SDK 說明與範例使用

(一) RAYSSDK 包含文件列表
(二) 模塊管理機制
(三) 模塊設計
(四) 系統控制模塊(System)
(五) 時鐘控制模塊(Timer)
(六) 上圖控制模塊(DirectDraw)
(七) 聲音控制模塊(DirectSound)
(八) 輸入控制模塊(DirectInput)
(九) 網路控制模塊(DirectPlay)
(十) CDROM 音軌控制模塊(CD Music)
(十一) 漢字顯示模塊(Font)
(十二) MP3 播放模塊(MP3)
(十三) Windows BMP 圖像格式支援
(十四) PCX 圖像格式支援
(十五) TGA 圖像格式支援
(十六) JPG 圖像格式支援
(十七) PhotoShop PSD 格式支援
(十八) FLC 動畫格式支援
(十九) AVI 動畫格式支援

(一) RAYSSDK 包含文件列表

基本列表如下：

[LIB]
2dengine.lib
2denginedbg.lib
2dengine_640x480.lib
2denginedbg_640x480.lib
2dengine_800x600.lib
2denginedbg_800x600.lib

[INCLUDE]
function.h
jpeg.h
menutree.h
menuwin.h
mp3.h　　// MP3 播放控制模塊
packfile.h
polling.h
rays.h　　// 系統頭文件
readinfo.h
ripple.h
topo.h
undo.h
vbmp.h
vflic.h
vpcx.h
vpsd.h
vtga.h
winfont.h // 漢字顯示模塊
winmain.h
xcak.h
xcdrom.h // CD 音軌驅動控制模塊
xdraw.h　// 上圖控制模塊
xfont.h　// 漢字顯示模塊
xgrafx.h
xinput.h　// 輸入控制模塊
xkiss.h

xmedia.h
xmem.h
xmodule.h　　// 模塊管理系統
xplay.h　　// 網路控制模塊
xpoly.h
xrle.h
xsound.h　　// 聲音控制模塊
xsystem.h　　// 系統控制模塊
xtimer.h　　// 時鐘控制模塊
xvga.h
xwavread.h
ybitio.h
ylzss.h

(二) **模塊管理機制**

規範與原理

在 RAYSSDK 中，所有的可以集成的函數或者介面，均被設計成為一個一個的模塊，以方便調用。

為了統一管理系統中所有的模塊，RAYSSDK 需要每個模塊提供三個函數：

１）初始化該模塊的函數

```
int       init_modulename(void);
```

２）釋放該模塊的函數

```
void      free_modulename(void);
```

３）該模塊響應應用程式啟動與否的處理函數

```
void      active_modulename(int active);
```

系統支援在同一時間內最多可以裝載６４個不同的模塊。需要注意的是，我們不能同時多次裝載同一模塊，否則，將導致不可預期的錯誤。

在 RAYSSDK 的內核中，有一套模塊管理機制，該機制完成以下三種任務：

１）用戶安裝某模塊時，自動執行該模塊的初始化函數，並返回成功與否的值。

２）在應用程式的窗口被啟動或者被取消啟動的時候，會自動調用該模塊的響應函數。

３）在應用程式將要退出時，自動執行該模塊的釋放函數。

那麼，系統是如何實現這些任務的呢？

首先，系統在內存中建立了一個存儲系統所有模塊的釋放函數指針的數組，同時還建立了一個存儲所有模塊的響應程式啟動與否函數的數組。當然，我們不用建立存儲每個模塊初始化函數指針的數組，因為如上所述，我們在用戶安裝某模塊時，已經調用了該模塊的初始化函數。

有了響應應用程式啟動與否模塊數組後，我們便可以很容易的實現正確的響應處理功能了。在系統接收到應用程式開始啟動或者非啟動的消息後，依次執行模塊響應函數（我們已經存儲了對應的函數指針！），ＯＫ！

同樣道理，在應用程式退出時，我們可以依次執行模塊釋放函數。

函數介面

EXPORT void FNBACK init_modules(void);

初始化模塊裝載系統數據。本函數為系統內部調用的。

EXPORT int FNBACK install_module(MODULE_INIT_FUNC
init,MODULE_FREE_FUNC free,MODULE_ACTIVE_FUNC active);

安裝一個模塊。同時，我們可以發現，在 xmodule.h 中，有一個非常好用的宏定義

#define install(a) install_module(init_##a,free_##a,active_##a)

該宏定義可以非常方便的用來實現本函數。

函數參數類型：

typedef int (*MODULE_INIT_FUNC)(void);
typedef void (*MODULE_FREE_FUNC)(void);
typedef void (*MODULE_ACTIVE_FUNC)(int bActive);

參數說明：

MODULE_INIT_FUNC init;//該模塊初始化函數
MODULE_FREE_FUNC free;//該模塊釋放函數
MODULE_ACTIVE_FUNC active;//該模塊響應程式啟動處理函數
EXPORT void FNBACK free_modules(void);

釋放系統中的所有模塊。本函數為系統內部調用的。

EXPORT void FNBACK active_modules(int bActive);

使系統中所有模塊響應應用程式的啟動與否狀態變化。本函數為系統內部調用的。

範例說明

ＯＫ，總結一下。為了讓應用程式介面變的很容易和簡單，用戶需要使用的函數只有一個，即安裝模塊的函數。甚至只是簡單的用一個宏調用。這裡有一些程式片段可以說明：

if(FAILED(install(system))) FailMsg("install system failed");
if(FAILED(install(draw))) FailMsg("install draw failed");

```
    if( FAILED( install(timer)    )) FailMsg("install timer failed");
```

該片段依次裝載系統模塊，上圖模塊和時鐘模塊。並在裝載模塊不成功的時候，彈出致命錯誤消息，結束應用程式的執行。

(三) 模塊設計

設計依據

假如我們需要設計一個新的模塊，通過模塊裝載系統，我們就可以很方便的安裝和管理該模塊了。

按照模塊裝載系統的工作原理和調用函數，我們需要準備三個基本的模塊介面函數。然後，在此基礎上，去發展其他的該模塊的對應函數。

範例說明

為了便於說明，我們假設要建立一個測試模塊，叫做 test。那麼我們必須的函數有：

```
int     init_test(void);
void    free_test(void);
void    active_test(int active);
```

然後，可能還有一些本模塊的其他相關函數如：

```
void    show_test(void);
void    log_test(void);
void    check_test(void);
```

等等。

這樣，我們就可以通過調用 install(test)的宏來裝載 test 模塊，然後在程式執行中，就可以使用比如 show_test(), log_test()等函數了。

容易嗎？你可以很簡單的設計你自己的模塊了。

(四) 系統控制模塊(System)

提供一些系統函數，包含錯誤誌文件，系統檢測等函數。同時，提供了一些數據介面，可以方便應用程式的調用。

數據介面

extern　USTR　　print_rec[2048];

一個臨時字串的緩衝區。在應用程式中，我們經常用到一些字串處理，小的字元數組處理等等，這是，便可以應用到本緩衝區。

extern　USTR　　game_path[_MAX_PATH];

紀錄遊戲運行路徑。在安裝系統模塊的時候，會檢測本應用程式的運行路徑，並保存在該數據區中。
在應用程式的發展中，我們可能會需要知道遊戲目前執行的目錄，這時，便可以通過該數據介面以獲取。

extern　USTR　　game_filename[_MAX_PATH+_MAX_FNAME];

一個臨時存放檔案名的緩衝區。在應用程式中經常會對一些檔案名進行操作，比如更換文件擴展名，自動產生序列的檔案名稱等等，這時，我們就可以使用該緩衝區了。

extern　ULONG　　game_capture_no;

遊戲中抓屏存儲為影像文件時，該文件的序號。往往反應在該檔案名的後幾位元字元上。

系統模塊初始化時，將置之為０，以後每抓屏一次，本數字將遞增１。

在應用程式中，可以直接改變該值，以規範抓屏的存儲影像檔案名。

extern　ULONG　　game_now_time;

紀錄著目前的系統時間。本數據往往和下面的遊戲開始時的系統時間聯合起來使用，以確定目前遊戲進行了多長時間。另外，利用這一時間，我們還可以完成很多其他的任務，比如，確定物件或者ＮＰＣ的刷新時間等。

需要注意的是，在遊戲過程中，本數據是在一直更新的。時間單位為１毫秒，即千分之一秒。

extern　ULONG　　game_start_time;

紀錄遊戲開始時的系統時間。本數據往往和上面的目前系統時間聯合起來使用。

在系統模塊初始化的時候，會獲取系統時間，並保存在本數據中。在遊戲運行過程中，本數據是保持不變的。

函數介面

EXPORT　int　　FNBACK　init_system(void);
EXPORT　void　FNBACK　active_system(int active);
EXPORT　void　FNBACK　free_system(void);

以上三個函數是系統控制模塊的模塊管理介面函數。

EXPORT　void　FNBACK　log_error(int p, USTR *strMsg);

功　能：將訊息存放在磁盤文件中。一般用來在出錯時，將錯誤訊息寫入磁盤文件，以方便ＤＥＢＵＧ。

參　數：int p; //判斷標誌，當為０時，會直接返回；否則，寫入訊息。
　　USTR *strMsg; //需要寫入的字串訊息。
返回值：無
說　明：
範　例：

```
char *buffer;
buffer = (char *)malloc(1000);
if(! buffer)
{
    log_error(1, (USTR*)“memory alloc error”);
}
if(buffer) free(buffer);
```

EXPORT　void　FNBACK　log_error(int p, char *strMsg, ...);

功　能：將訊息存放在磁盤文件中。一般用來在出錯時，將錯誤訊息寫入磁盤文件，以方便ＤＥＢＵＧ。

參　數：int p; //判斷標誌，當為０時，會直接返回；否則，寫入訊息。
　　char *strMsg; //格式化定義字串
　　…; //其他相關參數

返回值：無

說　明：無

EXPORT　void　FNBACK　idle_loop(void);

功　能：進行 WINDOWS 消息接收與處理。

參　數：無

返回值：無

說　明：在應用程式執行一個耗時的循環或者循環中需要接收 WINDOWS 消息時，需要在循環內的開始調用此函數。否則，將會使本應用程式長時間佔用 WINDOWS 系統時間，造成系統停頓。或者是本循環中接收不到 WINDOWS 的消息。

範　例：

```
SLONG main_pass;
UCHR ch;
main_pass = 0;
while( 0 == main_pass)
{
idle_loop();
//doing something
ch = read_data_key();
if(ch == S_Esc)
    main_pass = 1;
else
    reset_data_key();
}
```

EXPORT SLONG FNBACK is_gb_windows(void);

功　能：檢查操作系統是否是簡體 WINDOWS。

參　數：無

返回值：若是簡體 WINDOWS，返回 TRUE；否則，返回 FALSE。

說　明：無

EXPORT void FNBACK run_random_init(void);

功　能：初始化隨機數。

參　數：無

返回值：無

說　明：應用程式在執行時，隨機數序列的值是固定的，為了獲得每次不同的隨機數序列，我們需要調用本函數，以初始化隨機數序列。

EXPORT SLONG FNBACK get_cdrom_drive(void);

