernelInitialize	GpNetThreadDelete
GpKernelStart	GpPredefinedStackGet

① GpArgSet, GpArgGet : The reliability is not guaranteed at Version 2.0.0.

② GpAppExit : The reliability is not guaranteed at Version 2.0.0.

③ System call

④ GpUSBLineCheck : It serves as system call and it is not recommended for use at Version 2.0.0.

⑤ GpFxBlt : It is recommended for use ONLY in case that an ARM7 version of the PALETTE is used.

FILE LISTS

init.o	gpstdlib.alf
gpsound.alf	gpgraphic.alf
gpgraphic8.alf	gpgraphic16.alf
gpfont.alf	gpfont8.alf
gpfont16.alf	gpstdio.alf
gpmem.alf	gpg_ex01.alf
gpnet.alf	gpos.alf
target₩initval_port.h	target₩user_init.s
target₩gpstart.c	target₩fontres.dat

OPTIONS

The ARM-CC and ARM-ASM options on the ARM SDT 2.5x, PROJECT MANAGER tool are illustrated as follows.

DEBUG :

ARM-CC ₩ Preprocess

GP_DBG=1

ARM-ASM ₩ Predefines

GP_DBG={TRUE}

RELEASE :

ARM-CC ₩ Preprocess

GP_DBG=0

ARM-ASM ₩ Predefines

GP_DBG={FALSE}

GP Standard Library

This is the most basic library. The file name is ‘gpstdlib.alf’.

Implemented Functions :

- Keyboard related functions
- Communication manage functions
- Heap, String Library porting layers
- Processor clock speed & CACHE manage functions

GP32 Key Input Control

Overview :

As shown in the diagram on the right, GP32 has 6 function keys and a joystick that recognizes data on 8 directions.

Key data defined in ₩gpinclude₩gpdef.h is as follows :

GPC_VK_NONE	0x00	GPC_VK_RIGHT	0x04	GPC_VK_FL	0x10	GPC_VK_START	0x100
GPC_VK_LEFT	0x01	GPC_VK_UP	0x08	GPC_VK_FB	0x20	GPC_VK_SELECT	0x200
GPC_VK_DOWN	0x02	GPC_VK_FR	0x80	GPC_VK_FA	0x40		

Current key value is calculated by using the GpKeyGet, GpKeyGetEx function. As the key value turns round in a loop as many times as the value of gpKEYPOLLNUM, the values of the button scanned will be calculated in bitwise OR and returned. The constant of ₩target₩initval_port.h KEYPOLLING_NUM is the basic value of gpKEYPOLLNUM at system initialization. The bigger the value, the longer the time checking the key value will be. (This will also cause a lag in the speed of program.) An example is as follows:

```
for ( int i = 0 ; i < KEYPOLLING_NUM ; i + +  )
{
        key value  | = ( button scan value ) ;
}
```

The GpKeyChanged, GpKeyGetEx functions perform the bitwise OR calculation of the changed KEY value using values in the above table and returns. An example is as follows:

```
01          : GpKeyInit();
02          : // Press START button and move JOYSTICK left and up.
```

```
03          : key_result = GpKeyGet();
04          : changed_key = GpKeyChanged();
05          : // Press SELECT, A and put your JOYSTICK to the center position.
06          : changed_key = GpKeyGetEx(&key_result);
07          : GpKeyInit();
08          : // Do not press any key.
09          : changed_key = GpKeyGetEx(&key_result);
```

01 : Initialize KEY status relevant system parameter.
03 : key_result == (GPC_VK_START | GPC_VK_LEFT | GPC_VK_UP)
04 : changed_key == (GPC_VK_START | GPC_VK_LEFT | GPC_VK_UP)
06 : key_result == (GPC_VK_SELECT | GPC_VK_FA)
changed_key == (GPC_VK_START | GPC_VK_SELECT | GPC_VK_FA | GPC_VK_LEFT | GPC_VK_UP)
07 : Initialize KEY status relevant system parameter.
08 : key_result == (GPC_VK_NONE)
changed_key == (GPC_VK_NONE)

KEY INPUT FUNCTIONS :

void GpKeyInit(void);

This function tracks KEY value changes and initializes relevant parameters. Right after calling GpKeyInit, both GpKeyChanged and GpKeyGetEx return zero.

int GpKeyGet(void);

This function bit-sets the pressed buttons except for START, SELECT buttons and returns the 8bit value. The result is identical to the bitwise OR calculation of constants in the above table. It is positioned as API compatible with ARM7 at Version 2.0.0.

int GpKeyGetEx(int * key_status);

This function returns 10-bit value including the START, SELECT buttons through KEY_status. The returned value is the 10-bit value and bitwise OR calculation of the buttons whose values have been changed.

int GpKeyChanged(void);

This function bit-sets and returns the changed KEY value comparing with the latest KEY value right before calling GpKeyGet and GpKeyGetEx.

Header : gpdef.h, gpstdlib.h

Device / Application / Firmware INFORMATION Controls

Overview :

This is used to define APIs with regard to collecting information including a user's device ID, Password, Alliance Execution Path, Argument Factor, and System Tick from the device and the firmware. The relevant information is initialized and managed by the device launcher which is a basic program executed when the power is reset.

Functions :

int GpAppPathSet(const char * p_path, int n_len);

The execution path of the program must be set up in the subject program. Since the execution path is designated when the device launcher runs the subject program in RELEASE mode, you cannot get the right execution path in DEBUG mode. To troubleshoot this problem, when it comes to the DEBUG mode, a developer must call GpAppPathSet in the forepart of GpMain to designate the right path. N_len refers to the length of p_path. The maximum value is 256 and it must be terminated with null-string.

Note) The GP_DBG option should be set to 1 in the ARMSDT 2.5x APM and ARM-CC₩Preprocess menu.

```
void GpMain(void * arg)
{
      ………………..
      #if GP_DBG
      GpAppPathSet("gp:₩₩game₩0", 20);
      #endif
      ………………
}
```

char * GpAppPathGet(int * n_len);

This function returns the execution path of current running program. n_len refers to the length of executation path returned by this function.

int GpUserInfoGet(char * p_id, char * p_pwd);

This function returns a user’s id and password embedded in the device. The factory default id and password are respectively ‘GUEST’ and ‘GUEST’.

int GpArgSet(int len, char * p_arg); / int GpArgGet(char * p_arg);

This function is not recommended for use at Version 2.0.0 due to lack of reliability.
Version 2.0.0 에서는 신뢰도가 낮아 사용을 권장하지 않는다.

void GpAppExit(void);

This function terminates the program and puts the system into POWER RESET state. Its use is not recommended at Version 2.0.0.

unsigned int GpTickCountGet(void);

This function returns system tick count in milli seconds after the program starts to run.

int GpUSBLineCheck(void);

This function checks whether USB line is established or not. If the device manager of the PC host recognizes the USB connection, it returns 1, if not, it returns 0.

Header : gpstdlib.h

GP32 Communication controls

Overview :

This is designed to provide the driver API required to get access to the communication port of GP32 device. GP32 offers a maximum of 4 communication ports that can be identified by the next constant.

COMM_PORT_0	: Use this command to get access to UART 0 port.
COMM_PORT_1	: Use this command to get access to UART 1 port.
COMM_USB_D	: Use this command to get access to USB device port.
COMM_USB_H	: Use this command to get access to USB Host port.

GP32 also provides the following LOGICAL COMMUNICATION LAYER to enhance the future upgradeability and scalability. A developer must use the following information to get access to the driver in consideration of the software compatibility.

COMM_PORT_RF : This commnand designates the port to be used for short-range wireless networking and GP32 provides the mapping of the subject port using UART 0.

COMM_PORT_NET: This command designates the port to be used for TCP/IP based network and GP32 provides the mapping of the subject port using UART 0.

Functions :

int GpCommCreate(GPN_DESC * p_desc, GPN_COMM * p_play);

This function designates the option of port to be used and initializes the instance of the GPN_COMM structure by using GPN_DESC structure. All communication APIs must be used only after GpCommCreate proves successful.

void GpCommDelete(GPN_DESC * p_desc);

Type definitions :

```
typedef struct tagGPN_DESC{
        int port_kind;            → This function designates a GP port that you want to use (COMM_PORT_x).
        int tr_buf_size;          → The maximum buffer size for Tx/RxTx/Rx is 4KB, which is a necessity only for UART
                                  purpose
        int tr_rate;              → This function designates baudrate.   It is required only for UART purpose.
        int tr_mode;              → Ths function determines whether to set Interrupt mode or not. It is always 1 at Version
                                  2.0.0.
        int sz_pkt;               → This function defines packet size. Use this function only for USB purpose.
        void (*isr_comm_ram)(void); → If non NULL, interrupt service will be hooked.
        int reserved1;
        int reserved2;
}GPN_DESC;

typedef struct tagGPSTRFUNC{
       int (*comm_open)(GPN_DESC * p_desc);          → Open Port.
       void (*comm_hw_reset)(void);                  → Reset Port Hardware.
       void (*comm_sw_reset)(void);                  → Reset Port Software.
       void (*comm_close)(void);                     → Close Port.
       int (*comm_send_ready)(void);                 → Return whether Tx transmission is ready.
       int (*comm_send)(unsigned char * p_data, int n_sz);    → Transmit BYTES stream via Tx.
       int (*comm_send_one)(unsigned char c_data);            → Transmit BYTE via Tx.
       int (*comm_recv)(unsigned char * p_data, int n_max_sz);       → Receive BYTES stream via Rx.
       int (*comm_recv_one)(unsigned char * p_data);                 → Receive BYTE via Rx.
       unsigned char (*comm_recv_sync)(void);                 → Receive BYTE in BLOCKING mode via RX.
       unsigned char * p_comm_buf;                            → The actual buffer address for transceiving
```

```
        int * p_buf_size;                      → Transceiving buffer size.
        int * p_comm_var[4];                   → The addresses of variables for processing transceiving buffer queue
        void (*force_env_clear)(void);         → Force driver initialization.
        int (*comm_port_ready)(void);          → Return whether port is ready to receive data.
        void (*reserved1)(void);
        void (*reserved2)(void);
        void (*reserved3)(void);
        void (*reserved4)(void);
        void (*reserved5)(void);
    }GPN_COMM;
```

Header : gpcomm.h

Heap, String Function Porting Layers

Overview :

GPSDK 2.1.5 provides memory allocation / removal library and basic character string control library. In order to use an additional library, you need to re-allot or develop separately the instance members (function pointer) of GPMEMFUFC and GPSTRFUNC structure body. In other words, gp_mem_func, a GPMEMFUNC-type global variable, and gp_str_func, a GPSTRFUFNC-type global variable act as porting layers.

