SDK instructions and example usage

(1) RAYSSDK includes a list of files

(2) Module Management Mechanism

(3) Module design

(Four) **System Control Module (System)**

(five) **Clock Control Module (Timer)**

(six) **The control module in the above figure (DirectDraw)**

(seven) **Sound Control Module (DirectSound)**

(eight) **Input Control Module (DirectInput)**

(Nine) **Network Control Module (DirectPlay)**

(ten) **CDROM Audio Track Control Module (CD Music)**

(eleven) **Chinese character display module (Font)**

(twelve)**MP3 player module (MP3)**

(Thirteen) **Windows BMP image format support**

(fourteen) **PCX image format support**

(fifteen) **TGA image format support**

(sixteen) **JPG image format support**

(Seventeen) **PhotoShop PSD format support**

(eighteen) **FLC animation format support**

(nineteen) **AVI animation format support**

(one) **RAYSSDK includes a list of files**

The basic list is as follows:

[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 playback control module

packfile.h

polling.h

rays.h // System header file

readinfo.h

ripple.h

topo.h

undo.h

vbmp.h

vflic.h

vpcx.h

vpsd.h

vtga.h

winfont.h // Chinese character display module

winmain.h

xcak.h

xcdrom.h // CD audio track driver control module

xdraw.h // The control module in the above figure

xfont.h // Chinese character display module

xgrafx.h

xinput.h // Input control module

xkiss.h

xmedia.h

xmem.h

xmodule.h // Module Management System

xplay.h // Network control module

xpoly.h

xrle.h

xsound.h // Sound control module

xsystem.h // System control module

xtimer.h //Clock control module

xvga.h

xwavread.h

ybitio.h

ylzss.h

(two) Module management mechanism

Standards and Principles

In RAYSSDK, all functions or interfaces that can be integrated are designed as modules for easy calling.

In order to uniformly manage all modules in the system, RAYSSDK requires each module to provide three functions:

1) Initialize the function of the module

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

2) Release the module's function

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

3) The module responds to the application startup or not processing function

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

The system supports loading up to 64 different modules at the same time. Please note that we cannot load the same module multiple times at the same time, otherwise it will cause unexpected errors.

In the RAYSSDK kernel, there is a module management mechanism that performs the following three tasks:

1) When a user installs a module, the module's initialization function is automatically executed and returns a success or failure value. 2) When the application window is activated or canceled, the module's response function is automatically called.

3) When the application is about to exit, the release function of the module is automatically executed.

So, how does the system achieve these tasks?

First, the system creates an array in memory to store pointers to the release functions of all system modules. It also creates an array to store pointers to the activation/disable functions of all modules. Of course, we don't need to create an array to store pointers to each module's initialization function because, as mentioned above, the module's initialization function is already called when the user installs the module.

With the module array that responds to application startup or failure, we can easily implement the correct response processing function. After the system receives the application startup or failure message, it executes the module response function in sequence (we have already stored the corresponding function pointer!), OK!

Similarly, when the application exits, we can execute the module release function in sequence.

Function interface

EXPORT void FNBACK init_modules(void);

Initializes the module and loads system data. This function is called internally by the system.

EXPORT int FNBACK install_module(MODULE_INIT_FUNC init,MODULE_FREE_FUNC free,MODULE_ACTIVE_FUNC active); Install a module. At the same time, we can find that in xmodule.h, there is a very useful macro definition

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

This macro definition can be used very conveniently to implement this function.

Function parameter types:

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

Parameter Description:

MODULE_INIT_FUNC init; //The module initialization function

MODULE_FREE_FUNC free; //The module release function

MODULE_ACTIVE_FUNC active; //This module responds to application startup processing function EXPORT void FNBACK free_modules(void);

Releases all modules in the system. This function is called internally by the system.

EXPORT void FNBACK active_modules(int bActive);

Causes all modules in the system to respond to changes in the application's enabled or disabled state. This function is called internally by the system.

Example Description

OK, let's summarize. To keep the API easy and simple, users only need to use one function: the one that installs the module. Even that can be done with a simple macro call. Here are some code snippets to illustrate:

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

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

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

This snippet sequentially loads the system module, the module shown above, and the clock module. If any module loading fails, a fatal error message pops up, terminating the application.

(three) **Module design**

Design basis

If we need to design a new module, we can easily install and manage the module through the module loading system.

According to the working principle and calling function of the module loading system, we need to prepare three basic module interface functions. Then, based on this, we can develop other corresponding functions of the module.

Example Description

For the sake of explanation, let's assume we want to create a test module called test. Then the functions we need are:

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

Then, there may be some other related functions of this module such as:

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

etc.

In this way, we can load the test module by calling the install(test) macro, and then use functions such as show_test() and log_test() during program execution.

Easy? You can easily design your own modules.

(Four) **System Control Module (System)**

Provides some system functions, including error log files, system detection functions, etc. At the same time, it provides some data interfaces to facilitate application calls.

Data interface

extern USTR print_rec[2048];

A temporary string buffer. In applications, we often use some string processing, small character array processing, etc., which can be applied to this buffer.

extern USTR game_path[_MAX_PATH];

Record the game running path. When installing the system module, the running path of this application will be detected and saved in this data area.
During application development, we may need to know the directory where the game is currently running. In this case, we can obtain it through this data interface.

extern USTR game_filename[_MAX_PATH+_MAX_FNAME];

A temporary buffer for storing file names. In applications, we often operate on file names, such as changing file extensions, automatically generating sequence file names, etc. In this case, we can use this buffer.

extern ULONG game_capture_no;

When a screenshot is saved as an image file, the file's serial number is usually reflected in the last few characters of the file name.

When the system module is initialized, this number is set to 0. Each subsequent screenshot is taken, and this number increments by 1. In your application, you can directly change this value to specify the file name for the screenshot storage image.

extern ULONG game_now_time;

Records the current system time. This data is often used in conjunction with the system time at the start of the game to determine how long the game has been running. Additionally, this time can be used to perform many other tasks, such as determining object or NPC spawn times.