功　能：獲取 CDROM 驅動器的編號，並返回一個代表 CDROM 驅動器編號的數。

參　數：無

返回值：如果沒有找到任何 CDROM 驅動器，將返回-1；否則返回非負數。

不同的數字代表不同的磁盤驅動器，(0=A:;1=B:;2=C:;)3=D:;4=E;等，以此類推。

說　明：無

EXPORT SLONG FNBACK check_cdrom_volume(USTR *title);

功　能：獲取符合對應卷標的 CDROM 驅動器。

參　數：USTR *title; //卷標字串

返回值：如果沒有符合的，返回-1；否則，返回一個非負數。

不同的數字代表不同的磁盤驅動器，(0=A:;1=B:;2=C:;)3=D:;4=E;等，以此類推。

說　明：無

EXPORT USTR * FNBACK get_cdrom_volume(SLONG drive);

功　能：獲取指定 CDROM 的卷標。

參　數：SLONG drive; //CDROM 驅動器編號，0=A:;1=B;…

返回值：USTR *字串，為獲取的卷標。如果不成功，為 NULL。

說　明：無

EXPORT　void　FNBACK　store_game_path(USTR *path);
功　能：存儲當前遊戲運行路徑。
參　數：USTR *path; //存放路徑的指針
返回值：無
說　明：系統內部使用的函數。

EXPORT　void　FNBACK　capture_screen(void);
功　能：抓屏並保存為影像文件。
參　數：無
返回值：無
說　明：無

EXPORT　ULONG　FNBACK　get_fps(void);
功　能：獲得當前上圖的速度。
參　數：無
返回值：返回一個數值，表示 FPS，即 Frames Per Second，每秒幀數。
說　明：需要在每個上圖循環中調用一次才能獲得正確的上圖速度。

EXPORT　USTR * FNBACK　get_computer_name(void);
功　能：獲得計算機名稱
參　數：無
返回值：計算機名稱
說　明：無

EXPORT　USTR * FNBACK　get_user_name(void);
功　能：獲得當前 WINDOWS 用戶名稱
參　數：無
返回值：WINDOWS 用戶名稱
說　明：無

EXPORT　USTR *　FNBACK　get_windows_directory(void);
功　能：獲得 WINDOWS 的目錄。
參　數：無
返回值：WINDOWS 系統目錄。
說　明：無

EXPORT SLONG FNBACK get_windows_version(void);
功　能：獲得 WINDOWS 的版本。
參　數：無
返回值：如下所示

```
typedef enum WINDOWS_TYPE_ENUMS
{     WINDOWS_NT    = 3,
      WINDOWS_32        = 2,
      WINDOWS_95        = 1,
} WINDOWS_TYPE;
```

說　明：無

EXPORT void FNBACK get_memory_status(ULONG *total_phys,ULONG *avail_phys);
功　能：獲得當前系統物理內存使用情況。
參　數：ULONG *total_phys; //存放總共物理內存大小的指針
　　ULONG *avail_phys; //存放當前可使用物理內存大小的指針
返回值：無
說　明：無

EXPORT ULONG FNBACK get_disk_serial_no(void);
功　能：獲取 C:磁盤序列號碼。
參　數：無
返回值：該序列碼的值。
說　明：無

EXPORT ULONG FNBACK get_cpu_clock(void);
功　能：獲得 CPU 的時鐘週期。
參　數：無
返回值：CPU 的時鐘週期，單位為 MHZ。
說　明：無

EXPORT char * FNBACK get_cpu_id(void);
功　能：獲得 CPU 的標識 ID。
參　數：無
返回值：標識字串。
說　明：無

EXPORT void FNBACK analyst_system(void);

功　能：分析系統。

參　數：無

返回值：無

說　明：分析系統的 CPU，內存，作業系統等；並將這些寫入錯誤誌文件。

引用提示

在 RAYSSDK 中，與系統模塊相關的文件為 xsystem.cpp 和 xsystem.h。在使用已經生成的 LIB 的情況下，我們只需要包含頭文件 xsystem.h 即可。

(五) 時鐘控制模塊(Timer)

模塊功能

提供比較精確的時鐘(Timer)給應用程式，以便於應用程式中的定時等控制。本模塊所提供的時鐘單位為 1/100 秒，即 10 毫秒。

本模塊提供多個計時數據，用戶可以通過獲得該數據的值或者改變該數據的值。每當系統時間過去 1/100 秒，這些計時數據就會加 1，這個動作是時鐘控制模塊自己完成的。

數據介面

```
extern ULONG timer_tick00;
extern ULONG timer_tick01;
extern ULONG timer_tick02;
extern ULONG timer_tick03;
extern ULONG timer_tick04;
extern ULONG timer_tick05;
extern ULONG timer_tick06;
extern ULONG timer_tick07;
extern ULONG timer_tick08;
extern ULONG timer_tick09;
```

以上的時鐘數據是用戶可以引用的。

```
extern ULONG cdrom_timer_tick;
```

系統保留時鐘，專門用來控制 CDROM 音樂播放。

```
extern ULONG cursor_timer_tick;
```

系統保留時鐘，專門用來控制圖形動畫光標的刷新計時。

```
extern ULONG system_timer_tick;
```

系統保留時鐘。

函數介面

```
EXPORT int      FNBACK    init_timer(void);
EXPORT void     FNBACK    free_timer(void);
EXPORT void     FNBACK    active_timer(int bActive);
```

以上三個函數是時鐘控制模塊的模塊管理介面函數。

我們在需要使用本模塊的數據介面時，必須要先裝載模塊。通過模塊管理的宏 install(timer)，我們將可以很容易的做到這些。

(六) 上圖控制模塊(DirectDraw)

模塊功能

上圖控制模塊利用 DirectDraw 作為底層，提供應用程式對顯示螢幕直接操作的介面。

其基本動作為初始化 DirectDraw，設定協作級別，創建螢幕緩衝區和後備緩衝區，設定上圖模式等等。同時，會獲取顯示卡的 16 位色類型(555,565 等)並根據該類型設定一系列的上圖函數。

為了提高應用程式上圖及對位元圖操作的速度，RAYSSDK 中提供了一系列對應不同顯示卡的繪圖函數，同時有一系列的函數指針，當獲取顯示卡類型後，上圖控制模塊會設定這些函數指針的值，使其指向對應的函數實體。
這些函數指針包含於引擎的位元圖操作函數集中，相關文件為 xgrafx.cpp 和 xgrafx.h。

數據介面

extern LPDIRECTDRAW7 lpDD7; /* system directdraw object */
DirectDraw 物件。在裝載上圖控制模塊時會被初始化。

本數據為系統使用，建議應用程式不要對其進行操作。

extern LPDIRECTDRAWSURFACE7 lpDDSPrimary7;/* system directdraw primary surface*/
螢幕主緩衝區，建立於顯示內存中的，對應於遊戲的顯示窗口客戶區。

本數據為系統使用，建議應用程式不要對其進行操作。

extern LPDIRECTDRAWSURFACE7 lpDDSBack7; /* system directdraw back surface */

螢幕後備緩衝區，建立於顯示內存中的。在 DirectDraw 上圖時，為了獲得平滑的上圖效果，不產生螢幕撕裂，上圖控制模塊採用獲取螢幕後備緩衝區指針，對其進行操作，然後 Flip 的方式上圖。

本數據為系統使用，建議應用程式不要對其進行操作。

extern LPDIRECTDRAWSURFACE7 lpDDSMemory7; /* memory directdraw surface */
建立於系統內存中的緩衝區。
系統利用本緩衝區進行 AVI 動畫文件的播放(DirectMedia)。

應用程式不要對其進行操作。

extern ULONG nBackBuffers; /* system directdraw back surface count */
螢幕後備緩衝區的數目。
在初始化上圖控制模塊時，會根據機器顯示卡上顯存的大小，建立適當多的螢幕後備緩衝區，以提高上圖速度。本數據會記錄創建的後備緩衝區的數目。

extern SLONG vga_type;　　/* system video card type */
顯示卡 16 位色類型。

當上圖模塊初始化時，會獲取機器顯示卡 16 位色類型(PixelFormat)，並將該類型紀錄在本數據中。
數據可能為以下值：

VGA_TYPE_555　//555 類型顯卡
VGA_TYPE_655　//655 類型顯卡
VGA_TYPE_565　//565 類型顯卡
VGA_TYPE_556　//556 類型顯卡
VGA_TYPE_ANY　//其他類型顯卡

函數介面

EXPORT int　FNBACK init_draw(void);
EXPORT void FNBACK free_draw(void);
EXPORT void FNBACK active_draw(int bActive);

以上三個函數是上圖控制模塊的模塊管理介面函數。

我們在需要使用本模塊的數據介面時，必須要先裝載模塊。通過模塊管理的宏 install(draw)，我們將可以很容易的做到這些。

EXPORT void FNBACK set_update_area(int start,int height);
功　能：設定更新螢幕的區域。
參　數：int start;　//更新螢幕的起始行，初始化時為 0。
　　int height; //更新螢幕的行數，初始化為 SCREEN_HEIGHT。
返回值：無
說　明：

EXPORT void FNBACK set_update_type(int type);

功　能：設定更新螢幕的方式。

參　數：int type; //指定更新螢幕的方式

其取值可以為以下之一：

NORMAL_UPDATE_SCREEN //按照常規方式更新螢幕

PEST_UPDATE_SCREEN //採用濾掉穿透色 0X0000 的 PEST 方式更新螢幕

返回值：無

說　明：

EXPORT void FNBACK get_bitmap_from_memory_surface(BMP *bmp, RECT rect, SLONG left_top_flag);

功　能：從系統內存緩衝區 lpDDSMemory7 中獲取 BITMAP。

參　數：BMP *bmp; //存放獲取的 BITMAP 的引擎位元圖 BMP 結構指針

RECT rect;//獲取的矩形範圍

SLONG left_top_flag; //是否從 bmp 的左上角開始存放的標誌

當此標誌為 1 時，會將獲取的影像數據存放在 bmp 的左上角開始的區域。

為 0 時，會將獲取的影像數據存放在 bmp 中指定 RECT 對應的區域。

返回值：無

說　明：本函數在 RAYSSDK 中被用在 AVI 文件的播放控制中(DirectMedia)。

一般來說，不建議應用程式使用。

範　例：為了便於理解，舉例如下：

```
RECT rc;
rc.left = 50;
rc.right = 150;
rc.top = 50;
rc.bottom = 150;
//(1)獲取 lpDDSMemory7 中(50,50)-(150,150)指定的 100X100 的影像資料，並將該資料存放在 screen_channel0 的從左上角開始的(0,0)-(100,100)的區域。
get_bitmap_from_memory_surface(screen_channel0, rc, 1);

//(2) 獲取 lpDDSMemory7 中(50,50)-(150,150)指定的 100X100 的影像資料，並將該資料存放在 screen_channel0 的與 RECT rc 對應的(50,50)-(150,150)的區域。

get_bitmap_from_memory_surface(screen_channel0, rc, 0);
```

EXPORT void FNBACK switch_screen_mode(void);