GPSDK 2.1.5 provides gpmem.alf library.

GPMEMFUNC gp_mem_func

void * (*malloc)(unsigned int size);	➔ Function pointer for malloc.
void * (*zimalloc)(unsigned int size);	➔ Function pointer to initialize malloc area to 0.
void * (*calloc)(int count, unsigned int size);	➔ Funtion pointer for calloc.
void (*free)(void * pt);	➔ Function pointer for memory removal (to free memory).
unsigned int (*availablemem)(void);	➔ Function pointer for calculating the size of available malloc area in heap area.

void * (*malloc_ex)(unsigned int size, int ex_flag, unsigned char init_val);

void (*free_all)(void);

void (*make_mem_partition) (struct tagGPMEMFUNC * p_mem_sub, unsigned int size, int * err_no);

GPSTRFUNC gp_str_func

void (*memset)(void * ptr, unsigned char val, unsigned int size)	➔ Function pointer to initialize ptr to val as much as the size.
void * (*memcpy)(void * s1, const void * s2, unsigned int size)	➔ Function pointer to copy from s2 to s1 as much as the size.
char * (*strcpy)(char * s1, const char * s2)	➔ Function pointer to copy from character string s2 to s1. The end of s2 and s1 are null characters.
char * (*strncpy)(char * s1, const char * s2, unsigned int size)	➔ Function pointer to perform strcpy as much as the size.

char * (*strcat)(char * s1, const char * s2)	➔ Function pointer to add string s2 to string s1.
char * (*strncat)(char * s1, const char * s2, unsigned int size)	➔ Function pointer to perform strcat as much as the size.
int (*gpstrlen)(const char * s)	➔ Function pointer to return the calculation of s length in bytes.
int (*sprintf)(char * buf, const char * fmt, ...)	➔ Function pointer to perform sprintf.
void (*uppercase)(char *ptr, int count)	➔ Function pointer to change the string of ptr to capital letters as many as counted.
void (*lowercase)(char *ptr, int count)	➔ Function pointer to change the string of ptr to small letters as many as counted.
int (*compare)(const char *pt1, const char *pt2)	➔ Function pointer to return 0 when pt1 and pt2 are totally the same.
void (*trim_right)(const char *ptr);	

Using GPSDK Version 2.1.0 Basic Heap Library

● GP32 Memory Architecture

GP32 does a one-to-one mapping between virtual and physical memories. The structural overview is as follows:

```
RAM_START_ADDRESS ---------------------------→ Conform to input value and generally 0x0C000000.
      CODE AREA – READ ONLY
RAM_READ_ONLY_LIMIT_ADDRESS -----------------→ CODE SIZE
      DUMMY AREA –   NOT USED
RAM_READ_WRITE_BASE_ADDRESS ----------------→ Conform to input value and the default value is 0x0C080000 (i.e.
                                               code < 512KB)
READ WRITE & INITIALIZED VARIABLES
RAM_ZERO_INIT_BASE_ADDRESS ------------------→ Starting address of static and global variables which don't have to
                                               be initialized.
      ZERO INIT – NON-INITIALIZED VARIABLES. I.E. ZERO!!!
```

RAM_ZERO_INIT_LIMIT_ADDRESS ------------------→ Ending address of variables which don't have to be initialized.

Not used (< 1KB)

RAM_HEAP_START_ADDRESS ---------------------→ Starting address of HEAP

HEAP AREA :

The general formula to calculate HEAP size = (RAM_HEAP_END_ADDRESS) – (RAM_HEAP_START_ADDRESS)

RAM_HEAP_END_ADDRESS ----------------------→ Dependent on the reservation size of FIRMWARE, generally 0x0C780000

● Using GM_HEAP_DEF Structure

```
typedef struct tagGM_HEAP_DEF{
        void * heapstart; →   START ADDRESS of GP32 HEAP
        void * heapend; → END ADDRESS of GP32 HEAP
}
```

GP32 START-UP Code each time updates HEAPSTART and HEAPEND variables of gpstart.c with HEAP_START_ADDRESS and HEAP_END_ADDRESS. A developer must assign the subject value to GM_HEAP_DEF structure and call gm_heap_init function to initialize GP32 memory library.

● ARM9 – DEDICATED DESCRIPTION

malloc_ex(unsigned int size, int ex_flag, int   unsigned char init_val);

```
#define MALLOC_EX_AUTOFREE            0x1
```

If designated as ex_flag value, free_ex can free all of the allocated memory at once.

```
#define MALLOC_EX_MEMSET              0x2
```

If designated as ex_flag, it is initialized to the value of init_val after calling malloc.

free_ex(void)

Ex_flag of malloc_ex frees all of the allocated memory at once by using MALLOC_EX_AUTOFREE.

make_mem_partition (struct tagGPMEMFUNC * p_mem_sub, unsigned int size, int * err_no)

This function allocates as much memory as its size allows and also initializes the allocated memory so that it may be able to remalloc by using p_mem_sub. Please note that GP32 can service a maximum of two MALLOC routines concurrently.

- Allocating gp_mem_func members

  See section for Getting Started with GPSDK.

- Allocating gp_str_func members

  See section for Getting Started with GPSDK.

GP Graphic Library

Required Library : gpstdlib.alf, gpmem.alf, gpgraphic.alf

Optional Library : gpg_ex01.alf, gpfont.alf, gpgraphic8.alf, gpgraphic16.alf

//##

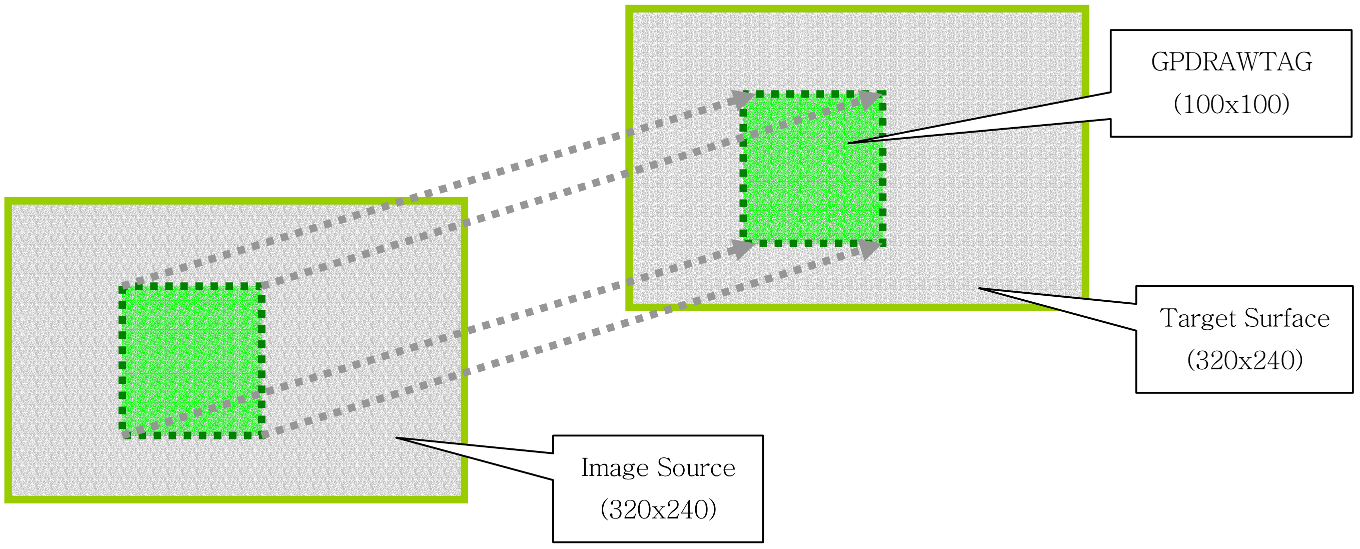
Important Note: The graphic library consists of 8-bit library, 16-bit library and common library.
To use graphic library, include the common library of gpgraphic.alf and choose between gpgraphic8.alf and gpgraphic16.alf depending on the graphic mode.

주의 사항 : graphic library는 8bit용과 16bit용, 그리고 공통 library로 구성되어있다.
Graphic library를 쓰려면 공통 library인 gpgraphic.alf를 포함하고 사용하고자 하는 graphic mode에 따라
Gpgraphic8.alf나 gpgraphic16.alf를 포함시키면 된다.

GPDRAWTAG

Remark : This structure body is used in defining the limit of the target area in the graphic library. If you want to transfer a 320 x 240 image to the designated GPDRAWTAG area at size 100 x 100, you need to auto-clip the related graphic API so that it will not be larger than 50 x 50.

As shown in the diagram below, only the designated area is transferred to GPDRAWTAG when you transfer a 320 x 240 image to a target of the same size.



Member :

int restoreflag : Not used in GPSDK 2.1.0 version. (To be used only when drawing in invalidated area.)

short clip_x : x coordinates for left-top of clipping area

short clip_y : y coordinates for left-top of clipping area

short clip_w : width of clipping area

short clip_h : height of clipping area

GPDRAWSURFACE

Remark : This is used to define target in the graphic library.

GPDRAWSURFACE is mostly used to define the buffer transferred to LDC, but is also possible to define the memory area where LCD is not transferred or the image data to surface.

This is generally created by using the GpSurfaceCrate function, but you can also write in each member directly.

Member :

unsigned char * ptbuffer : Pointer for memory area indicating the surface.

If ptbuffer is the LCD transfer buffer, it must not be in the middle of 8MB SDRAM.