It should be noted that this data is constantly updated during the game. The time unit is 1 millisecond, which is one thousandth of a second.

extern ULONG game_start_time;

Records the system time when the game starts. This data is often combined with the current system time above to use.

When the system module is initialized, the system time is obtained and saved in this data. This data remains unchanged while the game is running.

Function interface

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

The above three functions are the module management interface functions of the system control module.

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

Function: Stores information in a disk file. This is typically used to write error messages to a disk file when an error occurs, to facilitate debug.

Parameters: int p; // Judgment flag, when it is 0, it will return directly; otherwise, it will write the message.
USTR *strMsg; //The string message to be written. Return value: None

say bright:

Fan example:

```
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, ...);

Function: Stores information in a disk file. This is typically used to write error messages to a disk file when an error occurs, to facilitate debug.

Parameters: int p; // Judgment flag, when it is 0, it will return directly; otherwise, it will write the message.

char *strMsg; //Format definition string...; //

Other related parameters

Return value: None

Description: None

EXPORT void FNBACK idle_loop(void);

achievement Ability: Receive and process Windows messages.

ginseng Number: None

Return value: None

Note: When your application is executing a time-consuming loop or needs to receive Windows messages within the loop, you must call this function at the beginning of the loop. Otherwise, the application will occupy Windows system time for a long time, causing the system to stall. Alternatively, Windows messages may not be received within the loop.

Example:

```
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);

achievement: Yes: Check whether the operating system is Simplified Windows.

ginseng Number: None

Return value: If it is Simplified Windows, returns TRUE; otherwise, returns FALSE.

Description: None

EXPORT void FNBACK run_random_init(void);

achievement: Function: Initialize random numbers.

ginseng Number: None

Return value: None

Note: When the application is running, the value of the random number sequence is fixed. To obtain a different random number sequence each time, we need to call this function to initialize the random number sequence.

EXPORT SLONG FNBACK get_cdrom_drive(void);

achievement: Function: Gets the CDROM drive number and returns a number representing the CDROM drive number.

ginseng Number: None

Return value: If no CDROM drive is found, -1 will be returned; otherwise a non-negative number will be returned.

Different numbers represent different disk drives, (0=A:; 1=B:; 2=C:;) 3=D:; 4=E; etc. And so on.

Description: None

EXPORT SLONG FNBACK check_cdrom_volume(USTR *title);

achievement: Yes: Get a CDROM drive that matches the corresponding volume label.

ginseng Number: USTR *title; //label string

Return value: If no match is found, returns -1; otherwise, returns a non-negative number.

Different numbers represent different disk drives, (0=A:; 1=B:; 2=C:;) 3=D:; 4=E; etc. And so on.

Description: None

EXPORT USTR * FNBACK get_cdrom_volume(SLONG drive);

achievement: Yes: Get the volume label of the specified CDROM.

ginseng Number: SLONG drive; //CDROM drive number, 0=A:; 1=B;...

Return value: USTR * string, which is the obtained label. If unsuccessful, it is NULL.

Description: None

EXPORT void FNBACK store_game_path(USTR *path);

achievement Function: Store the current game running path.

ginseng Number: USTR *path; //Pointer to store the path

Return value: None

Description: Function used internally by the system.

EXPORT void FNBACK capture_screen(void);

achievement Can: Capture the screen and save it as an image file.

ginseng Number: None

Return value: None

Description: None

EXPORT ULONG FNBACK get_fps(void);

achievement Function: Get the current speed of uploading the image.

ginseng Number: None

Return Value: Returns a number representing FPS (Frames Per Second). Note: This function must be called once in each image upload loop to obtain the correct image upload speed.

EXPORT USTR * FNBACK get_computer_name(void);

achievement Ability: Get computer name

ginseng Number: None

Return value: computer name

Description: None

EXPORT USTR * FNBACK get_user_name(void);

achievement Can: Get the current Windows user name

ginseng Number: None

Return value: Windows user name

Description: None

EXPORT USTR * FNBACK get_windows_directory(void);

achievement Yes: Get the WINDOWS directory.

ginseng Number: None

Return value: WINDOWS system directory.

Description: None

EXPORT SLONG FNBACK get_windows_version(void);

achievement Yes: Get the WINDOWS version.

ginseng Number: None

Return value: as shown below

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

Description: None

EXPORT void FNBACK get_memory_status(ULONG *total_phys,ULONG * avail_phys);

achievement Function: Get the current system physical memory usage.

ginseng Number: ULONG *total_phys; //Pointer to store the total physical memory size

ULONG *avail_phys; //Pointer to the currently available physical memory size Return value: None

Description: None

EXPORT ULONG FNBACK get_disk_serial_no(void);

achievement Yes: Get the serial number of the C: drive.

ginseng Number: None

Return value: the value of the serial code.

Description: None

EXPORT ULONG FNBACK get_cpu_clock(void);

achievement Ability: Get CPU clock cycles.

ginseng Number: None

Return value: CPU clock cycle, unit is MHZ.

Description: None

EXPORT char * FNBACK get_cpu_id(void);

achievement Function: Get the CPU ID. Number:

ginseng None

Return value: identification string.

Description: None

EXPORT void FNBACK analyst_system(void);

achievement Capability: Analytical system.

ginseng Number: None

Return value: None

Description: Analyzes the system's CPU, memory, operating system, etc., and writes these to the error log file.

Citation Tips

In RAYSSDK, the files related to the system module are xsystem.cpp and xsystem.h. When using the generated LIB, we only need to include the header file xsystem.h.

(five) **Clock Control Module (Timer)**

Module Function

Provides a relatively accurate clock (Timer) to applications to facilitate timing control within the application. The clock unit provided by this module is 1/100 second, or 10 milliseconds.