功　能：切換螢幕的顯示模式為全屏模式或者視窗模式。

參　數：無

返回值：無

說　明：在應用程式運行過程中，用戶可以按 F12 來即時改變當前的螢幕顯示模式。系統便是通過這一函數來實現的。

本函數為系統使用，不建議用戶在應用程式中使用。

EXPORT void FNBACK setup_vga_function(DWORD dwRBitMask, DWORD dwGBitMask, DWORD dwBBitMask);

功　能：設定影像卡的顏色模式和相關函數。

參　數：DWORD dwRBitMask; //圖元的紅色分量掩碼

DWORD dwGBitMask; //圖元的綠色分量掩碼

DWORD dwBBitMask; //圖元的藍色分量掩碼

這些掩碼的取值可參考 xvga.h，具體為：

R_MASK_555 //對應於 555 格式的顯示卡

G_MASK_555 //

B_MASK_555 //

R_MASK_655 //對應於 655 格式的顯示卡

G_MASK_655 //

B_MASK_655 //

R_MASK_565 //對應於 565 格式的顯示卡

G_MASK_565 //

B_MASK_565 //

R_MASK_556 //對應於 556 格式的顯示卡

G_MASK_556 //

B_MASK_556 //

返回值：無

說　明：

在我們裝載上圖控制模塊 install(draw)時，系統會自動調用本函數來設定影像卡的顏色模式和相關函數。

但是，我們也可以直接使用本函數。比如，我們在寫一些需要 WINDOWS 的 GDI 支援時，因為 WINDOWS 的 GDI 採用的是 555 格式，所以我們可以如此來設定：

```
setup_vga_function( R_MASK_555, G_MASK_555, B_MASK_555 );
```

引用提示

在 RAYSSDK 中，與系統模塊相關的文件為 xdraw.cpp 和 xdraw.h。在使用已經生成的 LIB 的情況下，我們只需要包含頭文件 xdraw.h 即可。

(七) 聲音控制模塊(DirectSound)

模塊功能

聲音控制模塊利用 DirectSound 作為底層，提供給用戶一個方便適用的聲音控制介面。

引擎可以控制同時播放最多 8 個通道的聲音。為了方便在遊戲中對音樂和音效的不同處理，引擎將這聲音分類為兩類，一類為音效，佔據 0~6 的通道，另外一類為音樂，佔據第 7 號通道。

利用聲音控制模塊，我們可以進行混音，改變某通道聲音大小，均衡等。

數據介面

```
extern SOUND_CFG sound_cfg;
```

紀錄引擎聲音配置的結構。
該結構的定義如下：

```
typedef struct tagSOUND_CFG
{
    SLONG music_flag; //播放音樂的標誌，1＝播放音樂，0＝不播放音樂
    SLONG music_no; //當前播放音樂的標號
    SLONG music_total; //總共共多少首音樂
    SLONG music_volume;   //音樂的音量大小
    SLONG music_pan; //音樂的均衡值
    SLONG voice_flag; //音效的播放標誌，1＝播放，0＝不播放。
    SLONG voice_volume; //音效的音量大小
    SLONG voice_pan; //音效的均衡值
} SOUND_CFG;
```

有關音量大小與均衡值的取值範圍，有如下的定義：

```
    MUSIC_VOLUME_MIN    //-10000，音樂最小音量
    MUSIC_VOLUME_MAX    //0，音樂最大音量
    MUSIC_PAN_LEFT      //-10000，音效均衡居左
    MUSIC_PAN_CENTER    //0，音效均衡居中
    MUSIC_PAN_RIGHT     //10000，音效均衡居右
    VOICE_VOLUME_MIN    //-10000，音效最小音量
    VOICE_VOLUME_MAX    //0，音效最大音量
    VOICE_PAN_LEFT      //-10000，音效均衡居左
```

VOICE_PAN_CENTER　//0，音效均衡居中
VOICE_PAN_RIGHT　//10000，音效均衡居右

函數介面

EXPORT int　FNBACK init_sound(void);
EXPORT void FNBACK free_sound(void);
EXPORT void FNBACK active_sound(int active);

以上三個函數是聲音控制模塊的模塊管理介面函數。

EXPORT void FNBACK init_sound_cfg(void);
功　能：初始化聲音配置結構 sound_cfg。
參　數：無
返回值：無
說　明：

EXPORT void FNBACK notify_changed_sound_cfg(SLONG changed_flags);
功　能：當應用程式改變了 sound_cfg 中的內容時，為了讓引擎響應所作的改變，需要調用此函數以通知引擎。
參　數：SLONG changed_flags; //改變的數據域標記
該標記可以由以下的值組合(|)在一起：

CHANGED_MUSIC_FLAG　//改變了音樂標誌
CHANGED_MUSIC_TOTAL　//改變了音樂數目
CHANGED_MUSIC_VOLUME //改變了音樂的音量
CHANGED_MUSIC_PAN　//改變了音樂的均衡
CHANGED_VOICE_FLAG　//改變了音效的標誌
CHANGED_VOICE_VOLUME //改變了音效的音量
CHANGED_VOICE_PAN　//改變了音效的均衡

返回值：無
說　明：

EXPORT void FNBACK play_voice(SLONG channel,SLONG loop,USTR *filename);

功　能：播放音效。

參　數：SLONG channel; //指定播放音效的聲音通道，0~6。

　　SLONG loop;　//是否循環播放的標誌，1=循環播放，0=播放一次

　　USTR *filename; //音效檔案名稱(*.WAV)

返回值：無

說　明：

EXPORT void FNBACK stop_voice(SLONG channel);

功　能：停止指定通道的音效播放。

參　數：SLONG channel; //指定的音效通道

返回值：無

說　明：

EXPORT void FNBACK set_voice_volume(SLONG channel,SLONG volume);

功　能：設置指定音效通道的聲音音量。

參　數：SLONG channel; //指定音效通道

　　SLONG volume;　//需要設定的音量

返回值：無

說　明：

EXPORT void FNBACK set_voice_pan(SLONG channel,SLONG pan);

功　能：設定指定音效通道的聲音均衡。

參　數：SLONG channel; //指定音效通道

　　SLONG pan;　　//設定的均衡值

返回值：無

說　明：

EXPORT SLONG FNBACK is_voice_playing(SLONG channel);

功　能：檢查指定通道的音效是否正在播放

參　數：SLONG channel; //音效通道

返回值：如果正在播放，返回 TRUE；否則，返回 FALSE。

說　明：

EXPORT void FNBACK play_music(SLONG music_no, SLONG loop);

功　能：播放音樂

參　數：SLONG music_no; //音樂編號

　　SLONG loop;　　//循環播放標誌，1=循環播放，0=播放一次

返回值：無

說　明：

EXPORT void FNBACK stop_music(void);

功　能：停止播放音樂

參　數：無

返回值：無

說　明：

EXPORT void FNBACK set_music_volume(SLONG volume);

功　能：設定音樂的音量。

參　數：SLONG volume; //指定音量

返回值：無

說　明：

EXPORT void FNBACK set_music_pan(SLONG pan);

功　能：設定音樂的均衡。

參　數：SLONG pan; //指定均衡值

返回值：無

說　明：

EXPORT SLONG FNBACK is_music_playing(void);

功　能：檢查是否正在播放音樂。

參　數：無

返回值：如果正在播放，返回 TRUE；否則，返回 FALSE。

說　明：

EXPORT SLONG FNBACK get_wavfile_information(USTR *filename, ULONG *channels, ULONG *sample_rate,ULONG *bps, ULONG *size, ULONG *play_time);

功　能：獲取指定 WAV 文件的資訊。

參　數：USTR *filename;　　//WAV 檔案名
　　ULONG *channels;　　//存放該 WAV 的聲音通道數目
　　ULONG *sample_rate;　//存放該 WAV 的採樣率
　　ULONG *bps;　　//存放該 WAV 的比特率(BitsPerSample)
　　ULONG *size;　　//存放該 WAV 的數據大小
　　ULONG *play_time;　　//存放該 WAV 可以播放多長時間

返回值：無

說　明：如果獲取成功，會返回 TTN_OK；否則，返回 TTN_ERROR。

引用提示

　　與聲音控制模塊相關的文件為 xsound.cpp 和 xsound.h。應用程式在引用 LIB 時，直接包含頭文件
xsound.h 即可。

(八) 輸入控制模塊(DirectInput)

模塊功能

本模塊實現對鍵盤和滑鼠的輸入控制。早先規劃為利用 DirectInput 來實現，不過目前採用的是截取 WINDOWS 的消息來實現的。

數據介面

沒有提供任何對外數據介面。

函數介面

EXPORT int FNBACK init_input(void);
EXPORT void FNBACK free_input(void);
EXPORT void FNBACK active_input(int bActive);
以上三個函數為輸入控制模塊的模塊裝載介面函數。

EXPORT UCHR FNBACK read_system_key(void);

功　能：讀取系統按鍵。
參　數：無
返回值：若有按鍵，返回鍵值；否則，返回 0。
鍵值的定義在 rays.h 中，為以下值：

```
enum KEY_CODES
{   Backspace = 0x08,
    Tab       = 0x09,
    BackTab   = 0x0f,
    Lf        = 0x0a,
    Enter     = 0x0d,
    Esc       = 0x1b,
    Blank     = 0x20,
    Plus      = 0x2b,
    Comma     = 0x2c,
    Dot       = 0x2e,
    Minus     = 0x2d,
    Zero      = 0x30,
```

```
Colon       = 0x3a,
KB_F1        = 0x3b,
KB_F2        = 0x3c,
KB_F3        = 0x3d,
KB_F4        = 0x3e,
KB_F5        = 0x3f,
KB_F6        = 0x40,
KB_F7        = 0x41,
KB_F8        = 0x42,
KB_F9        = 0x43,
KB_F10       = 0x44,
KB_F11       = 0x85,
KB_F12       = 0x86,
KB_F13       = 0x8d,
KB_F14       = 0x8e,
KB_F15       = 0x8f,
Home         = 0x47,
Up           = 0x48,
PgUp         = 0x49,
Left        = 0x4b,
Right       = 0x4d,
End          = 0x4f,
Dn           = 0x50,
PgDn         = 0x51,
Ins          = 0x52,
Del          = 0x53,
Shift_F1    = 0x54,
Shift_F2    = 0x55,
Shift_F3    = 0x56,
Shift_F4    = 0x57,
Shift_F5    = 0x58,
Shift_F6    = 0x59,
Shift_F7    = 0x5a,
Shift_F8    = 0x5b,
Shift_F9    = 0x5c,
Shift_F10 = 0x5d,
Shift_F11 = 0x87,
Shift_F12 = 0x88,
```

```
Shift_F13 = 0x90,
Shift_F14 = 0x91,
Shift_F15 = 0x92,
Ctrl_F1      = 0x5e,
Ctrl_F2      = 0x5f,
Ctrl_F3      = 0x60,
Ctrl_F4      = 0x61,
Ctrl_F5      = 0x62,
Ctrl_F6      = 0x63,
Ctrl_F7      = 0x64,
Ctrl_F8      = 0x65,
Ctrl_F9      = 0x66,
Ctrl_F10     = 0x67,
Ctrl_F11     = 0x89,
Ctrl_F12     = 0x8a,
Ctrl_F13     = 0x93,
Ctrl_F14     = 0x94,
Ctrl_F15     = 0x95,
Alt_F1       = 0x68,
Alt_F2       = 0x69,
Alt_F3       = 0x6a,
Alt_F4       = 0x6b,
Alt_F5       = 0x6c,
Alt_F6       = 0x6d,
Alt_F7       = 0x6e,
Alt_F8       = 0x6f,
Alt_F9       = 0x70,
Alt_F10      = 0x71,
Alt_F11      = 0x8b,
Alt_F12      = 0x8c,
Alt_F13      = 0x96,
Alt_F14      = 0x97,
Alt_F15      = 0x98,
Ctrl_End     = 0x75,
Ctrl_PgDn = 0x76,
Ctrl_Home = 0x77,
Alt_1        = 0x78,
Alt_2        = 0x79,
```

```
    Alt_3       = 0x7a,
    Alt_4       = 0x7b,
    Alt_5       = 0x7c,
    Alt_6       = 0x7d,
    Alt_7       = 0x7e,
    Alt_8       = 0x7f,
    Alt_9       = 0x80,
    Alt_0       = 0x81,
    L_Shift     = 0x82,
    R_Shift     = 0x83,
    Ctrl        = 0x84,
    Alt         = 0x85,
    Num_5       = 0x87
};
```