(X) ptbuffer < 0x3E8000 AND (ptbuffer + 76800) > 0x3E8000 (at 8bit)

(X) (surface1.ptbuffer + 76800) < 0x3E8000 AND (surface2.ptbuffer > 0x3E8000) (at 8bit)

int bufflag : This value is reserved at GPSDK 2.1.0 as GPC_GDFLAG_FLIPBUF.

int buf_w : Width pixel of surface.

int buf_h : Height pixel of surface.

int ox : Offset coordinates to transfer the surface elsewhere or receive transfer at the surface.

int oy : Offset coordinates to transfer the surface elsewhere or receive transfer at the surface.

unsigned char * o_buffer : This function defines the original value for malloc allocation, in case that ptbuffer is allocated with malloc. In order to free the allocated memory, you must free the address of o_buffer.

LCD Control functions :

Overview : This function is designed to register the buffer for transmitting LCD on GP32. GP32 contains the following APIs for controlling LCD.

GpSurfaceSet	➔ This function registers the buffer for transmitting LCD on GP32.
GpLcdEnable	➔ This function activates the video signal toward LCD (call this function after GpSurfaceSet to reduce flicking).
GpLcdDisable	➔ This function disables the video signal toward LCD (Important Note: The continuation of this status during long hours may greatly reduce the lifetime of LCD).
GpSurfaceFlip	➔ This function replaces the existing buffer for transmitting LCD with a new one (synchronous replacement)

int GpLcdStatusGet(void);

This function returns the value, which represents the current state of LCD. It can return the bitwise OR values of the following values.

```
#define GPC_LCD_ON_BIT          0x80
#define GPC_LCD_VACTIVE         0x40
#define GPC_LCD_HACTIVE         0x20
```

int GpGraphicModeSet(int gd_bpp, int * gp_pal);

This function is used to set the graphic mode for the system.

int gd_bpp : This function designates the bit per pixel value, either 8 or 16.

int * gp_pal : This is a pointer to the palette array. It is valid only if gd_bpp value is 8. If NULL, a basic palette in the system should be used.

Return Value: Returns the number of LCD surface buffers generated by firmware.

int GpLcdSurfaceGet(GPDRAWSURFACE * ptgpds, int idx);

This function returns the screen surface buffer generated and managed by GP32 firmware. In 8-bit mode, it returns a maximum of 4 surface buffers. In 16-bit mode, a maximum of two surface buffers. Please keep in mind that only the surface buffer returned by GpLcdSurfaceGet can become the primary buffer of the actual LCD.

int GpMemSurfaceGet(GPDRAWSURFACE * ptgpds);

While GpLcdSurfaceGet returns the surface buffers previously generated by firmware, GpMemSurfaceGet allocates surface buffers in system memory. Therefore this function enables unlimited generation of surface buffers as the system memory permits. Please be advised that the surface buffer generated by GpMemSurfaceGet cannot become the primary buffer of the actual LCD.

void GpLcdInfoGet(GPLCDINFO * p_info);

This function returns the current status of LCD.

```
typedef struct tagGPLCDINFO{
                union{
                        int U32_lcd;
                        struct{
                                char clk_value;         //LCD frame rate factor (frame rate = (cpu clock / ((clk_value+ 1)*320)))
                                char lcd_buf_count;     //The number of LCD surface buffers generated and managed by firmware
                                char bpp;             //The current LCD bit
                                char b_lcd_on;          //Whether the LCD panel is turned on or not.
                        }U8_lcd;
                }lcd_global;
                unsigned int buffer_size;          //The current LCD buffer size
                unsigned int * buf_addr[4];             //The address of LCD surface buffer generated and managed by firmware
                unsigned int * r_palette;       //The address of hardware palette
                unsigned int * m_palette;       //The address of software palette
        }GPLCDINFO;
```

PALETTE Operations:

These functions can be used only in 8-bit mode.
8bit mode일 경우에만 사용가능하다. //##

Type define :

typedef unsigned short * GP_PALETTEENTRY;

➔ This function defines a pointer to the palette array with 16-bit resolution.

```
typedef struct tagGP_LOGPALENTRY{
    unsigned char peRed;
    unsigned char peGreen;
    unsigned char peBlue;
    unsigned char peFlags;
}GP_LOGPALENTRY;
```

This function is identical to the PALETTEENTRY structure in the WIN32 API. It is used when the PALETTEENTRY data is applied to GP32.

typedef unsigned int * GP_HPALETTE;

This function defines the palette handle previously defined in GP32.

GP_HPALETTE GpPaletteCreate(int entry_num, GP_PALETTEENTRY * pal_entry);
GP_HPALETTE GpPaletteCreateEx(int entry_num, GP_LOGPALENTRY * pal_entry);

entry_num : This function designates the number of paletter entry. If the total number is smaller than 256, the rest will be filled with 0.

pal_entry : 16bit (red 5, green 5, blue 5, intensity 1) An array of palette entry data

This function allocates 1 KB of palette area to the system memory and it returns the address. If failed, it returns NULL.

void GpPaletteSelect(GP_HPALETTE h_new);

h_new : Palette handle to register as a new software palette

return : Palette handle of existing software

unsigned int GpPaletteRealize(void);

This function copies software palette data to hardware palette, when the palette values are finally reflected in the LCD.

Return value is always 1.

int GpPaletteDelete(GP_HPALETTE hPal);

This function removes the palette area that corresponds to hPal from the system memory.

If the removal is successful, it returns 1. If hPal is already NULL, it returns 0.

정상적으로 제거되면 1을 리턴. 이미 hPal이 NULL이면 0을 리턴한다. //##

GP_HPALETTE GpRegPalGet(void);

This function returns the address of hardware palette.

GP_HPALETTE GpLogPalGet(void);

This function returns the address of software palette.

int GpPaletteEntryChange(int pal_offset, int entry_num, GP_PALETTEENTRY * pal_entry, int flag);

int GpPaletteEntryChangeEx(int pal_offset, int entry_num, GP_LOGPALENTRY * pal_entry, int flag);

pal_offset : The index of the first palette entry you plan to change

entry_num : The number of entry you plan to change

pal_entry : The data array of entry you plan to change

flag 0 : This function reflects the entry changes on hardware palette. Despite slow speed, no noise occurs while the entry changes are made to the screen.

1 : This function reflects the entry changes on hardware palette. Despite high speed, noise may occur while the entry changes are made to the screen.

2 : This function reflects the entry changes on but software palette, not hardware palette.

Return: If successful, this function returns 1 and 0 if an error occurs.

정상적으로 적용되면 1, 오류발생시 0 //##

int GpLcdFade(int fade_step, GP_HPALETTE old_pal); //old_pal – optional. default is NULL

Remark : Based on fade_step value, it either fades in or fades out.

Argument :

fade_step : If positive number, it fades in by the fade_step value and if negative number, it fades out by the fade_step value.

old_pal : It is optional and default is NULL.

Return : If fade is performed normally, this function returns 1 and 0 if fade_step is 0 or fade is completed.

정상적으로 fade시 1을 리턴, fade_step이 0이거나 더 이상 fade가 끝까지 수행되었을 때 0을 리턴 //##

void GpLcdNoFade(GP_HPALETTE old_pal);

Remark : This function returns LCD to normal condition.

Argument :

old_pal : This is identical to the one for GpLodFade.

int GpLcdChanFade(int fade_step, int chan, GP_HPALETTE old_pal);

Remark : A specific color fades in or fades out.

Argument :

fade_step : If positive number, it fades in by the fade_step value and if negative number, it fades out by the fade_step value.

chan : GPC_PALCHAN_RED_BIT : Only the red color fades

GPC_PALCHAN_GREEN_BIT : Only the green color fades.

GPC_PALCHAN_BLUE_BIT : Only the blue color fades.

old_pal : This is identical to the one for GpLodFade.

int GpLcdFadeNormalize(GP_HPALETTE basic_pal);

Remark : This function changes system palette into hardware palette gradually.

GpLcdFade를 통하여 Fade를 진행한 후 다시 원상태로 복귀할 때 사용한다.

Return : This function returns the number of palette entry that has been modified. It returns 0 if the palette is normalized.
변화된 Palette의 엔트리의 수를 리턴한다. 따라서 리턴값이 0이면 원상태로 복귀한것이다. //##

Blitting Operations:

: These functions can be divided into 8-bit mode API and 16-bit mode API.
8bit mode와 16bit mode API로 나누어져 있다. //##

*8bit API

Function define :

int GpBitBlt (GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx, int dy, int width, int height,unsigned char * src, int sx,
int sy, int imgw,int imgh)

Remark : Simply transfers data area (image) designated as src to a target surface designated as GPDRAWSURFACE.
For reference, the coordinates of GPSDK's LCD origin is (0, 0) for left-top and (320, 240) for right-bottom.
※ GPSDK Graphic Basic API

Argument :

gptag	: (See section on GPDRAWTAG) Designates clipping area. If gptag is NULL, the clipping area will be the whole area the target surface.
ptgpds	: (See section on GPDRAWSURFACE) This is the structure body factor designating the target area to receive data (image). This is the background image when copying the image.
dx	: Starting x coordinates for target surface where transferred data (image) will be drawn.
dy	: Starting y coordinates for target surface where transferred data (image) will be drawn.