This module provides multiple timing data, and users can obtain or change the value of the data. Every time the system time passes 1/100 second, these timing data will increase by 1. This action is completed by the clock control module itself.

Data interface

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; The above clock data can be referenced by users.

extern ULONG cdrom_timer_tick; The system reserves the clock, which is used to control CDROM music playback.

extern ULONG cursor_timer_tick; System reserved clock, specifically used to control the refresh timing of the graphical animation cursor.

extern ULONG system_timer_tick;
System reserved clock.

Function interface

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

The above three functions are the module management interface functions of the clock control module.

When we need to use the data interface of this module, we must first load the module. We can easily do this through the module management macro install(timer).

(six) **The control module in the above figure (DirectDraw)**

Module Function

The control module in the above figure uses DirectDraw as the underlying layer, providing an interface for applications to directly operate the display screen.

Its basic actions include initializing DirectDraw, setting the cooperation level, creating the screen buffer and backup buffer, setting the image rendering mode, etc. At the same time, it obtains the 16-bit color type of the graphics card (555, 565, etc.) and sets a series of image rendering functions based on this type.

To improve the speed of application graphics upload and bitmap manipulation, the RAYSSDK provides a series of drawing functions corresponding to different graphics cards. These functions are also provided as pointers. After obtaining the graphics card type, the upload control module sets these pointers to the corresponding function instances. These function pointers are included in the engine's bitmap manipulation function set, and the relevant files are xgrafx.cpp and xgrafx.h.

Data interface

extern LPDIRECTDRAW7 lpDD7; /* system directdraw object */

DirectDraw object. Initialized when the graphics control module is loaded.

This data is for system use only. Applications are advised not to manipulate it.

extern LPDIRECTDRAWSURFACE7 lpDDSPrimary7;/* system directdraw primary surface*/

The primary screen buffer is created in the display memory and corresponds to the client area of the game display window.

This data is for system use only. Applications are advised not to manipulate it.

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

The screen backup buffer is created in the display memory. To achieve smooth loading and avoid screen tearing when uploading images in DirectDraw, the uploading control module obtains a pointer to the screen backup buffer, operates on it, and then performs a flipping operation to upload the image.

This data is for system use only. Applications are advised not to manipulate it.

extern LPDIRECTDRAWSURFACE7 lpDDSMemory7; /* memory directdraw surface */ Create a buffer in the system memory.

The system uses this buffer to play AVI animation files (DirectMedia).

Applications should not operate on it.

extern ULONG nBackBuffers; /* system directdraw back surface count */ The number of screen back buffers.

When the image loading control module is initialized, it creates an appropriate number of screen backup buffers based on the amount of video memory on the machine's graphics card to increase image loading speed. This data records the number of backup buffers created.

extern SLONG vga_type; /* system video card type */

Graphics card 16-bit color type.

When the module above is initialized, it obtains the 16-bit color type (PixelFormat) of the machine's display card and records the type in this data.

The possible values for data are:

VGA_TYPE_555 //555 type graphics card
VGA_TYPE_655 //655 type graphics card
VGA_TYPE_565 //565 type graphics card
VGA_TYPE_556 //556 type graphics card
VGA_TYPE_ANY //Other types of graphics cards

Function interface

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

The above three functions are the module management interface functions of the control module in the figure above.

When we need to use the data interface of this module, we must first load the module. We can easily do this through the module management macro install(draw).

EXPORT void FNBACK set_update_area(int start,int height);

achievement Yes: Set the area of the screen to update.

ginseng Number: int start; //Update the starting line of the screen, initialized to 0.

int height; //Update the number of screen rows, initialized to SCREEN_HEIGHT. Return value: None

instruction:

EXPORT void FNBACK set_update_type(int type);

achievement Yes: Set the method for updating the screen.

ginseng Number: int type; //Specify how to update the screen

Its value can be one of the following:

NORMAL_UPDATE_SCREEN //Update the screen in the normal way. PEST_UPDATE_SCREEN //Update the screen using the PEST method, filtering out the transparent color 0x0000. Return value: None.

instruction:

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

achievement Function: Get BITMAP from the system memory buffer lpDDSMemory7. Parameters: BMP *bmp; //

ginseng Store the engine bitmap BMP structure pointer of the obtained BITMAP RECT rect; //Get the rectangular range

SLONG left_top_flag; //Whether to store data starting from the upper left corner of the bmp file. When this flag is 1, the acquired image data will be stored in the area starting from the upper left corner of the bmp file. When it is 0, the acquired image data will be stored in the area corresponding to the specified RECT in the bmp file. Return value: None

say Note: This function is used in RAYSSDK to control the playback of AVI files (DirectMedia).

Generally speaking, it is not recommended for applications.