說　明：強烈推薦讀取鍵盤用後面的 read_data_key()，它可以讀取更多的鍵值。保留這個函數只是為了與原來的程式相容。

EXPORT UCHR FNBACK read_data_key(void);

功　能：讀取數據鍵。

參　數：無

返回值：若有按鍵，返回數據值，否則，返回０。

按鍵的數據值可以為從鍵盤上的可列印字元，還可以為一些系統按鍵。這些系統按鍵的數據值的定義在 rays.h 中，為以下值：

```
enum KEY_SPECIAL_CODES
{   S_Backspace = 0x08,
    S_Tab       = 0x89,
    S_BackTab   = 0x8f,
    S_Lf        = 0x8a,
    S_Enter     = 0x8d,
    S_Esc       = 0x9b,
    S_Blank     = 0xA0,
    S_Plus      = 0xAb,
    S_Comma     = 0xAc,
    S_Dot       = 0xAe,
    S_Minus     = 0xAd,
```

```
    S_Zero          = 0xB0,
    S_Colon         = 0xBa,
    S_KB_F1          = 0xBb,
    S_KB_F2          = 0xBc,
    S_KB_F3          = 0xBd,
    S_KB_F4          = 0xBe,
    S_KB_F5          = 0xBf,
    S_KB_F6          = 0xC0,
    S_KB_F7          = 0xC1,
    S_KB_F8          = 0xC2,
    S_KB_F9          = 0xC3,
    S_KB_F10         = 0xC4,
    S_KB_F11         = 0xC5,
    S_KB_F12         = 0xC6,
    S_Home           = 0xC7,
    S_Up             = 0xC8,
    S_PgUp           = 0xC9,
    S_Left          = 0xCb,
    S_Right         = 0xCd,
    S_End            = 0xCf,
    S_Dn             = 0xD0,
    S_PgDn           = 0xD1,
    S_Ins           = 0xD2,
    S_Del            = 0xD3,
};
```

說　明：

EXPORT void FNBACK reset_key(void);

功　能：重置鍵盤緩衝區中所有的值。

參　數：無

返回值：無

說　明：我們在利用讀取了鍵盤後，該鍵值在鍵盤緩衝區中一直是保留著的。如果這時候我們不再按鍵，而一直調用讀鍵的函數，我們獲取的鍵值會一直為上次的值。所以，一般來說，當我們讀完鍵後，會重置鍵盤緩衝區，以便於後一步的讀鍵操作。典型用法見以下的範例：

範　例：

```
UCHR ch; //定義一個存放讀鍵返回值的數據
ch = read_system_key(); //讀取系統鍵
if(ch) reset_key(); //如果有按鍵，重置鍵盤緩衝區，已備下此讀鍵
switch(ch) //根據讀取的按鍵作不同的操作
{
	case Home:
		break;
	case End:
		break;
	default:
		break;
}
```

EXPORT void FNBACK reset_data_key(void);

功　能：清除鍵盤緩衝區中數據鍵的值。
參　數：無
返回值：無
說　明：與 reset_key()不同，本函數只是清除鍵盤緩衝區中可列印字元鍵的鍵值。一般在我們需要實現文字編輯功能的時候，我們往往需要實現這樣的功能，比如，按住 Shift 鍵，然後輸入一串文字，這時候我們希望的結果是得到對應鍵盤的上檔符號。程式在實現的時候，每獲得一個按鍵後，便利用這個函數重置鍵盤。實現這個的過程可以參考下面的範例。

範　例：

```
UCHR ch;
USTR input_str[80];
SLONG input_finish;
memset(input_str, 0, 80);
input_finish = 0;
while( ! input_finish)
{
	ch = read_data_key();
//如果我們在輸入時一直按住 Shift，然後按 12345，我們會獲得!@#$%；
```

```
//但是如果我們這裡用的不是 reset_data_key()，而是 reset_key()，我們將獲得!2345。
//因為在我們獲得了!後，reset_key()將我們按住的 Shift 也清除了

if(ch) reset_data_key();
switch(ch)
{
    case S_Del:
//TODO:刪除編輯光標後的一個字元
        break;
    case S_Backspace:
//TODO:刪除編輯光標前的一個字元
        break;
    case S_Enter: //輸入結束
        input_finish = 1;
        break;
    default:
        if(isprint(ch))
        {
//TODO:添加字元到 input_str 中
        }
        break;
    }
}
```

EXPORT void FNBACK wait_tick(ULONG no);

功　能：等待指定的時間，直到時間到。
參　數：ULONG no; //需要等待的時間，單位為 1/100 秒
返回值：無
說　明：

EXPORT void FNBACK wait_key(SLONG key);

功　能：等待指定的鍵，直到該鍵被按下。
參　數：SLONG key; //鍵值
該鍵值與 read_data_key()的返回鍵值的定義一樣。可以參照 read_data_key()的相關介紹。
返回值：無
說　明：

EXPORT UCHR FNBACK wait_any_key(void);

功　能：等待任意鍵被按下。

參　數：無

返回值：無

說　明：等待的鍵的定義與 read_data_key()相同。

EXPORT void FNBACK wait_key_time(SLONG key,SLONG no);

功　能：在指定時間內等待某鍵被按下。如果按下了該鍵，會立即返回。否則如果時間到，也會返回。

參　數：SLONG key; //等待的鍵值

SLONG no; //等待時間，單位為 1/100 秒

返回值：無

說　明：等待的鍵值的定義與 read_data_key()相同。

EXPORT void FNBACK clear_time_delay(void);

功　能：清除精確時間計數。

參　數：無

返回值：無

說　明：當我們需要利用精確的時間以控制某些代碼時，便可以利用 RAYSSDK 提供的精確時間控制函數。具體包含三個函數 clear_timer_delay()，get_time_delay() 和 wait_time_delay()。

EXPORT ULONG FNBACK get_time_delay(void);

功　能：獲得精確時間計數。

參　數：無

返回值：精確時間值，單位為毫秒(1/1000 秒)。

說　明：

EXPORT void FNBACK wait_time_delay(ULONG count);

功　能：等待指定的精確時間。

參　數：ULONG count; //等待時間，單位為毫秒

返回值：無

說　明：

EXPORT void FNBACK fnKeyBoardInterrupt(UCHR keycode);

功　能：鍵盤中斷處理。

參　數：

返回值：

說　明：系統專用函數。

EXPORT SLONG FNBACK fnMouseInterrupt(UINT message,WPARAM wParam,LPARAM lParam);

功　能：滑鼠中斷處理。

參　數：

返回值：

說　明：系統專用函數。

EXPORT void FNBACK show_mouse(SLONG flag);

功　能：根據顯示標誌顯示滑鼠。

參　數：SLONG flag;//顯示滑鼠的標誌

該標誌可以為以下值：

SHOW_WINDOW_CURSOR //顯示 WINDOWS 滑鼠光標

SHOW_IMAGE_CURSOR　//顯示用戶的動畫影像光標

返回值：無

說　明：

EXPORT void FNBACK set_mouse_cursor(SLONG type);

功　能：設定滑鼠的光標類型。

參　數：SLONG type; //光標類型值

這個值的定義可以由用戶自己定義。不過系統提供能夠支援最多 64 種不同光標。因此，定義的值應該在 0~63 之間。

用戶利用 RAYSSDK 提供的 load_mouse_cursor()和 load_mouse_image_cursor()可以實現載入 WINDOWS 滑鼠光標或用戶的動畫影像光標為指定的 type 類型。之後，便可以用本函數方便的選用所載入的光標了。

返回值：無

說　明：

EXPORT void FNBACK set_mouse_position(SLONG xpos,SLONG ypos);

功　能：設定滑鼠在工作區的位置。

參　數：SLONG xpos; //滑鼠的 x 座標

　　SLONG ypos; //滑鼠的 y 座標

返回值：無

說　明：

EXPORT void FNBACK get_mouse_position(SLONG *xpos,SLONG *ypos);

功　能：獲取滑鼠在工作區的位置。

參　數：SLONG *xpos; //存儲滑鼠 x 座標的數據指針

　　SLONG *ypos; //存儲滑鼠 y 座標的數據指針

返回值：無

說　明：

EXPORT UCHR FNBACK get_mouse_key(void);

功　能：讀取滑鼠的按鍵。

參　數：無

返回值：如果有按鍵，返回對應的滑鼠鍵值；否則，返回 0。

返回值的定義在 rays.h 中，為以下值：

```
enum      MOUSE_KEY_CODES
{    MS_Dummy     = 0x00, //沒有滑鼠按鍵
     MS_Move       = 0xF1,//滑鼠在移動
     MS_LDrag      = 0xF2,//滑鼠左鍵按住拖動
     MS_RDrag      = 0xF3,//滑鼠右鍵按住拖動
     MS_LDn        = 0xF4,//滑鼠左鍵按下
     MS_LUp        = 0xF5,//滑鼠左鍵放開
     MS_LDblClk    = 0xF6,//滑鼠左鍵雙擊
     MS_RDn        = 0xF7,//滑鼠右鍵按下
     MS_RUp        = 0xF8,//滑鼠右鍵放開
     MS_RDblClk    = 0xF9,//滑鼠右鍵雙擊
     MS_MDn        = 0xFA,//滑鼠中間按下
     MS_MUp        = 0xFB,//滑鼠中間松開
     MS_MDblClk    = 0xFC,//滑鼠中間雙擊
     MS_Forward    = 0xFD,//滑鼠滾輪往前滾動
     MS_Backward   = 0xFE //滑鼠滾輪往後滾動
};
```

說　明：

EXPORT UCHR FNBACK read_mouse_key(void);
功　能：與 get_mouse_key()完全相同。只是為了相容性提供兩個不同的函數名稱。
參　數：無
返回值：
說　明：