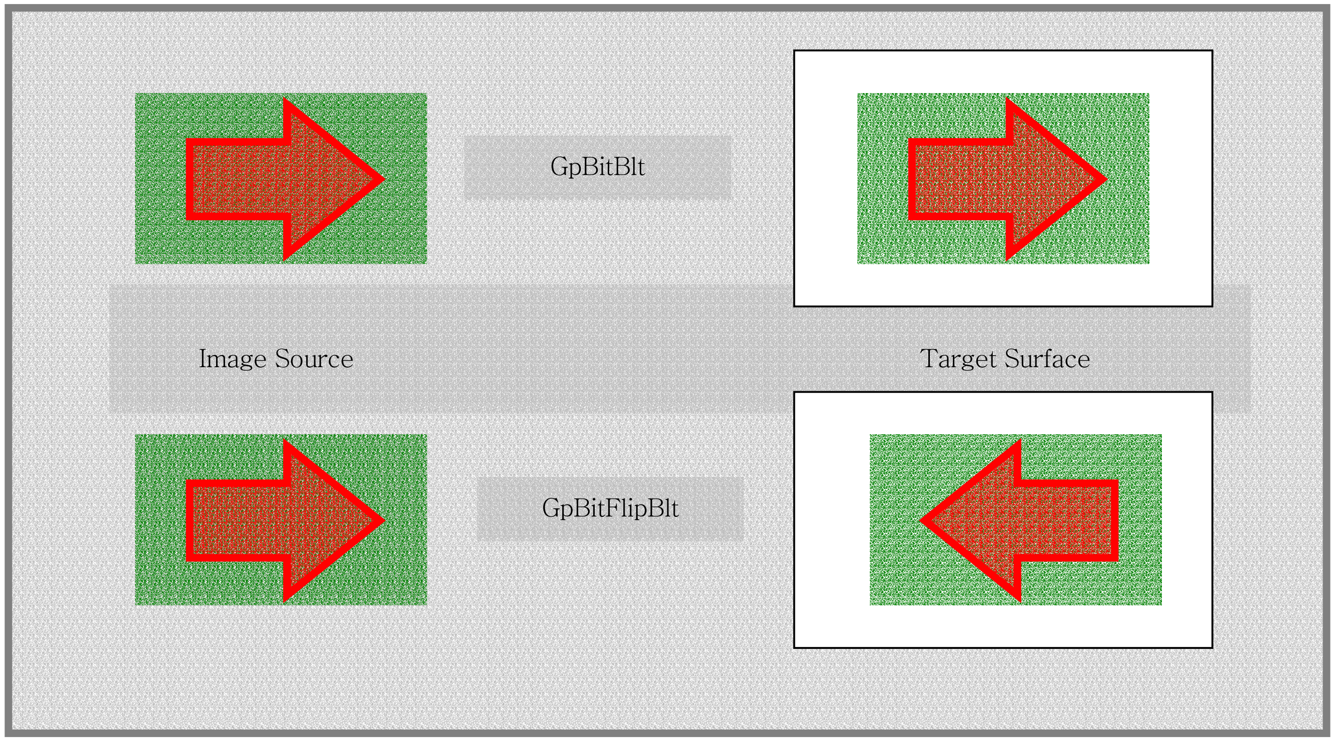
width	: Width of the area where data (image) will be transferred. (width of drawing area)
height	: Height of the area where data (image) will be transferred. (height of drawing area)
src	: Source of transferring image.
sx	: Offset x coordinates to start transferring from image source.
sy	: Offset y coordinates to start transferring from image source.
imgw	: Width of original image source.
imgh	: Height of original image source.

Return : GPC_GPDRAW_OK ➔ Return value when blitting has been properly carried out.

GPC_GPDRAW_ERR ➔ Return value when blitting is unnecessary as you have missed the clipping area.

int GpBitLRBlt (GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx, int dy, int width, int height,unsigned char * src,int sx, int sy, int imgw,int imgh)

Remark : Transfers to target source by mirroring the right and left of transferring data (image) source.



Remark : Transfers to target source by mirroring the right and left of transferring data (image) source.

```
int GpBitUDBlt ( GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char * src,int sx,
                    int sy,int imgw,int imgh);
```

This function transfers to target source by mirroring the top and bottom of transferring data (image) source.

```
int GpTransBlt  (GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char *src,int sx,
                  int sy,int imgw,int imgh,unsigned char color);
int GpTransLRBlt (GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char* src,int sx,
                    int sy,int imgw,int imgh,unsigned char color);
int GpTransUDBlt (GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char* src,int sx,
                      int sy,int imgw,int imgh,unsigned char color);
```

Remark : Just as GpBitBlt does, this function transfers (copies) data source (image) defined as src to target surface, but doesn't transfer the same color as the one designated by source. Consequently, it draws a character on the backgrond image except for transparent color.

GpTransLRBlt transfers to target by mirroring the right and left of data source (image) unlike GpTransBlt and GpTransUDBlt transfers to target by mirroring the top and bottom of data source (image).

```
int GpPointSet ( GPDRAWSURFACE * ptgpds, int x, int y, unsigned char color )
void GpLineDraw ( GPDRAWSURFACE * ptgpds,int xS,int yS,int xE,int yE,unsigned char color )
void GpRectDraw ( GPDRAWSURFACE * ptgpds, int left, int top, int right, int bottom, unsigned char color )
```

int GpRectFill (GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int dx, int dy, int width, int height, unsigned char color)
void GpEllipseDraw (GPDRAWSURFACE * ptgpds, int xS, int yS, int w, int h, unsigned char color)

Overview : These are basic APIs of GPSDK 2.1.0 for dots, lines, squares, and ellipses..

GpPointSet	➔ dots
GpLineDrarw	➔ lines
GpRectDraw	➔ uncolored squares
GpRectFill	➔ colored squares
GpEllipseDraw	➔ uncolored ellipses

* 16bit API

: The 16-bit API is basically identical with the 8-bit API in terms of its usage.
기본적으로 사용법은 8bit API와 동일하다. //##

int GpBitBlt16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char * src, int sx,int sy,int imgw,
int imgh)
int GpTransBlt16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char *src,int sx,int sy,int imgw,
int imgh,int color);
int GpBitLRBlt16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char * src,int sx,int sy,int imgw,
int imgh);
int GpTransLRBlt16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char* src,int sx,int sy,
int imgw,int imgh,int color);
int GpBitUDBlt16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char * src,int sx,int sy,
int imgw,int imgh);

```
int GpTransUDBlt16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,unsigned char* src,int sx,int sy,
                    int imgw,int imgh,int color);
int GpRectFill16(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,int width,int height,int color);

/*basic draw api*/
void GpPointSet16(GPDRAWSURFACE * ptgpds,int x, int y, int color);
void GpLineDraw16(GPDRAWSURFACE * ptgpds,int xS,int yS,int xE,int yE,int color);
void GpRectDraw16(GPDRAWSURFACE * ptgpds,int left,int top,int right,int bottom,int color);
void GpEllipseDraw16(GPDRAWSURFACE * ptgpds,int xS,int yS,int w,int h,int color);
```

GP Sound Library

Required Library : gpstdlib.alf, gpmm.alf, gpsound.alf

※ GP32 uses Software MIDI, which will be available in the next version.

void GpMidiPlay (unsigned char * midisrc, int repeatcount)
void GpMidiListPlay (unsigned char ** srclist, int listcount)
void GpMidiStop (void)

Overview : GP32 adopted general MIDI sound source chip. All the general specs follow the standard specs of general MIDI. However, as GP32 did not implement a separate MIDI chip for H/W driver, a GP32 dedicated MIDI sequencing engine was needed. The GP Sound Library's MIDI output acts as one but this prevents the original functions of a MIDI file. You must use Development Utility 2.1.0 and export the dedicatedly formatted data.

Remark : MIDI sound cannot be mixed and only one sound source is output simultaneously. Thus when you call GpMidiPlay, the currently outputting MIDI sound will be interrupted.

You can adjust the number of play with GpMidiPlay's repeatcount factor, but the GpMidiListPlay only automatically repeats the numbers on the list in rotation.

When you call GpMidiStop, the currently outputting MIDI sound will be interrupted and starts initialization.

Argument :

GpMidiPlay

unsigned char * midisrc : Pointer for media data.

int repeatcount : Number of repeat. Repeats automatically when the value is 0.

GpMidiListPlay

unsigned char **srclist : Pointer for the list of media data pointers.

int listcount : Number of lists.

Header : gpmm.h

Related Functions : GpMidiStop, GpMidiPause, GpMidiReplay, GpMidiStatusGet

void GpMidiPause (void)
void GpMidiReplay (void)

Remark : Pauses the currently playing MIDI sound or continues play. If you call GpMidiPlay and GpMidiListPlay at pause, all data on the paused MIDI sound will be deleted.

Header : gpmm.h

Related Functions : GpMidiPlay, GpMidiListPlay, GpMidiStop, GpMidiStatusGet

int GpMidiStatusGet (int * played)

Remark : The state of GP32 MIDI sequencing engine is defined as below.

GPC_MIDISTATUS_BUSY : MIDI device is in the middle of processing MIDI event.
GPC_MIDISTATUS_READY : MIDI device is not in use and ready to output new MIDI sound.
GPC_MIDISTATUS_PAUSED : MIDI device is paused with GpMidiPause.

Argument :

int * played : Meaningful only when outputting medium through the MIDI device or GpMidiPlay.
Currently processing MIDI source data transfers the values at start position in bytes.

Return : Returns one of GPC_MIDISTATUS_BUSY, GPC_MIDISTATUS_READY, and GPC_MIDISTATUS_PAUSED.

Header : gpmm.h
Related Functions: GpMidiPlay, GpMidiListPlay, GpMidiStop, GpMidiPaused, GpMidiReplay

int GpPcmInit (PCM_SR sr, PCM_BIT bit_count)

Overview : GP32 mixes and outputs up to 4 PCM (Pulse Code Modulation) channels. In order to maximize the processor's computing performance while supporting mixing, the sampling rate of PCM data you can use simultaneously is limited to one. While all PCM data's bit per sample is fixed to 16bit and channel count simultaneously supports mono and stereo. In case of mono, however, same data is output from both left and right channels.

Notice : PCM sample rate is closely connected to the processor clock speed, so you must refer to the reference table below.

Return : The sample rate value actually processed by the device based on the present set processor clock speed. As the difference between this value and the PCM sample rate becomes larger, the distortion of sound also increases.

Remark : GpPcmInit receives the PCM_SR Enum variables and initializes GP32 PCM sound output engine's channel, sample rate, bit per sample values. You need to call GpPcmInit and initialize, and then output only the data suitable for the initialized PCM data. If you want to output other kinds of PCM data, stop all PCM sound output and recall GpPcmInit. (If you simultaneously output PCM sounds with different sample rates, one sound will be distorted.)

```
typedef enum {
    PCM_M11,        ➔ Mono 11,160Hz PCM
    PCM_S11,        ➔ Stereo 11,160Hz PCM
    PCM_M22,        ➔ Mono 22,321Hz PCM
    PCM_S22,        ➔ Stereo 22,321Hz PCM
    PCM_M44,        ➔ Mono 39,062Hz PCM
    PCM_S44,        ➔ Mono 39,062Hz PCM
}PCM_SR;
```

```
typedef enum{
        PCM_8BIT,        ➔ 8Bit
        PCM_16BIT        ➔ 16Bit
}PCM_BIT;
```

int GpPcmEnvGet(PCM_SR * p_sr, PCM_BIT * p_bit_count, int * p_real_sr);

Remark : This function returns the current environment of PCM.

int GpPcmPlay (unsigned short * src, int size, int repeatflag)

Remark : GpPcmPlay adds data source to GP32 PCM mixing engine. Up to 4 sources can be mixed.
PCM data source must have the same sound form as that initialized at GpPcmInit.

Return :

GPC_EPCM_OK ➔ Data source has been normally added to mixing engine.

GPC_EPCM_FULL ➔ 4 PCM sources are being output on the present mixing engine and no more can be added.

Argument :

unsigned short * src : PCM data source.

int size : Size of PCM data source. (in byte)

int repeatflag : It can output only one at 0 and automatically repeats at 1.

void GpPcmRemove (unsigned short * src)

void GpPcmStop (void)

Remark : GpPcmRemove removes certain sound sources from the mixing engine and GpPcmStop removes all sound sources..

Argument :

GpPcmRemove

unsigned short * src : Sound source you wish to remove from the mixing engine.

GP Font Library

Files :

gplib₩gpfont.alf, gpfont8.alf,gpfont16.alf

gpinclude₩gpfont.h

target₩gpfont_port.h
target₩gpstart.c

Required Library : gpstdlib.alf, gpmem.alf, gpgraphic.alf, gpfont.alf

Optional Library : gpg_ex01.alf, gpstdio.alf, gpfont8.alf,gpfont16.alf

Important Note: The font library consists of 8-bit font library, 16-bit font library and common library. To use font library, include the common library of gpfont.alf and choose between gpfont8.alf and gpfont16.alf depending on the graphic mode.

주의 사항 : Font library는 8bit용과 16bit용, 그리고 공통 library로 구성되어있다.
Font library를 쓰려면 공통 library인 gpfont.alf를 포함하고 사용하고자 하는 graphic mode에 따라
Gpfont8.alf나 gpfont16.alf를 포함시키면 된다. //##

void GpFontInit (BGFONTINFO * ptr)

Overview : GP32 basic language is English and each country can add one language. In other words, GP32 basically supports 1~2 languages. You would need to register the code table, font data, and resource data for each language.

GP32 font resource consists of 1bit per dot Bitmap Font. Developers can register freely customized Font resource.

GP32 start-up code calls GpFontInit through the constants on ₩target₩initval_port.h and initializes the language and its system font.

For reference, GPSDK 2.1.0 has both Korean and English as basic languages and the code for Korean is KS2601.

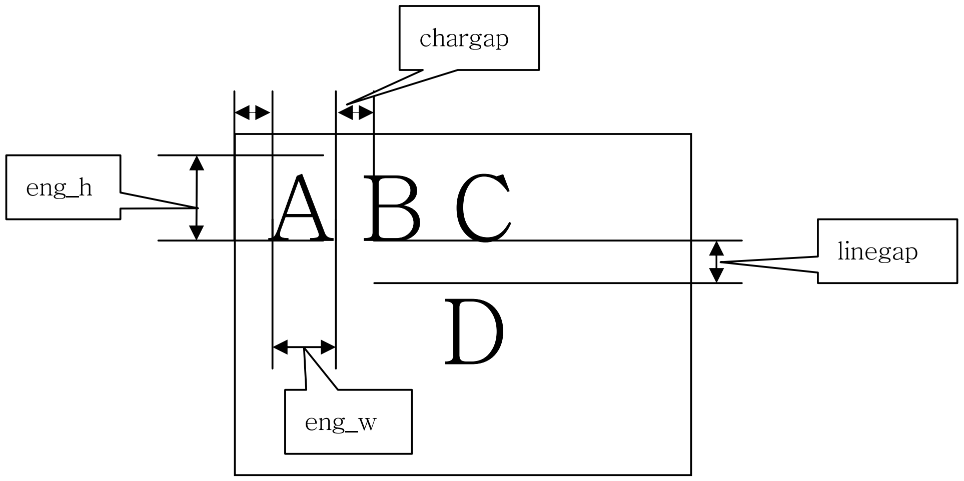
Remark : Transfers font data through BGFONTINFO structure body. **GpFontInit function must be called together with GpFontResSet function.**

You may want to use an English font using over font size 2 on an application. Whenever you want to use a new font, you must recall GpFontInit .In this case, you cannot output simultaneously over font size 2.

See GpTextOutEx section of this manual when you want to use various font types together with the system font.

BGFONTINFO :	/* below is based on GPSDK 2.1.0 Korean Version */
int kor_w	: Width of Korean font characters in pixel.
int kor_h	: Height of Korean font characters in pixel.
int eng_w	: Width of English characters in pixel.
int eng_h	: Height of English font characters in pixel.
int chargap	: Gap between characters when outputting character string. Inputs values in percentages.
int linegap	: Gap between lines when outputting character string. Inputs values in percentages

Ex)



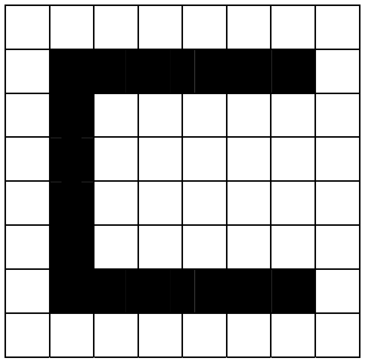
Header : gpfont.h

Related Functions : GpFontResSet, GpKorFontResGet, GpEngFontResGet, GpSysFontGet

void GpFontResSet (unsigned char * p_kor, unsigned char * p_eng)

Overview : GP32 font resource has Bitmap font as its basic value and 1dot is represented by 1bit. Here is an example.

8 * 8 Font



00000000 (b) → 0x00
01111110 (b) → 0x7E
01000000 (b) → 0x40
01000000 (b) → 0x40
01000000 (b) → 0x40
01000000 (b) → 0x40
01111110 (b) → 0x7E
00000000 (b) → 0x00

The above font data can be converted to GP32 font resource through GP Development Utility V10.

For reference, font resource proposes the following.

➔ The number of pixel (bit) of horizontal and vertical line cannot be smaller than 8.

➔ Even though the number of pixel of horizontal line is not a multiple of 8, the binary data must pad 0 so that it is a 8의 8수.

STEP BY STEP : The steps to register the new font is as follows.

STEP 0> Prepare Bitmap font.

STEP 1> Export as GP data from font utility in GP Development Utility V10. (text version)

STEP 2> Copy the output file to target₩gpfontres.dat. (delete existing data)

STEP 3> Change target₩gpfont_port.h options to agree with font data. Present set value is shown below.

```
#define KORFONT_W          16      // pixel
```

```
#define KORFONT_H          16      // pixel
#define ENGFONT_W          8       // pixel
#define ENGFONT_H          16      // pixel
#define FONT_CHARGAP       10      // percentage
#define FONT_LINEGAP       10      // percentage
```

STEP 4> Modify the red part below in InitializeFont() of target₩gpstart.c.

```
void InitializeFont()
{
    …..
    GpFontResSet((unsigned char*)fontresKor, (unsigned char*)fontresEng);
}
```

Header : gpfont.h

Related Functions : GpFontInit, GpKorFontResGet, GpEngFontResGet, GpSysFontGet

unsigned char * GpKorFontResGet (void)
unsigned char * GpEngFontResGet (void)

Remark : Call to access the font resource registered in the present system font during program operation.
Return value is the address of font resource data arrangement. The size for Korean is as follows.

Size of resource data = kor_w * (((kor_h + 7) & ~7) >> 3)

Header : gpfont.h

Related Function : GpFontInit, GpFontResSet, GpSysFontGet

void GpSysFontGet (BGFONTINFO * ptr)

Remark : Sytem font data registered in GpFontInit function is transferred through BGFONTINFO structure body.

Caution : The unit of chargap and linegap in BGFONTINFO is different to those when calling GpFontInit.

➔ In percentage unit when calling GpFontInit.

➔The unit of value received through GpSysFontGet is in pixels.

Ex > Value transferred when calling GpFontInit	Value received in GpSysFontGet
BGFONTINFO.kor_w = 16	BGFONTINFO.kor_w = 16
BGFONTINFO.kor_h = 16	BGFONTINFO.kor_h = 16
BGFONTINFO.eng_w = 8	BGFONTINFO.eng_w = 8
BGFONTINFO.eng_h = 16	BGFONTINFO.eng_h = 16
BGFONTINFO.chargap = 20	BGFONTINFO.chargap = (16 * 20) / 100 = 3
BGFONTINFO.linegap = 20	BGFONTINFO.linegap = (16 * 20) / 100 = 3

Header : gpfont.h

Related Functions : GpFontInit, GpFontResSet, GpKorFontResGet, GpEngFontResGet

int GpTextLenGet (const char * str)
int GpTextWidthGet (const char * lpsz)
int GpTextHeightGet (const char * lpsz)

Remark :

GpTextLenGet :

Returns the length of character string, Counts 1 Korean character by and does not count LF('₩n') and CR('₩r').

GpTextWidthGet :

Returns the widest width and highest height when outputting character string.

GpTextHeightGet :

Changes the line when LF ('₩n') or CR ('₩r') is present in the character string. The character string must end with a null character ('₩0').

Header : gpfont.h

Related Functions : GpTextOut, GpAsciiOut, GpTextNOut, GpTextDraw, GpTextOutEx

int GpCustTextOut(GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y, EXT_FONT * y_font)

Remark : character string output format is the same as GpTextOut function but it is characterized by the use of user-designated language and font instead of system language or font. It designates language code map, resource data, font size data, and output effect through EXT_FONT.

Notice : You must import gpg_ex01.alf to use this function.

Argument :

gptag	: See Graphic.

ptgpds	: See Graphic.
x	: Horizontal output starting position.
y	: Vertical output starting position.
EXT_FONT	: Data structure body for extensive user's designated font.

EXT_FONT

int fx_flag	: Designated effect when outputting. GPC_GDFX_COPY : Fill the background of character string with monochrome color designated at color2 and output color designated at color1 as the color of character string. GPC_GDFX_TRANS : Character string is output in the color designated at color1, but the rest of the area is in transparent color. GPC_GDFX_TRANS | GPC_GDFX_RAYPLUS : Areas other than the characters are transparent, but color of characters designated at color1 is synthesized with background color. (ray effect)
unsigned char * lpsz	: Outputting character string. (CAUTION : The character string follows user-designated codemap.)
unsigned char * pBmFont	: Pointer for user-designated font resource data.
int ex_font_w	: Width per 1 character in user-designated font.
int ex_font_h	: Height per 1 character in user-designated font.
int ex_chargap	: Gap between characters when outputting user-designated font character string. (unit in pixels)
int ex_linegap	: Gap between lines when outputting user-designated font character string. (unit in pixels)
int color1	: Foreground color.
int color2	: Background color. (Viable only when fx_flag is GPC_GDFX_COPY.)