EXPORT void FNBACK wait_mouse_any_key(void);
功　能：等待滑鼠的任意鍵(非 MS_Dummy)。
參　數：無
返回值：無
說　明：

EXPORT void FNBACK wait_mouse_key(UCHR key);
功　能：等待滑鼠的某個指定的鍵。
參　數：UCHR key; //需要等待的滑鼠按鍵
返回值：無
說　明：

EXPORT SLONG FNBACK check_mouse_shift(void);
功　能：檢查有滑鼠按鍵時，鍵盤的 Shift 是否被按住。
參　數：無
返回值：如果此時 Shift 被按住，返回 TRUE；否則，返回 FALSE。
說　明：

EXPORT SLONG FNBACK check_mouse_control(void);
功　能：檢查有滑鼠按鍵時，鍵盤的 Ctrl 是否被按住。
參　數：無
返回值：如果此時 Ctrl 被按住，返回 TRUE；否則，返回 FALSE。
說　明：

EXPORT void FNBACK reset_mouse(void);
功　能：重置滑鼠的按鍵緩衝區。
參　數：無
返回值：無
說　明：

EXPORT void FNBACK reset_mouse_key(void);

功　能：本函數與 reset_mouse()完全相同。只是為了相容性提供兩個不同的函數名稱。

參　數：

返回值：

說　明：

EXPORT SLONG FNBACK load_mouse_cursor(SLONG index,HCURSOR hCursor);

功　能：載入 WINDOWS 的滑鼠光標資源，並將該光標指定為特定的滑鼠光標類型。

參　數：SLONG index; //指定的滑鼠光標類型
HCURSOR hCursor; //WINDOWS 的光標資源的 HANDLE

返回值：成功時返回 0 ，否則為其他數。

說　明：

EXPORT void FNBACK set_mouse_spot(SLONG index,SLONG x,SLONG y);

功　能：設定用戶動畫影像滑鼠的熱點。

參　數：SLONG index; //需要設定的滑鼠光標類型
SLONG x; //熱點的 x 座標
SLONG y; //熱點的 y 座標

返回值：無

說　明：當我們將以個動畫圖檔設定為滑鼠光標時，系統會將該影像的矩形外框左上角作為滑鼠光標的熱點。但是，在具體應用中，我們往往需要指定特定的點，比如指定影像的中心點或者任意其他點作為熱點等。這時候我們就可以利用本函數來實現之。熱點座標為從上述左上角開始的相對座標。

EXPORT SLONG FNBACK init_mouse_image_cursor(void);

功　能：初始化滑鼠的影像光標存儲區。

參　數：無

返回值：返回 TTN_OK。

說　明：系統專用函數。

EXPORT void FNBACK free_mouse_image_cursor(void);

功　能：釋放滑鼠的影像光標存儲區。

參　數：無

返回值：無

說　明：系統專用函數。

EXPORT SLONG FNBACK load_mouse_image_cursor(SLONG index,USTR *filename);

功　能：載入指定文件的動畫影像作為滑鼠的光標，並將該光標指定為特定的滑鼠類型。

參　數：SLONG index; //指定的滑鼠光標類型值

USTR filename; //動畫影像文件(*.CAK)

返回值：載入成功返回 TTN_OK，否則返回 TTN_NOT_OK。

說　明：

EXPORT SLONG FNBACK make_mouse_image_cursor(SLONG index, CAKE_FRAME_ANI *image_cfa, SLONG frames);

功　能：截取動畫的片段生成滑鼠動畫影像光標，並將該光標指定為特定的滑鼠型。

參　數：SLONG index; //滑鼠類型值

CAKE_FRAME_ANI *image_cfa; //動畫影像指針，必須保證本指針的所有節點(包含頭節點)均包含影像資料

SLONG frames; //滑鼠動畫的幀數

返回值：成功返回 TTN_OK，否則返回 TTN_NOT_OK。

說　明：本函數將依據 image_cfa 後的 frames 幀影像產生動畫影像光標。

EXPORT void FNBACK redraw_mouse_image_cursor(char *pbuffer,long pitch,long width,long height);

功　能：顯示滑鼠動畫影像資料到指定的存儲區。在上圖控制模塊的內核中，會用到本函數。

參　數：

返回值：

說　明：系統專用函數。

EXPORT SHINT FNBACK fnCheckCtrlKey(void);
功　能：檢查是否按下了 Ctrl 鍵。
參　數：無
返回值：按下了 Ctrl 時返回 1，否則返回 0。
說　明：

EXPORT SHINT FNBACK fnCheckLeftCtrlKey(void);
功　能：檢查是否按下了左 Ctrl 鍵。
參　數：無
返回值：如果按下了，返回 1，否則返回 0。
說　明：

EXPORT SHINT FNBACK fnCheckRightCtrlKey(void);
功　能：檢查是否按下了右 Ctrl 鍵。
參　數：無
返回值：如果按下了，返回 1，否則返回 0。
說　明：

EXPORT SHINT FNBACK fnCheckAltKey(void);
功　能：檢查是否按下了 Alt 鍵。
參　數：無
返回值：如果按下了，返回 1，否則返回 0。
說　明：

EXPORT SHINT FNBACK fnCheckLeftAltKey(void);
功　能：檢查是否按下了左 Alt 鍵。
參　數：無
返回值：如果按下了，返回 1，否則返回 0。
說　明：

EXPORT SHINT FNBACK fnCheckRightAltKey(void);
功　能：檢查是否按下了右 Alt 鍵。
參　數：無
返回值：如果按下了，返回 1，否則返回 0。
說　明：

EXPORT SHINT FNBACK fnCheckShiftKey(void);

功　能：檢查是否按下了 Shift 鍵。

參　數：無

返回值：如果按下了，返回 1，否則返回 0。

說　明：

EXPORT SHINT FNBACK fnCheckLeftShiftKey(void);

功　能：檢查是否按下了左 Shift 鍵。

參　數：無

返回值：如果按下了，返回 1，否則返回 0。

說　明：

EXPORT SHINT FNBACK fnCheckRightShiftKey(void);

功　能：檢查是否按下了右 Shift 鍵。

參　數：無

返回值：如果按下了，返回 1，否則返回 0。

說　明：

引用提示

與輸入控制模塊相關的文件為 xinput.cpp 和 xinput.h。

(九) **網路控制模塊(DirectPlay)**

模塊功能

本模塊利用 DirectPlay 作為底層，提供應用程式選擇連線類型，創建會話，加入會話，接收消息等介面。

數據介面

目前沒有提供任何外部數據介面。

函數介面

```
EXPORT    int        FNBACK    init_net(void);
EXPORT    void       FNBACK    free_net(void);
EXPORT    void       FNBACK    active_net(int active);
```

以上三個函數提供模塊裝載介面。

EXPORT void FNBACK set_net_func(FUNCDOMSG *do_sys, FUNCDOMSG *do_app);

功　能：設定處理消息的函數。

參　數：FUNCDOMSG *do_sys; //處理系統消息的函數
FUNCDOMSG *do_app; //處理應用程式的函數
FUNCDOMSG 的定義如下：

```
typedef void  (FUNCDOMSG)(DPMSG_GENERIC* pMsg, DWORD dwMsgSize,
    DPID idFrom, DPID idTo );
```

其參數分別為指向消息數據的指針，消息數據大小，發送消息的玩家的網路 ID，接收消息的玩家的網路 ID。

返回值：無

說　明：當我們裝載完網路控制模塊後，我們需要設定處理消息的函數以便在我們接收到網路消息後對其進行處理。
在 RAYSSDK 中，網路消息被分成兩類，一類為 DirectPlay 的系統消息，另外一類為應用程式自己定義的消息。
接收的網路消息有：

DPSYS_CHAT //接收到一個聊天訊息
DPSYS_CREATEPLAYERORGROUP　//創建了一個新的玩家或者組，當有新的玩家加入遊戲時，已經在線的玩家會收到此消息

DPSYS_DESTROYPLAYERORGROUP　//刪除一個玩家或者組，當有玩家離開遊戲時，其他玩家會收到此消息

DPSYS_HOST //本機成為主機

DPSYS_SESSIONLOST //網路會話丟失

DPSYS_ADDPLAYERTOGROUP //添加一個新的玩家到組中

DPSYS_DELETEPLAYERFROMGROUP //從組中刪除一個玩家

DPSYS_SETPLAYERORGROUPDATA //設定玩家或組的資訊

DPSYS_SETPLAYERORGROUPNAME //設定玩家或組的名稱

DPSYS_SETSESSIONDESC //設定會話的資訊

DPSYS_SENDCOMPLETE //傳送資料完成

等等。

而對於應用程式的消息，由用戶自己定義。

範　例：典型用法範例如下。

假如我們有以下兩個處理網路消息的函數，

```
void do_system_message(DPMSG_GENERIC* pMsg, DWORD dwMsgSize, DPID
idFrom, DPID idTo )
{
switch(pMsg->dwType)
{
    case DPSYS_CREATEPLAYERORGROUP:
//處理創建新的玩家或組
        break;
    case DPSYS_DESTROYPLAYERORGROUP:
//處理刪除一個玩家或組
        break;
//其他處理代碼
//(在此省略)
    }
}

void do_application_message(DPMSG_GENERIC* pMsg, DWORD dwMsgSize, DPID
idFrom, DPID idTo)
{
    switch(pMsg->dwType)
    {
```

```
//處理用戶自己定義的網路消息
        }
}
```

那麼，我們在裝載完網路控制模塊後，便可以使用以下方式來設定處理消息的函數指針了。

```
set_net_func(do_system_message, do_application_message);
```

ＯＫ，這樣我們便可以專心地設計我們處理消息的函數了。其他事情？RAYSSDK 已經幫我們做好了。

需要注意的是，我們自己定義的網路消息，需要帶有一個 dwType 的數值，以作為消息類型的標示，同時也可以做到與 DirectPlay 系統的相容性。

例如我們的消息可以這樣定義：

```
typedef struct    MSG_SET_EXP
{
    DWORD     dwType;
    SLONG     exp;
} MSG_SET_EXP;

typedef struct    MSG_DEL_ITEM
{
    DWORD     dwType;
    SLONG     map_no;
    SLONG     x;
    SLONG     y;
    ULONG     contain;
} MSG_DEL_ITEM;

typedef struct    MSG_REQUEST_ADD_BASE_RESP
{
    DWORD     dwType;
    SLONG     use_skill;
    SLONG     add_data;
} MSG_REQUEST_ADD_BASE_RESP;
```

EXPORT int FNBACK NET_get_connects(int ID, DPCNAME **lpname, int *count);

功　能：獲取系統可以使用的連線類型列表。

參　數：int ID; //網路 GUID 的附加值，一般取 0。

DPCNAME **lpname; //存放獲取的網路連線類型數據

連線類型數據定義如下：

```
typedef struct DPCNAME_STRUCT DPCNAME,LPDPCNAME;
struct DPCNAME_STRUCT
{
    char    *name; //連接類型名稱
    int     type;  //連接類型的類型定義
};
```