The following API is divided into two parts, which are 8-bit API and 16-bit API.
♦아래의 API들은 graphic mode에 따라 8bit용과 16bit용으로 나누어진다. //##

* 8-bit API

```
void GpTextOut ( GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y,
              unsigned char * sour, unsigned char color )
void GpCharOut(GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y,
                char * sour, unsigned char color);
```

Remark : Output character string in area defined by GPDRAWSURFACE. Automatically calculate and clip GPDRAWTAG area, chracter string output starting position (x, y), and width and height of output character string.
When escape characters LF (‘₩n’) or CR (‘₩r’) are present in character string, it changes lines.
When LF and CR are one after the other, it changes the lines twice.

Argument :

gptag	: GPDRAWTAG structure body. (See Graphic)
ptgpds	: GPDRAWSURFACE structure body. (See Graphic)
x	: Horizontal position at the start of character string output.
y	: Vertical position at the start of character string output
sour	: Character string (ends with a null character)
color	: Foreground and GP32 color values. (8 bit)

Header : gpfont.h

Related Functions : GpTextWidthGet, GpTextHeightGet, GpAsciiOut, GpTextNOut, GpTextDraw, GpTextOutEx

```
void GpTextNOut ( GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y,
                  unsigned char * sour, int nStart, int nString, unsigned char color )
```

Remark : GpTextOut repeats output until there is a null character in the character string or it strays from the output area, but this function repeats output until there is a null character in the character string or a certain number of characters in certain position of designated character string is output.

Argument :

nStart	: Position value of outputting character string. zero base
nCount	: Length of character string reading from nStart and output. (2byte code is also considered as 1 character.)

Header : gpfont.h

Related Functions : GpTextWidthGet, GpTextHeightGet, GpTextOut, GpAsciiOut, GpTextDraw, GpTextOtEx

* 16-bit API

: The 16-bit API is basically identical with the 8-bit API in terms of its usage.
기본적으로 8bit용과 사용법이 동일하다.

```
void GpTextOut16(GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y, char * sour, int color);
void GpCharOut16(GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y, char * sour, int color);
void GpTextNOut16(GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y, char * sour, int nStart, int nString, int color);
```

GP File I/O Library

Required Library : gpstdlib.alf, gpmem.alf, gpstdio.alf

Optional Library :

Overview :

GP32 uses Smart Media Card for external storage medium. Since SMC has a file system in support for FAT12 and FAT16 formats, it boasts its high compatibility with many other platforms including GP32.
File I/O library refers to APIs that may read and write data on SMC in FAT12 and FAT16 formats.
The return values of File I/O API are the enumeration values defined as ERR_CODE and they return SM_OK when APIs are normally implemented.

Overview : GP32 uses Smart Media Card for storing. SMC's file system is compatible on platforms other than GP32 with FAT12 or FAT16.
File I/O library has APIs which reads and writes data on SMC in FAT12 and FAT16 format.
Return values of File I/O APIs are enumeration values defined as ERR_CODE. When APIs are carried out normally, they return SM_OK.

ERR_CODE GpFatInit (void)

Remark : To access SMC on GP32, you need to initialize FAT first. GP32 start-up code does not conduct the necessary initialization. If the developer is developing an access program for SMC, call GpFatInit and then use other File I/O functions.

Header : gp--stdio.h

Return : Always returns SM_OK.

ERR_CODE GpRelativePathSet (const char * p_path)
void GpRelativePathGet (char * p_path)

Remark : To use file system path as relative path, set the standard path. Once designated, a path can be changed only by calling GpRelativePathSet. Otherwise it will remain valid unless deleted by calling GpFatInit and GpFormat. In addition, if the path which designates a standard path and sends to argument has a drive name, it is recognized as the absolute path.

GP32 Path Designation Rule :

Volume name :₩ Directory name ₩ … ₩ File name.extender

1. Volume name ➔ 'gp' (GP32 uses only one drive in the name `gp'.)
2. Directory name ➔ Cannot be more than 8Byte.
3. '₩' ➔ Distinguishes volume and directory, directory and directory, and directory and file.
4. File name ➔ Cannot be more than 8Byte.
5. '.' ➔ File must distinguish the extender with '.'.
6. Extender ➔ Fixed at 3Byte.

Maximum path length is 256Byte.

Argument :

const char * p_path : Full path which will be the standard path when using relative path.

Return : SM_OK ➔ Carried out successfully.
ERR_NOT_FOUND ➔ The is no :(colon) after gp in p_path argument.
ERR_OUT_OF_MEMORY ➔ Length of p_path is more than 244 characters.

ERR_CODE GpFileCreate (const char * p_file_name, ulong fcreate_mode, F_HANDLE *p_handle)

Remark : Creates a designated file at p_file_name and sends the file handle.

Return: Returns SM_OK if the file has been successfully created.

Argument :

const char * p_file_name : Name of the new file. If you have called GpRelativePathSet, it can be used as absolute path or relative path. If not, you must use the absolute path including `gp:'.

ulong fcreate_mode : NOT_IF_EXIT, ALWAYS_CREATE. When a file of the same name already exists, use the ALWAYS_CREATE mode to delete the existing file.

F_HANDLE * p_handle : Received file handle.

ERR_CODE GpFileOpen(const char * p_file_name, ulong fopen_mode, F_HANDLE *p_handle)

Remark : Finds and opens the file with the name given, and sends the handle value through the argument.
If a file is open at OPEN_W mode and you want to GpFileOpen that file, ERR_FILE_OPENED will be returned. If you open a file at OPEN_R mode, you can reopen it only at OPEN_R mode.
You must always close an opened file by calling GpFileClose.

Return :

Argument :
const char * p_file_name : Name of the file you want to open.
ulong fopen_mode : Bitwise OR combination is possible as both OPEN_R (read) and OPEN_W (write) are available.
F_HANDLE * p_handle : Variable which has received the handle of the open file

ERR_CODE GpFileRead (F_HANDLE h_file, void * p_buf, ulong buf_size, ulong * p_read_count)

Remark : Reads the contents of the open file from the current position and increases by the amount read. The size of the contents actually read will be returned through p_read_count. SM_OK is retuned if it has been carried out successfully. Although ERR_EOF is returned, the size returned through p_read_count is the amount read normally.

Return : ERR_EOF ➔ File is read to the end.
ERR_FILE_NOT_OPENED : The file of h_file is not open.

Argument :

F_HANDLE h_file : Handle of files opened using GpFileCreate or GpFileOpen.
void * p_buf : Pointer of the buffer storing the contents read.
ulong buf_size : Size of the contents read.
ulong * p_read_count : Size of the contents actually read.

ERR_CODE GpFileWrite (F_HANDLE h_file, const void * p_buf, ulong count)

Remark : Records the contents of the open file from the current position and increases by the amount recorded. An error code is returned when errors such as memory shortage occurs while using the file. The contents written before will remain on the disk. When you want to delete the file with error, use GpFileClose to close the file and delete with GpFileRemove function.

※ Notice : If you do not call GpFileClose after using the files, the FAT table will not be updated.

Return :

Argument :

F_HANDLE h_file : Handles of the files opened using GpFileCreate or GpFileOpen.

const void * p_buf : Pointer of the buffer storing the contents recorded.

ulong count : Size of the contents to being recorded.

ERR_CODE GpFileSeek (F_HANDLE h_file, ulong seek_mode, ulong offset, long * p_old_offset)

Remark : Change the current position of the open file. ERR_EOF is returned if the file you want to transfer is over the limited size and the moves the file from the current position to the end.

Return :

Argument :

F_HANDLE h_file : Handle of the files opened using GpFileCreate or GpFileOPen.

ulong seek_mode : FROM_CURRENT(current present is the standard) / FROM_BEGIN(beginning of file is the standard) / FROM_END(ending of the file is the standard)

ulong offset : Offset of the gap between the current position and the position you are transferring to. If the value is smaller than 0, it will be transferred to the front.

long * p_old_offset : Current position of the existing file. Offset from the former position of the file.

ERR_CODE GpFileClose (F_HANDLE h_file)

Remark : Closes the open file.

Return :

F_HANDLE h_file : Handle of the files opened using GpFileCreate or GpFileOpen.

Argument :

ERR_CODE GpFileRemove (const char * p_file_name)

Remark : Removes the file in the name given.

Return :

Argument :

const char * p_file_name : Name of the file you want to remove.

ERR_CODE GpFileGetSize (const char * p_file_name, ulong * p_size)

Remark : Finds the file in the name given and returns its size through p_size.

Return :

Argument :

const char * p_file_name	: Name of the file.
ulong * p_size	: Size of the file (units in Byte)

ERR_CODE GpFileExtend (F_HANDLE h_file, ulong size)

Remark : Reserves the size of the file in advance. The size in bytes is reserved rather than the size of the current file. The writing speed increases when you reserve the size before writing a large-sized file. This function enables you to enhance the writing speed by reserving clusters in a sequence and erasing the SMC in advance.

Although you use this function to reserve the file size in advance, there will be no change in the actual size of the file. In case you write after calling GpFileExtend, you have to write in the multiple of 512 bytes for the speed to be greatly increased.

Return :

Argument :

F_HANDLE h_file : Handle of the file opened using GpFileCreate or GpFileOpen.

ulong size : Size of the current file plus the size of the file reserved in advance.

ERR_CODE GpFileMove (const char * old_path, const char * new_path)

Remark : Moves file or directory. When old_path and new_path exist on different directories, ERR_FILE_EXIST will be returned if a file or directory with new_path already exists. ERR_FILE_NOT_EXIST will be returned when there is no file or directory with old_path and when there is no upper directory for new_path.

※ Notice : You cannot use a relative path for old_path and new_path!

Return :

Argument :

const char * old_path	: File or directory's existing path. (Must be absolute path.)
const char * new_path	: File or directory's new path. (Must be absolute path.)