連線類型的類型定義如下：

NETCONNECTIPX //IPX 連線

NETCONNECTTCPIP //TCPIP 連線

NETCONNECTMODEM //數據機連線

NETCONNECTSERIAL //串口連線

NETCONNECTOTHER //其他類型連線

int *count; //存放獲取到的連線類型總數

返回值：列表如下。

NET_DOK //獲取連線成功

NETERR_INIT //網路控制沒有初始化

NETERR_DNODXX //創建 DirectPlay 或 DirectPlayLobby 對象錯誤

NETERR_DENUMERR //枚舉連線類型錯誤

說　明：當我們獲取了連線類型列表後，可以使用 NET_set_connect(num)來設定我們希望採用哪種方式來建立網路連線。num 為該列表的索引值。

EXPORT int FNBACK NET_set_connect(int num);

功　能：設定連線類型

參　數：int num; //連線類型數據的索引

返回值：如果設定成功，返回０；否則，為其他數。

說　明：

EXPORT int FNBACK NET_off_connect(void);

功　能：釋放網路連線。

參　數：無

返回值：成功後返回０，否則為其他數。

說　明：直接調用 DirectPlay 的 Release 釋放 DirectPlay 對象。

EXPORT void FNBACK NET_set_phone(char *phone);

功　能：設定採用 MODEM 連線時的電話號碼。

參　數：char *phone; //電話號碼字串，如"5186890"

返回值：無

說　明：

EXPORT void FNBACK NET_set_ip_address(char *address);

功　能：設定採用 TCPIP 連線時的 IP 地址。

參　數：char *address; //IP 地址，如"192.168.10.146"

返回值：無

說　明：

EXPORT void FNBACK NET_set_com_port(int port,int speed);

功　能：設定採用串口連線時的埠號和串列傳輸速率。

參　數：SLONG port; //埠號

int speed; //串列傳輸速率

返回值：

說　明：

EXPORT int FNBACK NET_get_sessions(char ***lpname, int *count);

功　能：取得當前連線中的會話列表。

參　數：char ***lpname; //存放會話名稱列表

int *count; //存放總的會話數

返回值：成功後返回０，否則為其他值。

說　明：

EXPORT int FNBACK NET_create_session(char *sname, int max_player, DPID *play_id, char *names, char *namel);

功　能：創建一個新的會話。

參　數：char *sname; //設定會話名稱
int max_player; //設定容許的最多玩家數目
DPID *play_id; //存放玩家的網路 ID
char *names; //設定玩家的暱稱(ShortName)
char *namel; //設定玩家的名稱(LongName)

返回值：成功後返回０，否則為其他值。

說　明：當我們創建一個新的會話時，系統同時會創建一個新的玩家，將該玩家的網路識別 ID 存放在*play_id 中，同時將該玩家加入到該會話並作為主機。

EXPORT int FNBACK NET_join_session(int num, DPID *play_id, char *names, char *namel);

功　能：加入指定的會話。

參　數：int num; //會話列表索引
DPID *play_id; //存放玩家的網路 ID
char *names; //設定玩家的暱稱(ShortName)
char *namel; //設定玩家的名稱(LongName)

返回值：成功後返回０，否則為其他值。

說　明：當我們加入一個會話時，和創建會話 NET_create_session()類似，系統同時也會創建一個新的玩家，將該玩家的網路識別 ID 存放在*play_id 中。

EXPORT int FNBACK NET_session_name(char *name);

功　能：設定會話的名稱。

參　數：char *name; //新的會話名稱

返回值：成功後返回０，否則為其他值。

說　明：只有會話的創建者才可以設定會話的名稱。

EXPORT int FNBACK NET_send(NETMSG *netmsg);

功　能：發送網路消息。

參　數：NETMSG *netmsg; //存放需要發送的消息
NETMSG 的定義如下：

```
struct  NETMSG_STRUCT
{
    DWORD       dwType;         //消息類型
    union
```

```
    {
     WORD     Flags;
     DWORD     size;                  //消息長度
     };
  PLAYERID      idFrom;               //發送消息的玩家的 ID
  LAYERID       idTo;                 //接收消息的玩家的 ID
  DWORD           dwCurrentPlayers;
  union
     {
     LPVOID    data;                  //消息數據的位址
     char *    message;              //以 0 結束的字串
     char *    ShortName;
     };
  union
     {
     char *    LongName;
     LPDPSESSIONDESC2 desc;
     };
  };
```

返回值：發送成功後返回 0 ，否則為其他值。

說　明：為保持相容性，RAYSSDK 中保留了這個函數。建議並強烈建議在發送消息時，使用 NET_send_data(LPVOID lpBuffer, DWORD dwSize, DPID idFrom, DPID idTo)函數。提請注意。

EXPORT　int　　FNBACK　NET_send_data(LPVOID lpBuffer, DWORD dwSize, DPID idFrom, DPID idTo);

功　能：發送網路數據消息。

參　數：LPVOID lpBuffer; //消息數據位址
　　DWORD dwSize; //消息數據的長度
　　DPID idFrom; //發送消息的玩家的網路 ID
　　DPID idTo; //接收消息的玩家的網路 ID

返回值：發送成功後返回 0 ，否則為其他值。

說　明：

EXPORT int FNBACK NET_send_chat(NETMSG *netmsg);

功　能：發送聊天訊息。

參　數：NETMSG *netmsg; //存放聊天訊息的消息結構。

返回值：成功後返回０，否則為其他值。

說　明：為保持相容性，RAYSSDK 中保留了本函數。強烈建議使用 NET_send_chat(LPVOID lpBuffer, DWORD dwSize, DPID idFrom, DPID idTo)來發送聊天訊息。

EXPORT int FNBACK NET_send_chat(LPVOID lpBuffer, DWORD dwSize, DPID idFrom, DPID idTo);

功　能：發送聊天訊息。

參　數：LPVOID lpBuffer; //聊天訊息的存放地址
DWORD dwSize; //聊天訊息的大小
DPID idFrom; //發送訊息的玩家網路 ID
DPID idTo; //接收訊息的玩家網路 ID

返回值：成功後返回０，否則為其他值。

說　明：

EXPORT int FNBACK NET_close(void);

功　能：關閉網路連線。

參　數：無

返回值：成功後返回０，否則為其他值。

說　明：關閉網路連線的同時，系統會刪除玩家(創建或加入連線時創建的)，並且調用 DirectPlay 的 Close 介面關閉 DirectPlay。

EXPORT int FNBACK NET_end_join(void);

功　能：終止其他新的玩家加入會話。

參　數：無

返回值：成功後返回０，否則為其他值。

說　明：通過改變會話的相關參數，終止其他新的玩家加入本會話。注意，只有創建會話的連線端可以調用本函數。

EXPORT int FNBACK NET_enum_players(int num,ENUMPLAYER *p);

功　能：枚舉指定會話中的所有玩家。

參　數：int num; //會話列表的索引值
ENUMPLAYER *p; //枚舉回調函數指針
回調函數的定義為

typedef void (ENUMPLAYER)(DPID id, char *names, char *namel);

對應參數分別代表該玩家的網路 ID，暱稱(ShortName)，長名稱(LongName)。

返回值：如果枚舉成功，返回０，否則為其他值。

說　明：用戶可以設定自己的回調函數，以便於作統計等。有關回調函數的詳細原理請參考 DirectX 的開發手冊。

EXPORT int FNBACK NET_player_name(char *names,char *namel);

功　能：設定玩家的名稱。

參　數：char *names; //暱稱(ShortName)

char *namel; //長名稱(LongName)

返回值：成功後返回０，否則為其他值。

說　明：

EXPORT void FNBACK NET_get_player_name(DPID player_id,char **friendlyname,char **formalname);

功　能：獲取指定玩家的名稱。

參　數：DPID player_id; //玩家的網路 ID

char **friendlyname; //獲取的該玩家的暱稱

char **formalname; //獲取的該玩家的正式名稱

返回值：無

說　明：

引用提示

相關文件為 xplay.cpp 和 xplay.h。

(十) CDROM 音軌控制模塊(CD Music)

模塊功能

CDROM 音軌播放模塊利用 WINDOWS 的 MCI 函數作為底層，提供給應用程式用戶一個方便的應用介面。

利用本模塊，用戶可以方便的實現播放 CD 音樂，停止播放 CD 音樂，暫停，繼續播放等等功能。

數據介面

目前沒有任何外部數據介面。

函數介面

```
EXPORT int    FNBACK init_cdrom_music(void);
EXPORT void FNBACK free_cdrom_music(void);
EXPORT void FNBACK active_cdrom_music(int active);
```

以上三個函數為 CDROM 音軌控制模塊的模塊裝載函數。

EXPORT void FNBACK pause_cdrom_music(void);

功　能：暫停播放 CDROM 音樂。
參　數：無
返回值：無
說　明：通過調用 resume_cdrom_music()可以從暫停的位置開始，繼續播放 CDROM 音樂。

EXPORT void FNBACK resume_cdrom_music(void);

功　能：繼續播放 CDROM 音樂。
參　數：無
返回值：無
說　明：與 pause_cdrom_music()配合使用。

EXPORT void FNBACK stop_cdrom_music(void);
功　能：停止播放 CDROM 音樂。
參　數：無
返回值：無
說　明：

EXPORT void FNBACK play_cdrom_music(SLONG track);
功　能：播放指定音軌的音樂。
參　數：SLONG track; //音軌編號，從 0 開始。
返回值：無
說　明：注意，此函數會從頭開始播放該音軌。

EXPORT SLONG FNBACK status_cdrom_music(void);
功　能：檢查 CDROM 音樂的播放狀態。
參　數：無
返回值：如果 CDROM 音樂已經停止，返回 0 ；否則為其他值。
說　明：

EXPORT void FNBACK loop_cdrom_music(void);
功　能：循環播放 CDROM 音樂。
參　數：無
返回值：無
說　明：

如果我們需要循環播放 CDROM 音樂，只需要在我們的程式中不斷地，定時地調用本函數。這樣，我們便可以循環地播放當前的音樂。比較典型的方式是在類似 idle_loop()的函數中檢查。不過為了靈活性的考慮，目前 RAYSSDK 中提供的 idle_loop()函數中並沒有調用本函數。

事實上建議使用 MP3 或者 WAV 來作為我們遊戲的音樂。使用 WAV 的方式在聲音控制模塊中有對應的函數實現。使用 MP3 的方式請參閱 MP3 控制模塊。

EXPORT SLONG FNBACK total_cdrom_music(void);

功　能：獲取當前 CDROM 上的音軌數目。

參　數：無

返回值：返回獲取的音軌數。

說　明：

引用提示

相關文件為 xcdrom.cpp 和 xcdrom.h。

(十一) 漢字顯示模塊(Font)

模塊功能

對於漢字的顯示，以前使用的方法是直接用點陣字體文件，其優點是不管應用程式運行的操作系統平臺的版本是簡體中文版，繁體中文版或者英文版，均可以正常方便地顯示漢字(模塊名稱為 font)。

不過，為了與 WINDOWS 系統的最大相容性和可擴充性，目前使用的為基於操作系統的 TrueTypeFont(TTF)文字顯示(模塊名稱為 winfont)。

兩者的用法大體上是相同的。這裡以 winfont 模塊來加以說明。
建議在應用程式的發展是，用 winfont 模塊來作為顯示漢字的實現方式。

數據介面

本模塊目前沒有提供任何外部數據介面。

函數介面

```
int     init_winfont(void);
void    active_winfont(int active);
void    free_winfont(void);
```

以上三個函數為漢字顯示模塊的模塊裝載介面函數。

```
void    print_640x480x16M(SLONG x,SLONG y,USTR *data,SLONG
            put_type,BMP *bit_map);
void    print_range_640x480x16M(SLONG x,SLONG y,USTR *data,SLONG
            put_type,BMP *bit_map);
void    set_word_color(UHINT color);
void    set_back_color(UHINT color);
void    set_word_color(SLONG lab,UHINT color);
void    set_back_color(SLONG lab,UHINT color);
SLONG get_word_width(void);
```

以上函數為顯示漢字時會用到的應用函數。

不過，為了與 xfont 模塊的相容性，也方便應用程式以後運行於其他平臺的實現，這裡，強烈建議使用本模塊提供的宏定義來顯示漢字。以下將重點介紹這些宏定義的使用方法。

print16(x,y,str,type,bitmap)
print20(x,y,str,type,bitmap)
print24(x,y,str,type,bitmap)
print28(x,y,str,type,bitmap)
print32(x,y,str,type,bitmap)

它們實現漢字大小分別為 16X16，20X20，24X24，28X28，32X32 點的漢字顯示。

參數說明：

(SLONG) x 為字串顯示的起始點的 X 座標。
(SLONG) y 為字串顯示的起始點的 Y 座標。
(USTR*) str 為需要顯示的漢字串。
(SLONG) type 為顯示漢字的方式。
(BMP*) bitmap 為目的位元圖。

這裡有幾點需要重點說明：

(1) 漢字的對齊方式為左上角對齊。比如我們想將大小為 16 的漢字”希望”顯示在位圖 bitmap 的左上角(0,0)，並且保持該顯示的漢字的左上角於該點，那麼我們可以這樣來實現：

print16(0,0,"希望",PEST_PUT,bitmap);

(2) 顯示的漢字串中可以包含一些控制字，這些控制字可以實現諸如改變漢字顯示的前景色，改變漢字顯示的效果等。目前可以使用的控制字有：

~C0,~C1,~C2,~C3,~C4,~C5,~C6,~C7,~C8,~C9

將實現改變漢字顯示的前景色。C 代表 Color。後面的 0,1,..9 分別代表不同的顏色，如下所示：

0 SYSTEM_WHITE 白色
1 SYSTEM_RED 紅色
2 SYSTEM_GREEN 綠色
3 SYSTEM_BLUE 藍色
4 SYSTEM_YELLOW 黃色
5 SYSTEM_CYAN 青色
6 SYSTEM_PINK 粉紅色

7 SYSTEM_WHITE 白色，與 0 相同
8 SYSTEM_BLACK 黑色
9 SYSTEM_DARK0 灰色

~O0,~O1,~O2,~O3,~O4,~O5,~O6,~O7,~O8,~O9

將實現改變漢字顯示的描邊效果。其中 0 標示取消描邊；其他 1~9 代表不同的描邊顏色，顏色定義與上面的一樣。

(3)(SLONG)type 指定顯示漢字的方式。其取值如下定義：

PEST_PUT 將漢字以濾掉穿透色的方式(PEST)顯示
COPY_PUT 將漢字以直接 COPY 的方式顯示

或者是以下的宏

COPY_PUT_COLOR(a)

這個宏定義指定漢字顯示的背景色。宏參數 a 為某顏色值。鑒於相容不同的顯示卡，強烈建議該參數用系統色，如 SYSTEM_RED，SYSTEM_BLUE 等。系統色的定義請參考引擎的繪圖函數主題。

比如我們想要用藍色作為背景色顯示描有紅色邊框的白色漢字”希望”，我們可以這樣來使用：

print16(0,0, "~C0~O1 希望~C0~O0",COPY_PUT_COLOR(SYSTEM_BLUE),bitmap);

如果我們不指定背景色，我們可以這樣：

print16(0,0, "~C0~O1 希望~C0~O0",COPY_PUT,bitmap);

引用提示

與漢字顯示相關的文件為 xfont.cpp，xfont.h 和 winfont.cpp，winfont.h。

(十二) MP3 播放模塊(MP3)

模塊功能

本模塊實現對 MP3 格式的音樂文件進行動態解壓並播放。
需要注意的是，目前本模塊只支援 WINDOWS2000 下的正常工作。在下一步的版本中，將會更新，使之可以支援 WINDOWS98，WINDOWS95 等。

數據介面

暫時沒有任何對外的數據介面。

函數介面

```
EXPORT int    FNBACK init_mp3(void);
EXPORT void FNBACK active_mp3(int active);
EXPORT void FNBACK free_mp3(void);
```

以上三個函數提供 MP3 模塊的裝載介面。

EXPORT void FNBACK play_mp3(SLONG loop, USTR *filename);
功　能：播放 MP3 音樂。
參　數：SLONG loop; //循環播放標誌，1=循環播放，0=播放一次
　　USTR *filename; //MP3 文件的名稱
返回值：無
說　明：

EXPORT void FNBACK stop_mp3(void);
功　能：停止 MP3 音樂播放。
參　數：無
返回值：無
說　明：

EXPORT void FNBACK set_mp3_volume(SLONG volume);
功　能：設置播放 MP3 的音量。
參　數：SLONG volume; //指定的音量
　　請參考音樂控制模塊(xsound.h)中有關音量的定義。
返回值：無
說　明：

EXPORT void　FNBACK set_mp3_pan(SLONG pan);

功　能：設置播放 MP3 的均衡值。

參　數：SLONG pan; //指定的均衡值

　　請參考音樂控制模塊(xsound.h)中有關均衡值的定義。

返回值：無

說　明：

EXPORT SLONG FNBACK is_mp3_playing(void);

功　能：檢查 MP3 音樂是否正在播放中。

參　數：無

返回值：如果正在播放，返回 TRUE；否則，返回 FALSE。

說　明：

引用提示

　　與本模塊相關的文件為 mp3.h 以及其他一系列文件，在此就不一一列出了。

(十三) Windows BMP 圖像格式支援

文件格式介紹

BMP 為 WINDOWS 的位圖格式。
文件頭格式如下：

```
typedef    struct tagBITMAPHEADER
{
      UHINT     type; //
      ULONG    size; //
      ULONG    reserved; //
      ULONG    off_bits; //
      ULONG    head_size; //
      ULONG    width; //
      ULONG    height; //
      UHINT     planes; //
      UHINT     bit; //
} BITMAPHEADER,*LPBITMAPHEADER;
```

數據介面

暫時沒有提供任何對外數據介面。

函數介面

EXPORT BMP*　　FNBACK　BMP_load_file(char *filename);

功　能：載入一個 WINDOWS 的 BMP 位元圖文件到引擎的位元圖中。
參　數：char *filename; //WindowsBMP 位圖檔案名稱
返回值：返回一個指向引擎位元圖數據的指針。
　　有關引擎位元圖數據結構(BMP)請參考引擎圖像操作的位元圖操作部分。
說　明：

EXPORT SLONG　FNBACK　BMP_save_file(BMP *bmp, char *filename);

功　能：將引擎位元圖數據存儲為一個 WINDOWS 的 BMP 格式文件。
參　數：BMP *bmp; //指向引擎位元圖數據的指針
　　char *filename; //目標檔案名稱
返回值：成功後返回 0 ，否則為其他值。
說　明：

EXPORT SLONG　FNBACK　BMP_save_file(UHINT *buffer, SLONG xl, SLONG yl, char *filename);

功　能：將指定內存區中的數據存儲為 Windows BMP 格式文件。

參　數：UHINT *buffer; //指向影像數據緩衝區的指針

SLONG xl; //位圖寬度

SLONG yl; //位圖高度

char *filename; //目標檔案名

返回值：成功後返回０，否則為其他值。

說　明：

引用說明

相關文件為 vbmp.cpp 和 vbmp.h。

(十四) PCX 圖像格式支援

文件格式介紹

文件頭結構如下：

```
typedef struct tagPCX_HEAD
{
    UCHR    manufacturer;
    UCHR    version;
    UCHR    encoding;
    UCHR    bitsPerPixel;
    SHINT xMin;
    SHINT yMin;
    SHINT xMax;
    SHINT yMax;
    SHINT hResolution;
    SHINT vResolution;
    UCHR    palette[48];
    UCHR    videoMode;
    UCHR    colorPlanes;
    SHINT bytesPerLines;
    SHINT paletteType;
    SHINT shResolution;
    SHINT svResolution;
    UCHR    filler[54];
} PCX_HEAD,*LPPCX_HEAD;
```

數據介面

目前沒有提供任何外部數據介面。

函數介面

EXPORT BMP* FNBACK PCX_load_file(char *filename);

功　能：載入一個 PCX 圖像文件到引擎位元圖結構中。

參　數：char *filename; //PCX 圖像檔案名稱

返回值：返回一個指向引擎位元圖結構數據的指針。

　　引擎位元圖數據結構的定義請參考引擎圖像操作的位元圖操作部分。

說　明：

引用說明

對應文件為 vpcx.cpp 和 vpcx.h。

(十五) TGA 圖像格式支援

文件格式介紹

文件頭結構如下：

```
typedef struct tagTGA_HEAD
{
    UCHR    bIdSize;
    UCHR    bColorMapType;
    UCHR    bImageType;
    UHINT iColorMapStart;
    UHINT iColorMapLength;
    UCHR    bColorMapBits;
    UHINT ixStart;
    UHINT iyStart;
    UHINT iWidth;
    UHINT iHeight;
    UCHR    bBitsPerPixel;
    UCHR    bDescriptor;
} TGA_HEAD,*LPTGA_HEAD;
```

數據介面

目前沒有提供任何外部數據介面。

函數介面

EXPORT BMP* FNBACK TGA_load_file(char *filename);

功　能：載入 TGA 圖像到引擎位元圖中。

參　數：char *filename; //PCX 檔案名稱

返回值：返回指向引擎位元圖數據結構的指針。

說　明：在我們載入 TGA 圖像時，本函數會自動過濾 TGA 的 Alpha 通道，並將其轉成系統穿透色(純黑色)。

過濾的閥值可以通過 TGA_set_transparency_level()來設定。Alpha 大於該閥值的將保留原來的顏色，小於該閥值的將被換成系統穿透色。

不過，當我們需要載入帶有 Alpha 通道的 TGA 圖像時，為了保持原來 Alpha 的訊息，建議使用 TGA_load_file_with_alpha()函數。

EXPORT ABMP* FNBACK TGA_load_file_with_alpha(char *filename);

功　能：載入 TGA 圖像到引擎的 Alpha 位元圖中。

參　數：char *filename; //TGA 檔案名稱

返回值：返回指向引擎 Alpha 位元圖數據的指針。
引擎 Alpha 位元圖數據結構的定義請參考引擎圖像操作的位元圖操作部分。

說　明：

EXPORT void FNBACK TGA_set_transparency_level(SLONG level);