ERR_CODE GpFileRename (const char * old_path, const char * new_path)

Remark : Changes the name of file or directory. The file or directory you want to change must have the same upper directory as the existing file or directory. This function may slow down the operation speed, so we recommend not to use this unless necessary.

※ Notice : You can not use a relative path for old_path and new_path!

Return :

Argument :

const char * old_path : File or directory's existing path. (absolute path)

const char * new_path : File or directory's existing path. (absolute path)

ERR_CODE GpDirCreate (const char * p_dir_name, ulong dcreate_mode)

Remark : Create the directory in the name given. The directory will be created only in the last name given at P_dir_name and the upper paths given afore must already exist. In case they do not exist, ERR_INVALID_PARAM is returned.

※ Notice : The name of the directory cannot be more than 8byte.

Return :

Argument :

const char * p_dir_name : The path of the directory you are creating can be used as absolute or relative path.

ulong dcreate_mode : Not used. Enter 0.

ERR_CODE GpDirRemove (const char * p_dir_name, ulong ddel_mode)

Remark : Finds and removes the directory in the name given.

Return :

Argument :

const char * p_dir_name : Path of the directory you want to remove.

ulong ddel_mode : NOT_IF_NOT_EMPTY ➔ Does not remove directory when there are sub directories or files.

ALWAYS_DELETE ➔ Deletes the directory and all the data inside it.

ERR_CODE GpDirEnumNum (const char * p_dir_name, ulong * p_num)

Remark : Returns the number of files in the directory of the name given and its sub directories through p_num. This does not include the subs in the sub directory of the given directory.

Return :

Argument :

const char * p_dir_name : Path of the directory.

ulong * p_num : Number of files and sub directories at p_dir_name.

ERR_CODE GpDirEnumList (const char * p_dir_name, ulong entry_start, ulong entry_count,
GPDIRENTRY * p_list, ulong * p_read_count)

Remark : Returns a list of sub items (files, directories) in the given directory through p_list. ERR_EOF will be returned if the number of items read through entry_count goes over the last item and as many items as in p_read_count will be saved in p_list.

Return :

Argument :

const char * p_dir_name : Path of the directory.

ulong entry_start : Index of items you want to start reading. Zero base.

ulong entry_count : Number of items to read.

GPDIRENTRY * p_list : Pointer for the arrangement to save information on the items read.

ulong * p_read_count : Number of items actually read.

```
typedef struct {
    char name[16];
}GPDIRENTRY;
```

ERR_CODE GpFileAttr (const char * p_name, GPFILEATTR * p_attr)

Remark : Returns data on file or directory in the name given through p_attr.
It returns data such as file sizes and speeds.

Return :

Argument :

const char * p_name : Name of file or directory.

GPFILEATTR * p_attr : Pointer to receive data.

```
typedef struct {
        udword attr;
        udword cluster;
        udword size;
        sTIME time;
}GPFILEATTR;
```

bit	Data allotted to each bit of attr
[31 : 6]	Reserved
[5]	Archive. (1 : modified file, 0 : normal)
[4]	1 : directory, 0 : file
[3]	1 : Volume lable
[2]	1 : system file or system directory
[1]	1 : hidden file or directory
[0]	1 : read only file

ERR_CODE GpFormat (const char * p_vol_name, ulong format_id, ulong * p_bad_block)

Remark : Formats the given volume (always `gp' for GP32). There are two modes, FORMAT_NORMAL and FORMAT_RESCUE.

FORMAT_NORMAL : Formats using the data on existing invalid block. Using this option to format may shorten the time, but as it formats the valid block referring to the data on existing invalid block, it cannot modify any wrong data on the existing invalid block.

FORMAT_RESCUE : Tests ERASE → WRITE → READ on all the sectors, so it creates new data on invalid block and formats accordingly.

In general, it is better to use FORMAT_NORMAL. You can use the FORMAT_RESCUE option if there is an excessive number of invalid blocks detected in the medium or in other necessary situations.

When GpFormat is completed, you will find that previous designated 상대경로 have been deleted, so you would need to recall GpRelativePathSet in order to use it.

※ When you use SMC for the first time after purchase, you must format by using GpFormat. If you don't, all FILE I/O APIs will return ERR_NOT_FORMATTED.

Return :

Argument :

const char * p_vol_name : Always `gp' for GP32.

ulong format_id : FORMAT_NORMAL or FORMAT_RESCUE

ulong * p_bad_block : Number of invalid blocks detected.

ERR_CODE GpNoFATUpdate (const char * p_vol_name)
ERR_CODE GpFATUpdate (const char * p_vol_name)

Remark : You must always use GpNoFATUpdate and GpFATUpdate together.
Before you conduct operations that require updating the FAT table often, call GpNoFATUpdate before working on them. Then call GpFATUpdate. This will improve the speed.
Memory will not be updated in between calling GpNoFATUpdate and GpFATUpdate, thus the FAT table may be saved incorrectly if a power cut occurs.

Return :

Argument :

const char * p_vol_name : Always 'gp'.

GP Network Library

(Refer to Appendix_Multi)

OVERVIEW

TCP,UDP / IP

-. Socket Implementation API

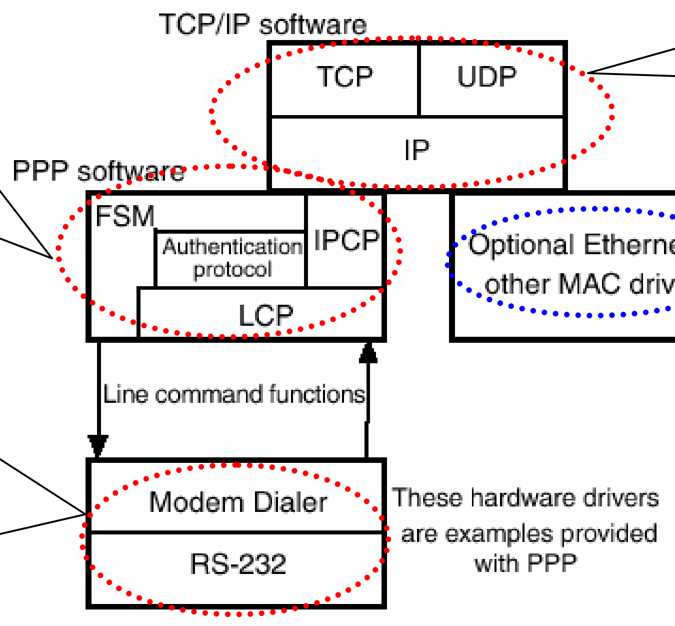
PPP

-.LCP, IPCP (RFC 1661)

-.CHAP, UPAP Autentication

-.Line Management.

(Serial Driver)



Not Used.

UART

Network LIB has implemented RS-232 communication block under gpnet.alf. ‘

RS-232 communication should be directly implemented by UART API.

Figure 1-2 The IP protocol and PPP

Glue Layer

cticks :

GP32 TCP/IP stack requires clock tick for both TCP and ARP. The clock tick refers to the current variable of unsigned long and it is represented as cticks. This value increases 5 to 100 per second, which the operation of TCP and PPP stacks is based on (Please refer to the followings for cticks initialization.).

GP_COMM_OPT : /* PPP Serial Driver Layer */

int baudrate	: It is dependent on both UART transmission speed and communication terminal speed. The baudrate is 19800bps when running on IS95-A and 115200bps on IS95-B.
void (*uart_reset)(int ch)	: It refers to void of GPN_COMM member (*comm._close)(void).
int (*uart_open)(int ch, int baudrate, int en_int)	: It refers to int of GPN_COMM member (*comm._open)(GPN_DESC * p_desc).
void (*uart_sendc)(int ch, unsigned char data)	: It refers to int of GPN_COMM member (*comm._send_one)(unsigned char c_data).
int (*uart_getc)(int ch, unsigned char * data)	: It refers to ini of GPN_COMM member (*comm._recv_one)(unsigned char * p_data)
int (*uart_sendready)(int channel)	: It refers to int of GPN_COMM member (*comm._send_ready)(void).
unsigned int (*get_nettick)(void)	: It refers to GpTickCountGet ().

➔ extern GP_COMM_OPT gp_comm_opt;

➔ A developer must initialize each member of the GP_COMM_OPT structure instance before calling Network API.

GP_COMM_MEM : /* Network LIB의 Heap Manage functions layer */

void* (*malloc)(unsigned int nbyte)	: It refers to gp_mem_func.zimalloc. (Note: The allocated area must be initialized to zero.)
void (*free)(void * pt)	: It refers to gp_mem_func.free.

➔ extern GP_COMM_MEM gp_comm_mem;

➔ A developer must initialize each member of the GP_COMM_MEM structure instance before using Network API.

GP_INET_OPT : /* Communication Information */

char gp_phone_num[MAX_PARAM_STRING]	: Phone number to login
char gp_userid[MAX_PARAM_STRING]	: Authorized User ID when a host requests authentication at login.
char gp_pwd[MAX_PARAM_STRING]	: Authorized Password when a host requests authentication at login.
char gp_fhost[MAX_PARAM_STRING]	: Host address, Domain Name, ip string to login (Example: game.gamepark.com, or 211.192.163.xxx)
int gp_fport	: Host port to login
char gp_modem_init_cmd[MAX_PARAM_STRING]	: Modem Initialization Command (AT Command) Generally “ATZ₩r”
char gp_modem_sub_cmd[MAX_PARAM_STRING]	: Modem Commands (It corresponds to other commands used to configure Dial-Up networking.)
int gp_ppp_tmo	: time out value to be used in PPP Layer (60<<10 is used in 2.1.0.).
int gp_line_tmo	: time out value to be used in Modem Layer (6<<10 is used in BV 10.)
void (*conn_progress)(int p_val)	: Not yet used. Allocates NULL.
unsigned int min_ticker	: Not yet used. Allocates 0.
char gp_dns_addr[MAX_PARAM_STRING]	: Known BUG in Network LIB 2.1.0. When logging into host using your domain name, the address of Domain Name Server must be initialized to ip string (xxx.xxx.xxx.xxx).

The MAX_PARAM_STRING value is 128.

➔ extern GP_INET_OPT gp_inet_opt;

➔ A developer must initialize each member of the GP_INET_OPT structure instance before calling Network API.