功　能：設置載入 TGA 時透明度閥值(Alpha 閥值)。

參　數：SLONG level; //Alpha 閥值，0~255

返回值：無

說　明：配合 TGA_load_file()使用。

引用說明

(十六) JPG 圖像格式支援

文件格式介紹

JPG 的格式比較複雜，請參考有關書籍。

數據介面

函數介面

EXPORT int FNBACK LoadJPG(const char *filename, unsigned char **pic, int *width, int *height);

功　能：載入 JPG 圖像到內存緩衝區中。

參　數：const char *filename; //JPG 檔案名稱
unsigned char **pic; //指向內存緩衝區指針的指針
int *width; //存放圖像寬度
int *height; //存放圖像高度

返回值：成功後返回 0 ，否則為其他值。

說　明：

載入完成後，對應內存緩衝區中的數據為經過還原的 JPG 圖像訊息。圖像每個圖元對應內存緩衝區中連續 4 個字節，依序分別為該圖元的 r,g,b,a 值。

我們可以將這些值經過運算合併，然後獲得適合需要的其他引擎位元圖數據。

範　例：下面舉一個例子來說明：

```
char * jpg_filename = "TEST.JPG";
BMP * fore_bitmap = NULL;
unsigned char *jpg_pic = NULL;
int jpg_width, jpg_height;
int x,y,offset;
unsigned char r,g,b,a;

//載入 JPG 圖像文件

if(0 == LoadJPG( (const char *)jpg_filename, &jpg_pic, &jpg_width, &jpg_height ) )
{
    fore_bitmap = create_bitmap(jpg_width, jpg_height);
```

```
    if(fore_bitmap)
        {
//將 jpg_pic 中的內容轉換到 fore_bitmap 中
//並將其轉成高彩(hi color)
        offset = 0;
        for(y=0; y<jpg_height; y++)
            {
            for(x=0; x<jpg_width; x++)
                {
                r = jpg_pic[offset + (x << 2) + 0];
                g = jpg_pic[offset + (x << 2) + 1];
                b = jpg_pic[offset + (x << 2) + 2];
                a = jpg_pic[offset + (x << 2) + 3];
                fore_bitmap->line[y][x] = rgb2hi(r,g,b);
                }
            offset += (jpg_width << 2);
            }
//釋放不再需要的 jpg_pic 所佔用的內存
        if(*jpg_pic)
            {
            free(*jpg_pic);
            *jpg_pic = NULL;
            }
//後續的對 fore_bitmap 操作
//?(在此省略)
        }
    }
```

EXPORT int FNBACK LoadImage(const char *name, byte **pic, int *width, int *height);

目前不建議使用本函數。

EXPORT int FNBACK LoadImageBuff(byte *buffer, byte **pic, int *width, int *height ,int flag);

目前不建議使用本函數。

引用說明

相關文件為 jpeg.h。

(十七) PhotoShop PSD 格式支援

文件格式介紹

關於 PSD 檔案格式，請參考有關書籍。

數據介面

函數介面

EXPORT　PSDFILE *　FNBACK　open_psd_file(USTR *filename,SLONG *layers,SLONG *xl,SLONG *yl);

功　能：打開一個 PSD 文件。

參　數：USTR *filename; //PSD 檔案名稱
　　SLONG *layers; //存放 PSD 文件的層數
　　SLONG *xl; //PSD 影像的寬度
　　SLONG *yl; //PSD 影像的高度

返回值：如果打開文件成功，返回一個指向結構類型為 PSDFILE 的指針。否則，返回 NULL。

說　明：

EXPORT　SLONG　FNBACK read_psd_layer_info(SLONG layer,SLONG *sx,SLONG *sy,SLONG *xl,SLONG *yl,PSDFILE *f);

功　能：從打開的 PSD 文件中讀取指定層的資訊。

參　數：SLONG layer; //指定要讀取的 PSD 的圖層
　　SLONG *sx; //該圖層影像的起始 x 座標
　　SLONG *sy; //該圖層影像的起始 y 座標
　　SLONG *xl; //該圖層影像的寬度
　　SLONG *yl; //該圖層影像的高度
　　PSDFILE *f; //PSDFILE 文件指針

返回值：如果讀取成功，返回 0；否則為其他值。

說　明：

EXPORT SLONG FNBACK read_psd_layer_image(SLONG layer,UHINT *buffer,USTR *alpha_buffer,PSDFILE *f);

功　能：從打開的 PSD 文件中讀取某指定圖層的影像。

參　數：SLONG layer; //指定圖層
UHINT *buffer; //存放影像的緩衝區
USTR *alpha_buffer; //存放影像 Alpha 的緩衝區
PSDFILE *f; //PSDFILE 文件指針

返回值：讀取成功後返回 0 ，否則返回其他值。

說　明：在讀取 PSD 文件中某圖層的影像時，我們必須先調用 read_psd_layer_info() 以獲得該圖層的資訊，然後分配一個存放影像的緩衝區 buffer 和一個存放影像 Alpha 的緩衝區 alpha_buffer，再調用本函數 read_psd_layer_image()來讀取之。

範　例：依次顯示 PSD 文件各圖層

```
PSDFILE *f=NULL;
SLONG psd_xl, psd_yl, psd_layers;
SLONG layer, layer_sx, layer_sy, layer_xl, layer_yl;
SLONG result;
UHINT *buffer=NULL;
USTR *alpha_buffer=NULL;
f = open_psd_file((USTR*)"TEST.PSD", &psd_layers, &psd_xl, &psd_yl);
if(NULL != f)
    {
    for(layer=0; layer<psd_layers; layer++)
        {
        result = read_psd_layer_info(layer, &layer_sx, &layer_sy, &layer_xl,
            &layer_yl);
        if(0 == result)
            {
            buffer = (UHINT*)malloc(layer_xl * layer_yl * sizeof(UHINT));
            alpha_buffer = (USTR*)malloc(layer_xl * layer_yl * sizeof(USTR));
            if(buffer && alpha_buffer)
                {
                result = read_psd_layer_image(layer, buffer, alpha_buffer, f);
                if(0 == result)
                    {
//show image here?
```

```
                    }
                }
            if(buffer) {free(buffer); buffer = NULL; }
            if(alpha_buffer) {free(alpha_buffer); alpha_buffer = NULL; }
            }
        }
    }
if(f) close_psd_file(f);
```

EXPORT SLONG FNBACK read_psd_graph_image(UHINT *buffer, PSDFILE *f);

功　能：讀取 PSD 文件的顯示 GRAPH 影像。

參　數：UHINT *buffer; //存放顯示影像的緩衝

PSDFILE *f; //PSDFILE 文件指針

返回值：讀取成功後返回 0 ，否則返回其他值。

說　明：

EXPORT void FNBACK close_psd_file(PSDFILE *f);

功　能：關閉打開的 PSD 文件。

參　數：PSDFILE *f; //PSD 文件指針

返回值：無

說　明：

EXPORT void FNBACK set_psd_transparency_level(SLONG level);

功　能：設定 PSD 文件讀取時的穿透閥值。

參　數：SLONG level; //閥值，0~255

返回值：無

說　明：

引用說明

相關文件為 vpsd.cpp 和 vpsd.h。

(十八) **FLC 動畫格式支援**

文件格式介紹

FLC 為早期 256 色時的動畫格式，在 AnimatorPro 中有非常廣泛的應用。其具體檔案格式請參考有關書籍。

數據介面

目前沒有任何外部數據介面。

函數介面

EXPORT SLONG FNBACK FLIC_open_flic_file(USTR *name,SLONG mode_flag,USTR *memory_plane);

功　能：打開一個 FLC 動畫文件。
參　數：USTR *name; //FLC 檔案名稱
SLONG mode_flag; //打開 FLC 的模式
OPEN_FLIC_MODE_640X480 以 640X480 的模式
OPEN_FLIC_MODE_320X200 以 320X200 的模式
USTR *memory_plan; //內存緩衝區
返回值：如果打開成功，返回 FLC 動畫的幀數；否則，返回 TTN_ERROR。
說　明：

EXPORT void FNBACK FLIC_close_flic_file(void);
功　能：關閉 FLC 文件。
參　數：無
返回值：無
說　明：

EXPORT SHINT FNBACK FLIC_read_flic_data(void);
功　能：讀取 FLC 的數據。
參　數：無
返回值：如果讀取成功，返回 0；否則，返回-1(比如在讀取 FLC 動畫的最後一幀後)。
說　明：

EXPORT SHINT FNBACK FLIC_play_flic_frame(void);

功　能：將讀取的數據解壓到指定的內存緩衝中。

參　數：無

返回值：返回 0 。

說　明：

範　例：讀取並顯示 FLC 的各幀。

```
SLONG total_frames, i, ret;
USTR *memory_plan = NULL;
USTR palette[768];
memory_plan = (USTR*)malloc(640*480);
if(memory_plan)
	{
	total_frames = FLIC_open_flic_file((USR*)"", OPEN_FLIC_MODE_640X480,
	memory_plan);
	for(i=0; i<total_frames; i++)
		{
		if(i == 0)
			{
			FLIC_export_palette(&palette[0]);
			}
		ret = FLIC_read_flic_data();
		if(ret == 0)
			{
			FLIC_play_flic_frame();
//convert memory plan to screen_buffer by palette
//and show screen_buffer to screen
			}
		else break;
		}
	FLIC_close_flic_file();
}

if(memory_plan) {free(memory_plan); memory_plan = NULL; }
```

EXPORT void FNBACK FLIC_export_palette(USTR *palette);

功　能：將 FLC 文件的色盤導出。

參　數：USTR *palette; //指向色盤緩衝區的指針

返回值：無

說　明：

引用說明

相關文件為 vflic.cpp 和 vflic.h。

(十九) **AVI 動畫格式支援**

文件格式介紹

AVI 為 WINDOWS 的動畫格式。在 RAYSSDK 中，利用 DirectMedia 來進行 AVI 文件的播放。

數據介面

extern RECT rcXMedia;
動畫影像的大小。
extern STREAM_TIME stStartTime;
動畫開始時間。
extern STREAM_TIME stEndTime;
動畫結束時間。
extern STREAM_TIME stCurrentTime;
動畫當前播放時間。
extern STREAM_TIME stDuration;
動畫的總播放時間長度。

函數介面

void set_xmedia_over_draw(PFNREDRAW my_redraw);
功 能：設定動畫播放上的
參 數：
返回值：
說 明：

void play_xmedia_movie(SLONG sx,SLONG sy,SLONG ex,SLONG ey,USTR *filename,SLONG wait_flag);
功 能：
參 數：
返回值：
說 明：

SLONG　　open_xmedia_file(USTR *filename,RECT *rect);

功　能：

參　數：

返回值：

說　明：

SLONG　　play_xmedia_frame(BMP *bitmap,SLONG left_top_flag);

功　能：

參　數：

返回值：

說　明：

SLONG　　close_xmedia_file(void);

功　能：

參　數：

返回值：

說　明：

void　　seek_xmedia_file(STREAM_TIME st);

功　能：

參　數：

返回值：

說　明：

引用說明

相關文件為 xmedia.cpp 和 xmedia.h。