Header : gpsockdef.h

int GpNetInit(int net_tps);

Remark : Initializes TCP/IP and PPP stacks by using the values allocated to GP_COMM_OPT, GP_COMM_MEM and GP_INET_OPT structures. In case that the connecting host address is not appropriate, it returns GPC_ENET_INITFAIL and if the login is successful, it returns GPC_ENET_OK. A developer must firstly call GPSDKK Network functions in order to use them.

Note: Network LIB takes up approximately from 50KB up to 100KB of memory for dynamic memory.

cticks Initialization

In addition to the allocation of GP_COMM_OPT, GP_COMM_MEM and GP_INET_OPT structures, a developer must initialize cticks before calling GpNetInit. Initialization refers to timer operation and further details are as follows. (See under GP Standard LIB timer control clause.)

TPS value is defined as 50. The interrupt occurs 50 times per second.

int GpNetParseIp (char * host, unsigned int * m_ip)

Remark :

Transforms either domain name or ip string into unsigned int format. If failed, it returns GPC_ENET_INVALID ARG. In order to get IP by using domain name or ip string, the gp_dns _addr member of GP_INET_OPT structure must be initialized to the right ip string in advance.

Argument :

char * host	: Domain name or ip string
unsigned int * m_ip	: The 32bit address value has the following format. Ip string : 255.254.253.252 ➔ *m_ip = 0xFFFEFDFC

```
long GpSockCreate ( int family, int type, int protocol )
void GpNetTicker(void);
int GpSockConnect ( long so, struct sockaddr_in * s_addr );
int GpSockSend ( long so, char *msg, int len, int flags );
int GpSockSendto ( long so, char *msg, int len, int flags, struct sockaddr_in *s_addr );
int GpSockRecv ( long so, char *msg, int len, int flags );
int GpSockRecvfrom ( long so, char *msg, int len, int flags, struct sockaddr_in *s_addr );
int GpSockClose ( long so );
int GpSockShutdown ( long so, int how );
int GpPPPShutdown ( void );
int GpSockPeerGet ( long so, struct sockaddr_in *peer );
int GpSockNameGet ( long so, struct sockaddr_in *name );
int GpSockOptGet ( long so, int optname, void *optval );
int GpSockOptSet ( long so, int optname, void *optval );
int GpSockConnected( long s);
```

Header : gpnetlib.h

GP OS

Overview :

GP32 uses game-centric OS known as GPOS. GPOS is a compact OS only designed to meet the key requirement of real-time. GPOS garantees the same level of high-performance Real Time OS (RTOS) as Kernels do. Notwithstanding, GPOS generates and manages a maximum of eight threads and it manages four system resources based on criticalsection synchronization.

Four out of eight GPOS threads correspond to system threads and the rest of them refer to user threads. Users threads are used in the format of the TIMER function call to create a timer event such as GpTimerSet, GpTimerKill, GpTimerPause and so forth, which enables an event-driven programming.

GPOS also shines where it is designed for application-intended OS. That is, an application launches OS and has subordinate threads managed by OS, not the way that an application is launched and managed by OS. Therefore, only one application can exist on GP32 and GPOS is capable of multitasking by generating subordinate threads.

The threads defined by GPOS are illustrated as follows:

IDLE Thread : If GPOS has no other executable thread, its idle thread runs. It runs the idle thread as the lowest priority thread.

Primary Thread : It has a function known as GpMain as entry point. It is the main thread of the application. It runs with NORMAL priority. Such a thread can interfere with the normal operation.

Sound Thread : It runs with the highest priority. It manages the sound output with the highest priority.

Net Thread : It has a higher than normal priority by two levels. It manages TCP and IP stacks on GP32.

User Threads : It is complete with four user threads. Each user thread is used in the format of the TIMER function call to create a timer event such as GpTimerSet, GpTimerKill, GpTimerPause and etc. It has a higher than normal priority by one level.

GPOS defines each of the thread handle as follows.

```
typedef enum{
        H_THREAD_SOUND,
        H_THREAD_IDLE,
        H_THREAD_GPMAIN,
        H_THREAD_NET,
        H_THREAD_TMR0,
        H_THREAD_TMR1,
        H_THREAD_TMR2,
        H_THREAD_TMR3
}H_THREAD;
```

Comparison between WIN32 Multi-Threaded programming and User Threads Implementation

The general Win32 Multi-Threaded programming is elaborated as follows:

```
void thread_run(...)
{
    while (TRUE )
    {
                if ( b_exit == TRUE )
                {
                            PostMessage(…);
                }
                Your code….
                Sleep(..);
    }
12:         PostMessage(…);
13:     }
```

When implementing User Threads, only the codes that appear in line# 5 to 9 must be implemented. In other words, the other codes in other lines shall be implemented within GPOS.

GPOS Resources

GPOS manages the following resources.

1. System Memory :	A system call that manages system memory is protected by the Criticalsection.
2. FILE I/O:	A system call relevant to File I/O is protected by the Criticalsection.
3. NET (TCP/IP Stack):	A system call relevant to NET is protected by the Criticalsection.
4. GDI:	A system call relevant to Graphic Device is protected by the Criticalsection.

Note: A software interrupt call is prohibited within an interrupt service routine. A calling convention is not guaranteed for GPOS in case of using an SWI interrupt within an interrupt service routine.

Function Description :

void GpKernelInitialize(void);

Remark : This function is first called by gpstart.c and it initializes all stacks and threads on GPOS. In order to change the stack size of each thread, initval_port.h shall be revised.

```
#define GPMAIN_STACK_SIZE       (100<<10)       //100KB -- access code = 0
#define NET_STACK_SIZE          (64<<10)        //64KB  -- access code = 1
#define USER_STACK_SIZE         (4 << 10)       //4KB   -- access code = 2
```

void GpKernelStart(void);

Remark :

This function is first called by gpstart.c and GPOS starts. GpKernelStart operates a system tick for scheduling service and the thread with the highest priority designated by GpKernelIntialize activates a definite Primary thread.

void GpKernelLock(void);

Remark :

This function aborts the operation of Kernel. It can prevent task switching between threads, but can't avoid task switching with Sound Thread, which is the level-best thread.

void GpKernelUnlock(void);

Remark :

Resume the operation of Kernel meaning the continuation of scheduling service.

void GpThreadSleep(unsigned int delay);

Remark :

This function is identical to Sleep in WIN32 API. The thread that calls GpThreadSleep gets inactivated as much as the factor value (delay*2 milliseconds) and task switching occurs with other thread.

void GpThreadSleep(unsigned int delay);

Remark :

This function is identical to Sleep in WIN32 API. The thread that calls GpThreadSleep gets inactivated as much as the factor value (delay*2 milliseconds) and task switching occurs with other thread.

int GpThreadOptSet(H_THREAD th, int priority, int stk_size); //return ok or err

Remark :

This function changes the thread option for literal string “th”. What can be changed is a priority and a stack size of the thread. A priority is defined as follows. A lower numerical value means higher priority.

```
#define GPOS_PRIO_ABOVE_NORMAL   2
#define GPOS_PRIO_NORMAL         3
#define GPOS_PRIO_BELOW_NORMAL   4
#define GPOS_PRIO_LOW            5
```

H_THREAD GpThreadHandleGet(void);

Remark :

This function returns the current handle value of thread.

int GpPredefinedStackGet(H_THREAD th);

Remark :

This function returns the stack size of thread for literal string “th” from the value for the size of the stack defined in initval_port.h.

int GpTimerOptSet(int idx, int tmr_tps, int max_exec_tick, void (*irq_tmrfunc)(void));

Remark :

This function registers the kind of user thread, the cycle of event service, event hander and etc by using GpTimerOptSet.

Argument :

int idx	: This function controls the user thread that corresponds to idx.
int tmr_tps	: This function sets how many event services a thread will get per second, that is, Tick per Second.
int max_exec_tick	:
void (*irq_tmrfunc)(void)	: Event Handler

int GpTimerSet(int idx);

This function generates and operates the user thread that corresponds to idx.

int GpTimerPause(int idx);

This function suspends the user thread that corresponds to idx.

int GpTimerResume(int idx);

This function resumes the user thread that corresponds to the suspended idx.

void GpTimerKill(int idx);

This function removes the user thread that corresponds to idx.

Header : gpos_def.h

Appendix: About GpStream

int GpStreamInit(int idx_timer, int tps, int sz_blk_exp, int cnt_block, int cnt_once, int cnt_init);

Remark : This is an initialization function that should be called only once at the beginning of your programming.

Argument : int idx_timer : Timer Index

int tps : Ticks per Second

int sz_blk_exp : The index of block size (The size of actual block = 2^sz_blk_exp byte)

int cnt_block : According to the number of blocks, the actual buffer size is (cnt_blk * (2^sz_blk_exp)) byte.

int cnt_init : The number of data blocks, therefore, the size of data that were initially read is (cnt_init * (2^sz_blk_exp)) byte.

int GpStreamSet(char *p_filename, int repeated);

Remark : This function allocates the sound data.

Argument : char *p_filename : File name

int repeated : Continuous repeating of the same sound. (If 1, the same sound is continuously repeated)

int GpStreamReset(void);

This function initializes streaming buffer (Called by GpSteamEnd).

int GpStreamPlay(void);

This function plays steaming data.

int GpStreamEnd(void);

This function ends streaming data.

void GpStreamTask(void);

This is the work function of streaming data. Currently it is performed at regular intervals of time using timer event.

Header : gpstream.h

Appendix: About GpTyper(Example:refer to Ex_typer)

```
 typedef struct tagTYPER_RESULT{
      char * type_buf;
      char * cursor_ptr;
      char * cursor_line;
      int flag_carret;
      int row_count;
      int row_pos;
      int typer_mode;
      GP_TYPER_FOCUS focus_set;
      int shift_flag;
      unsigned int ticker;
}TYPER_RESULT;
typedef struct tagTYPER_INFO{
      int max_slen;
      char * type_buf;
      int max_row;
      unsigned char cancel_key;
      void (*display_typer)(TYPER_RESULT * t_result);
}TYPER_INFO;
T_RETURN_CODE GpTyperInit(TYPER_INFO * typer_info, int default_mode);
char * GpTyperExecute(int refresh_time);
void GpTyperDestroy(void);
```

Header : gptyper.h