Fan Example: To make it easier to understand, here are some examples:

```
RECT rc;
rc.left = 50;
rc.right = 150;
rc.top = 50;
rc.bottom = 150;
//(1) Get the 100x100 image data specified by (50,50)-(150,150) in lpDDSMemory7 and store it in the area (0,0)-(100,100) starting from the upper left corner of screen_channel0.
get_bitmap_from_memory_surface(screen_channel0, rc, 1);

//(2) Get the 100X100 image data specified by (50,50)-(150,150) in lpDDSMemory7 and store the data in the (50,50)-(150,150) area corresponding to RECT rc of screen_channel0.

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

EXPORT void FNBACK switch_screen_mode(void);

achievement Function: Switch the screen display mode to full screen mode or window mode. Number:

ginseng None

Return value: None

Description: While an application is running, the user can press F12 to instantly change the current screen display mode. This function is used by the system to achieve this.

This function is for system use and is not recommended for use in applications.

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

achievement Function: Set the color mode and related functions of the image card.

ginseng Number: DWORD dwRBitMask; //Red component mask of the primitive

DWORD dwGBitMask; //The green component mask of the primitive

DWORD dwBBitMask; //The blue component mask of the primitive

The values of these masks can be found in xvga.h, specifically:

R_MASK_555 //Corresponds to the 555 format graphics

card G_MASK_555 //

B_MASK_555 //

R_MASK_655 //Corresponds to the 655 format graphics

card G_MASK_655 //

B_MASK_655 //

R_MASK_565 //Corresponds to the 565 format graphics

card G_MASK_565 //

B_MASK_565 //

R_MASK_556 //Corresponds to the 556 format graphics

card G_MASK_556 //

B_MASK_556 //

Return value: None

instruction:

When we load the image control module install(draw), the system will automatically call this function to set the image card's color mode and related functions.

However, we can also use this function directly. For example, when we are writing something that requires Windows GDI support, because Windows GDI uses the 555 format, we can set it like this:

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

Citation Tips

In RAYSSDK, the files related to the system module are xdraw.cpp and xdraw.h. When using the generated LIB, we only need to include the header file xdraw.h.

(seven) **Sound Control Module (DirectSound)**

Module Function

The sound control module uses DirectSound as the underlying layer to provide users with a convenient and applicable sound control interface.

The engine can control the simultaneous playback of up to 8 channels of sound. To facilitate the different processing of music and sound effects in the game, the engine classifies these sounds into two categories: one is sound effects, occupying channels 0-6, and the other is music, occupying channel 7.

Using the sound control module, we can mix, change the volume of a certain channel, balance, etc.

Data interface

extern SOUND_CFG sound_cfg;

Structure for recording engine sound configuration.

The structure is defined as follows:

typedef struct tagSOUND_CFG {

SLONG music_flag; //Flag of playing music, 1 = playing music, 0 = not playing music SLONG music_no; //Number of the currently playing music

SLONG music_total; //Total number of songs

SLONG music_volume; //Music volume

SLONG music_pan; //Music pan value

SLONG voice_flag; //Sound effect play flag, 1 = play, 0 = not play.

SLONG voice_volume; //Sound effect volume

SLONG voice_pan; //sound effect balance value} SOUND_CFG;

The range of volume and balance values is defined as follows:

MUSIC_VOLUME_MIN //-10000, minimum music volume

MUSIC_VOLUME_MAX //0, maximum music volume

MUSIC_PAN_LEFT //-10000, balanced sound on the left

MUSIC_PAN_CENTER //0, the sound effect is balanced and centered

MUSIC_PAN_RIGHT //10000, balanced sound on the right

VOICE_VOLUME_MIN //-10000, minimum sound effect volume

VOICE_VOLUME_MAX //0, maximum volume of sound effect

VOICE_PAN_LEFT //-10000, balanced sound on the left

VOICE_PAN_CENTER //0, the sound effect is balanced and centered
VOICE_PAN_RIGHT //10000, balanced sound on the right

Function interface

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

The above three functions are the module management interface functions of the sound control module.

EXPORT void FNBACK init_sound_cfg(void);

achievementEnable: Initialize the sound configuration structure sound_cfg.

ginseng Number: None

Return value: None

instruction:

EXPORT void FNBACK notify_changed_sound_cfg(SLONG changed_flags);

Function: When the application changes the content in sound_cfg, this function needs to be called to notify the engine in order for the engine to respond to the changes.

Parameters: SLONG changed_flags; // Changed data field flags This flag can be combined with the following values (|):

CHANGED_MUSIC_FLAG //Changed the music logo
CHANGED_MUSIC_TOTAL //Changed the number of music
CHANGED_MUSIC_VOLUME //Changed the volume of the music CHANGED_MUSIC_PAN //Changed the balance of the music
CHANGED_VOICE_FLAG //Changed the logo of the sound effect
CHANGED_VOICE_VOLUME //Changed the volume of the sound effect.
CHANGED_VOICE_PAN //Changed the balance of the sound effect.

Return value: None

instruction:

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

achievement Function: Play sound effects.

ginseng Number: SLONG channel; //Specify the sound channel for playing sound effects, 0~6.

SLONG loop; // Flag indicating whether to play in a loop, 1 = loop, 0 = play once

USTR *filename; // Sound effect file name (*.WAV)

Return value: None

instruction:

EXPORT void FNBACK stop_voice(SLONG channel);

achievement Function: Stop the sound effect playing of the specified channel.

ginseng Number: SLONG channel; //Specified sound channel

Return value: None

instruction:

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

achievement Function: Set the volume of the specified audio channel.

ginseng Number: SLONG channel; //Specify the sound channel

SLONG volume; //The volume to be set. Return

value: None

instruction:

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

achievement Function: Set the sound balance of the specified audio channel.

ginseng Number: SLONG channel; //Specify the sound channel

SLONG pan; //Set the balance value

Return value: None

instruction:

EXPORT SLONG FNBACK is_voice_playing(SLONG channel);

achievement Function: Check whether the sound effect of the specified channel is playing

ginseng Number: SLONG channel; //sound channel

Return value: If the video is playing, returns TRUE; otherwise, returns

FALSE.

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

achievement Function: Play music

ginseng Number: SLONG music_no; //Music number

SLONG loop; // Loop playback flag, 1 = loop playback, 0 = play once Return value: None

instruction:

EXPORT void FNBACK stop_music(void);

achievement Can: Stop playing music

ginseng Number: None

Return value: None

instruction:

EXPORT void FNBACK set_music_volume(SLONG volume);

achievement Function: Set the volume of music.

ginseng Number: SLONG volume; //Specify volume

Return value: None

instruction:

EXPORT void FNBACK set_music_pan(SLONG pan);

achievement Can: Set the balance of music.

ginseng Number: SLONG pan; //Specify the balance value

Return value: None

instruction:

EXPORT SLONG FNBACK is_music_playing(void);

achievement Yes: Check if music is playing.

ginseng Number: None

Return value: If the video is playing, returns TRUE; otherwise, returns FALSE.

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

achievement Can: Get information about a specified WAV file.

ginseng Number: USTR *filename; //WAV file name

ULONG *channels; //Store the number of sound channels of the WAV

ULONG *sample_rate; //Store the sampling rate of the WAV

ULONG *bps; //Store the bit rate of the WAV (BitsPerSample) //

ULONG *size; Store the data size of the WAV

ULONG *play_time; //How long can the WAV be played?

Return value: None

Note: If the acquisition is successful, TTN_OK is returned; otherwise, TTN_ERROR is returned.

Citation Tips

The files related to the sound control module are xsound.cpp and xsound.h. When the application references LIB, it directly includes the header file

xsound.h is enough.

(eight) **Input Control Module (DirectInput)**

Module Function

This module implements keyboard and mouse input control. It was originally planned to use DirectInput, but is currently implemented by intercepting Windows messages.

Data interface

No external data interface is provided.

Function interface

EXPORT int FNBACK init_input(void); EXPORT void FNBACK free_input(void); EXPORT void FNBACK active_input(int bActive); The above three functions are the module loading interface functions of the input control module.

EXPORT UCHR FNBACK read_system_key(void);

achievement Ability: Read system keys.

ginseng Number: None

Return value: If a key is pressed, returns the key value; otherwise, returns 0.

The key values are defined in rays.h and have the following values:

```
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,
PdD            = 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
};
```

Note: It is highly recommended to use the following read_data_key() function to read the keyboard, as it can read more key values. This function is retained only for compatibility with the original program.

EXPORT UCHR FNBACK read_data_key(void);

achievement: Can: Read data keys.

ginseng Number: None

Return value: If a key is pressed, return the data value; otherwise, return 0.

The data value of a key can be a printable character on the keyboard or some system keys. The data values of these system keys are defined in rays.h and are as follows:

```
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,
};
```

instruction:

EXPORT void FNBACK reset_key(void);

achievement: Yes: Reset all values in the keyboard buffer.

ginseng Number: None

Return value: None

Note: After we read the keystrokes, the keystroke value is retained in the key buffer. If we do not press any keys at this point and continue to call the keystroke read function, the keystroke value we obtain will remain the previous value. Therefore, generally speaking, after we finish reading the keys, we reset the keystroke buffer to facilitate the next keystroke read operation. Typical usage is shown in the following example:

Example:

```
UCHR ch; //Define a data to store the return value of the key reading
ch = read_system_key(); //Read the system key
if(ch) reset_key(); //If a key is pressed, reset the keyboard buffer and prepare for this key to be read
switch(ch) //Perform different operations based on the key read
{
    case Home:
        break;
    case End:
        break;
    default:
        break;
}
```

EXPORT void FNBACK reset_data_key(void);

achievement Function: Clear the value of the data key in the keyboard buffer.

ginseng Number: None

Return value: None

Note: Unlike reset_key(), this function simply clears the keyboard buffer for printable keys. When implementing text editing, we often need to implement functionality like this: for example, holding down the Shift key while typing a string of text, we want to obtain the corresponding shift symbol. In this implementation, the program uses this function to reset the keyboard after each keystroke. The following example illustrates this implementation.

Example:

```
UCHR ch;
USTR input_str[80];
SLONG input_finish;
memset(input_str, 0, 80);
input_finish = 0;
while( ! input_finish)
{
    ch = read_data_key();
//If we hold down Shift while typing and then press 12345, we will get !@#$%
```

//But if we use reset_key() instead of reset_data_key(), we will get !2345. //Because after we get !, reset_key() also clears the Shift key we hold down.

```
if(ch) reset_data_key();
switch(ch)
{
	case S_Del:
//TODO: Delete the character after the edit cursor
		break;
	case S_Backspace:
//TODO: Delete the character before the edit cursor
		break;
	case S_Enter: //End of input
		input_finish = 1;
		break;
	default:
		if(isprint(ch))
		{
//TODO: Add characters to input_str
		}
		break;
	}
}
```

EXPORT void FNBACK wait_tick(ULONG no);

achievement Yes: Wait for the specified time until the time is up.

ginseng Number: ULONG no; //The waiting time, in 1/100 seconds

Return value: None

instruction:

EXPORT void FNBACK wait_key(SLONG key);

achievement Can: Wait for the specified key until it is pressed.

ginseng Number: SLONG key; //key value

The key value has the same definition as the return value of read_data_key(). Please refer to the relevant introduction of read_data_key().

Return value: None

instruction:

EXPORT UCHR FNBACK wait_any_key(void);

achievement: Can: Wait for any key to be pressed.

ginseng Number: None

Return value: None

Note: The definition of the key to wait for is the same as read_data_key().

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

Function: Wait for a key to be pressed within a specified time. If the key is pressed, it will return immediately. Otherwise, it will also return if the time is up.

Parameter: SLONG key; //waiting key value

SLONG no; //waiting time, unit is 1/100 second. Return value: None

Note: The definition of the key value to be waited for is the same as read_data_key().

EXPORT void FNBACK clear_time_delay(void);

achievement: Function: Clear the exact time count.

ginseng Number: None

Return value: None

Note: When we need to use precise timing to control certain code, we can use the precise timing control functions provided by RAYSSDK. Specifically, they include three functions: clear_timer_delay(), get_time_delay(), and wait_time_delay().

EXPORT ULONG FNBACK get_time_delay(void);

achievement: Ability to obtain accurate time counting.

ginseng Number: None

Return value: accurate time value, in milliseconds (1/1000 seconds).

EXPORT void FNBACK wait_time_delay(ULONG count);

achievement: Yes: Wait for the exact time you specify.

ginseng Number: ULONG count; //waiting time, in milliseconds

Return value: None

instruction:

EXPORT void FNBACK fnKeyBoardInterrupt(UCHR keycode);

achievement Capability: Keyboard interrupt handling.

ginseng Number:

Return value:

Description: System-only function.

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

achievement Function: Mouse interrupt processing.

ginseng Number:

Return value:

Description: System-only function.

EXPORT void FNBACK show_mouse(SLONG flag);

achievement Yes: Display the mouse according to the display logo.

ginseng Number: SLONG flag; //Display the mouse flag

This flag can have the following values:

SHOW_WINDOW_CURSOR //Show the Windows mouse cursor

SHOW_IMAGE_CURSOR //Show the user's animated image cursor

Return value: None

instruction:

EXPORT void FNBACK set_mouse_cursor(SLONG type);

achievement Function: Set the mouse cursor type.

ginseng Number: SLONG type; // cursor type value

The value of this cursor can be defined by the user. However, the system supports up to 64 different cursors. Therefore, the value should be between 0 and 63.

The RAYSSDK provides load_mouse_cursor() and load_mouse_image_cursor() functions to load a Windows mouse cursor or a user-defined animated image cursor into a specified type. This function can then be used to conveniently select the loaded cursor.

Return value: None

instruction:

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

achievement Function: Set the position of the mouse in the workspace.

ginseng Number: SLONG xpos; //x coordinate of the mouse

SLONG ypos; //y coordinate of the mouse

Return value: None

instruction:

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

achievement Function: Get the position of the mouse in the workspace.

ginseng Number: SLONG *xpos; //Store the data pointer of the mouse x coordinate

SLONG *ypos; //Store the data pointer of the mouse y coordinate. Return

value: None

instruction:

EXPORT UCHR FNBACK get_mouse_key(void);

achievement Can: Read mouse buttons.

ginseng Number: None

Return value: If there is a key, return the corresponding mouse key value; otherwise, return 0. The

return value is defined in rays.h and is as follows:

```
enum     MOUSE_KEY_CODES
{     MS_Dummy      = 0x00, //No mouse button
      MS_Move         = 0xF1, //The mouse is moving
      MS_LDrag        = 0xF2, // Press and hold the left mouse button and drag
      MS_RDrag        = 0xF3, // right click and drag
      MS_LDn          = 0xF4, // Left mouse button pressed
      MS_LUp          = 0xF5, // Release the left mouse button
      MS_LDblClk      = 0xF6, // Double click the left mouse button
      MS_RDn          = 0xF7, // right mouse button pressed
      MS_RUp          = 0xF8, //Release the right mouse button
      MS_RDblClk      = 0xF9, // Double-click the right mouse button
      MS_MDn          = 0xFA, // Middle mouse click
      MS_MUp          = 0xFB, //Release the middle mouse button
      MS_MDblClk      = 0xFC, // Middle mouse double click
      MS_Forward      = 0xFD, //Mouse wheel scrolls forward
      MS_Backward     = 0xFE //Mouse wheel scrolls backward
};
```

say bright:

EXPORT UCHR FNBACK read_mouse_key(void);

achievement Function: Exactly the same as get_mouse_key(). Two different function names are provided for compatibility. Number:

ginseng None

Return value:

instruction:

EXPORT void FNBACK wait_mouse_any_key(void);

achievement Function: Wait for any mouse key (not MS_Dummy). Number:

ginseng None

Return value: None

instruction:

EXPORT void FNBACK wait_mouse_key(UCHR key);

achievement Yes: Wait for a specific mouse key.

ginseng Number: UCHR key; //The mouse key to wait for

Return value: None

instruction:

EXPORT SLONG FNBACK check_mouse_shift(void);

achievement Function: Check if the Shift key is held down when the mouse is

ginseng pressed.

Return value: If Shift is held down, returns TRUE; otherwise, returns FALSE.

EXPORT SLONG FNBACK check_mouse_control(void);

achievement Function: Check if the Ctrl key is held down when the mouse is

ginseng pressed.

Return value: If Ctrl is pressed, returns TRUE; otherwise, returns FALSE.

EXPORT void FNBACK reset_mouse(void);

achievement Function: Reset the mouse key buffer.

ginseng Number: None

Return value: None

instruction:

EXPORT void FNBACK reset_mouse_key(void);

achievement Function: This function is exactly the same as reset_mouse(). It is only provided as two different functions for compatibility.

Name.

ginseng Number:

Return value:

instruction:

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

achievement Can: Load the WINDOWS mouse cursor resource and assign the cursor to a specific mouse cursor

type.

ginseng Number: SLONG index; //Specified mouse cursor type

HCURSOR hCursor; //HANDLE of WINDOWS cursor resource Return

value: 0 if successful, otherwise other numbers.

instruction:

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

achievement Function: Set the hotspot of the user's animated image mouse.

ginseng Number: SLONG index; //The mouse cursor type that needs to be set

SLONG x; //x coordinate of hot spot

SLONG y; //y coordinate of hot spot

Return value: None

Note: When we set an animated image file as the mouse cursor, the system will set the rectangular frame of the image

The upper left corner is used as the hotspot for the mouse cursor. However, in practical applications, we often need to specify a specific point, such as the center of an image or any other point as the hotspot. This function can be used to achieve this. The hotspot coordinates are relative to the upper left corner.

EXPORT SLONG FNBACK init_mouse_image_cursor(void);

achievement Enable: Initialize the mouse image cursor storage area.

ginseng Number: None

Return value: Returns TTN_OK.

Description: System-only function.

EXPORT void FNBACK free_mouse_image_cursor(void);

achievement Function: Release the mouse image cursor storage area.

ginseng Number: None

Return value: None

Description: System-only function.

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

achievement Function: Load the animation image of the specified file as the mouse cursor, and assign the cursor to a specific slider.

rat type.

ginseng Number: SLONG index; //Specified mouse cursor type value

USTR filename; //Animation image file (*.CAK) Return value: Returns

TTN_OK if loaded successfully, otherwise returns TTN_NOT_OK. Description:

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

achievement Capable of: capturing animation clips to generate an animated mouse cursor image and assigning the cursor to a specific mouse

type.

ginseng Number: SLONG index; //Mouse type value

CAKE_FRAME_ANI *image_cfa; //Animation image pointer. It must be ensured that all nodes of this pointer

(including the head node) contain image data.

SLONG frames; // Number of mouse animation frames. Return

value: TTN_OK if successful, TTN_NOT_OK otherwise.

Description: This function will generate an animated image cursor based on the frames image after image_cfa.

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

achievement Function: Display the mouse animation image data to the specified storage area. In the core of the control module above,

Use this function.

ginseng Number:

Return value:

Description: System-only function.

EXPORT SHINT FNBACK fnCheckCtrlKey(void);

achievement Yes: Check if the Ctrl key is pressed.

ginseng Number: None

Return value: 1 if Ctrl is pressed, otherwise 0.

EXPORT SHINT FNBACK fnCheckLeftCtrlKey(void);

achievement Yes: Checks if the left Ctrl key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckRightCtrlKey(void);

achievement Yes: Check if the right Ctrl key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckAltKey(void);

achievement Yes: Checks whether the Alt key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckLeftAltKey(void);

achievement Yes: Checks if the left Alt key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckRightAltKey(void);

achievement Yes: Checks if the right Alt key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckShiftKey(void);

achievement Yes: Checks whether the Shift key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckLeftShiftKey(void);

achievement Yes: Checks if the left Shift key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

EXPORT SHINT FNBACK fnCheckRightShiftKey(void);

achievement Yes: Checks if the right Shift key is pressed.

ginseng Number: None

Return value: If pressed, returns 1, otherwise returns 0.

instruction:

Citation Tips

The files related to the input control module are xinput.cpp and xinput.h.

(Nine) **Network Control Module (DirectPlay)**

Module Function

This module uses DirectPlay as the underlying layer, providing applications with interfaces for selecting connection types, creating sessions, joining sessions, and receiving messages.

Data interface

Currently no external data interface is provided.

Function interface

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

The above three functions provide the module loading interface.

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

achievemeFunction: Set the function to process the message.

ginseng Number: FUNCDOMSG *do_sys; //Function for processing system messages

FUNCDOMSG *do_app; //The function that processes the

application FUNCDOMSG is defined as follows:

typedef void (FUNCDOMSG)(DPMSG_GENERIC* pMsg, DWORD dwMsgSize,

DPID idFrom, DPID idTo); Its parameters are a pointer to the message data, the message data size, the network ID of the player sending the message, and the network ID of the player receiving the message.

Return value: None

Note: After we have loaded the network control module, we need to set the function to process the message in order to

After receiving network messages, they process them.

In RAYSSDK, network messages are divided into two categories: one is DirectPlay system messages, and the other is messages defined by the application itself.

The network messages received are:

DPSYS_CHAT //Received a chat message

DPSYS_CREATEPLAYERORGROUP //Create a new player or group, when there is

When a new player joins the game, the online players will receive this message

DPSYS_DESTROYPLAYERORGROUP //Delete a player or group.

When you leave the game, other players will receive this message

DPSYS_HOST //This machine becomes the host

DPSYS_SESSIONLOST //Network session lost

DPSYS_ADDPLAYERTOGROUP //Add a new player to a group

DPSYS_DELETEPLAYERFROMGROUP //Delete a player from a group

DPSYS_SETPLAYERORGROUPDATA //Set player or group information

DPSYS_SETPLAYERORGROUPNAME //Set the player or group name

DPSYS_SETSESSIONDESC //Set session information

DPSYS_SENDCOMPLETE //Data transmission completed

etc.

Application messages are defined by users.

Example: Typical usage examples are shown below.

Suppose we have the following two functions that process network messages:

```
void do_system_message(DPMSG_GENERIC* pMsg, DWORD dwMsgSize, DPID
idFrom, DPID idTo )
{
switch(pMsg->dwType)
{
    case DPSYS_CREATEPLAYERORGROUP: //Handle the
creation of a new player or group
        break;
    case DPSYS_DESTROYPLAYERORGROUP: //
Processes deleting a player or group
        break;
//Other processing codes
//(omitted here)
    }
}

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

```
//Process user-defined network messages
        }
}
```

Then, after loading the network control module, we can use the following method to set the function pointer for processing messages.

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

OK, now we can focus on designing our message processing functions. Everything else? RAYSSDK has already taken care of it for us.

It should be noted that the network message we define ourselves needs to carry a dwType value as an identifier of the message type, and at the same time it can also achieve compatibility with the DirectPlay system.

For example, our message could be defined like this:

```
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);

achievement Yes: Get a list of connection types that the system can use.

ginseng Number: int ID; //Additional value of the network GUID, usually 0.

DPCNAME **lpname; //Store the acquired network connection type data. The

connection type data is defined as follows:

```
typedef struct DPCNAME_STRUCT DPCNAME,LPDPCNAME;
struct DPCNAME_STRUCT
{
    char    * name; //Connection type name
    int     type; //Type definition of connection type
};
```

The connection type is defined as follows:

NETCONNECTIPX //IPX connection

NETCONNECTTCPIP //TCPIP connection NETCONNECTMODEM //

Modem connection NETCONNECTSERIAL //Serial port connection

NETCONNECTOTHER //Other types of connections int *count; //

Stores the total number of connection types obtained

Return value: The list is as follows.

NET_DOK //Connection successful

NETERR_INIT //Network control not initialized

NETERR_DNODXX //Error creating DirectPlay or DirectPlayLobby object

NETERR_DENUMERR //Error enumerating connection types

say Note: After we get the connection type list, we can use NET_set_connect(num) to set

The method we want to use to establish the network connection. num is the index value of the list.

EXPORT int FNBACK NET_set_connect(int num);

achievement Can: Set the connection type

ginseng Number: int num; //Index of connection type data

Return value: If the setting is successful, it returns 0; otherwise, it returns

other numbers.

EXPORT int FNBACK NET_off_connect(void);

achievement Function: Free up network connection.

ginseng Number: None

Return value: 0 if successful, otherwise other numbers.

Description: Directly call DirectPlay's Release method to release the DirectPlay object.

EXPORT void FNBACK NET_set_phone(char *phone);

achievement Yes: Set the phone number when connecting via MODEM.

ginseng Number: char *phone; //Phone number string, such as "5186890"

Return value: None

instruction:

EXPORT void FNBACK NET_set_ip_address(char *address);

achievement Function: Set the IP address when using TCPIP connection. Parameters:

ginseng char *address; //IP address, such as "192.168.10.146"

Return value: None

instruction:

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

achievement Yes: Set the port number and serial transmission rate when using a serial connection.

ginseng Number: SLONG port; //Port number

int speed; //Serial transmission rate

Return value:

instruction:

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

achievement Yes: Get a list of currently connected sessions.

ginseng Number: char ***lpname; //Store the session name list

int *count; //Store the total number of sessions

Return value: Returns 0 if successful, otherwise other values.

instruction:

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

achievement: Can: Create a new session.

ginseng Number: char *sname; //Set the session name

int max_player; //Set the maximum number of players

allowed DPID *play_id; //Store the player's network ID

char *names; //Set the player's nickname (ShortName)

char *namel; //Set the player's name (LongName) Return

value: 0 if successful, otherwise another value.

Note: When we create a new session, the system will also create a new player.

The network identification ID of the player is stored in *play_id, and the player is added to the session as the host.

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

achievement: Yes: Join the specified session.

ginseng Number: int num; //session list index

DPID *play_id; //Store the player's network ID char

*names; //Set the player's nickname (ShortName) char

*namel; //Set the player's name (LongName) Return

value: 0 if successful, otherwise other values.

Note: When we join a session, similar to creating a session with NET_create_session(), the system will also create a new player and store the player's network identification ID in *play_id.

EXPORT int FNBACK NET_session_name(char *name);

achievement: Yes: Set the name of the session.

ginseng Number: char *name; //New session name

Return value: Returns 0 if successful, otherwise other values.

Note: Only the creator of a session can set the session name.

EXPORT int FNBACK NET_send(NETMSG *netmsg);

achievement: Can: Send online messages.

ginseng Number: NETMSG *netmsg; //Store the message to be sent

The definition of NETMSG is as follows:

```
struct NETMSG_STRUCT
{
    DWORD       dwType;         //Message type
    union
```

```
    {
      WORD     Flags;
      DWORD      size;                  //Message length
      };
PLAYERID       idFrom;                //The ID of the player who sent the message
LAYERID        idTo;                  //The ID of the player receiving the message
DWORD            dwCurrentPlayers;
union
      {
      LPVOID    data;                  //Message data address
      char *    message;               // A string ending with 0
      char *    ShortName;
      };
union
      {
      char *    LongName;
      LPDPSESSIONDESC2 desc;
      };
};
```

Return value: 0 if the message is sent successfully, otherwise other values.

Note: To maintain compatibility, this function is retained in RAYSSDK. It is recommended and strongly recommended to use

NET_send_data(LPVOID lpBuffer, DWORD dwSize, DPID idFrom, DPID idTo) function.

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

achievement Yes: Send network data messages.

ginseng Number: LPVOID lpBuffer; //Message data address

DWORD dwSize; //Message data length DPID idFrom; //Network

ID of the player sending the message DPID idTo; //Network ID

of the player receiving the message Return value: 0 if the

message is sent successfully, otherwise other values.

instruction:

EXPORT int FNBACK NET_send_chat(NETMSG *netmsg);

achievement Yes: Send chat messages.

ginseng Number: NETMSG *netmsg; //Message structure for storing chat messages.

Return value: Returns 0 if successful, otherwise other values.

Note: To maintain compatibility, this function is retained in the RAYSSDK. It is strongly recommended to use NET_send_chat(LPVOID lpBuffer, DWORD dwSize, DPID idFrom, DPID idTo) to send chat messages.

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

achievement Yes: Send chat messages.

ginseng Number: LPVOID lpBuffer; //Chat message storage address

DWORD dwSize; //Size of the chat message DPID idFrom; //

Network ID of the player sending the message DPID idTo; //

Network ID of the player receiving the message Return

value: 0 if successful, otherwise another value.

instruction:

EXPORT int FNBACK NET_close(void);

achievement Yes: Turn off the network connection.

ginseng Number: None

Return value: Returns 0 if successful, otherwise other values.

Note: When you close the network connection, the system will delete the players (created when you created or joined the connection) and

And call the Close interface of DirectPlay to close DirectPlay.

EXPORT int FNBACK NET_end_join(void);

achievement Can: Terminate new players from joining the session.

ginseng Number: None

Return value: Returns 0 if successful, otherwise other values.

Description: By changing the relevant parameters of the session, terminate other new players from joining the session.

The client that creates a session can call this function.

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

achievement Can: Enumerate all players in a given session.

ginseng Number: int num; //The index value of the session list

ENUMPLAYER *p; // Enumerate callback function pointers. The

definition of the callback function is

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

The corresponding parameters represent the player's network ID, nickname (ShortName), and long name (LongName). Return value: If the enumeration is successful, it returns 0, otherwise it returns another value.

Note: Users can set their own callback functions to facilitate statistics, etc. For detailed principles of callback functions, please refer to the DirectX development manual.

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

achievement Can: Set the player's name.

ginseng Number: char *names; //Nickname (ShortName)

char *namel; //Long name (LongName) Return

value: 0 if successful, otherwise other values.

instruction:

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

achievement Can: Get the specified player's name.

ginseng Number: DPID player_id; //Player's network ID

char **friendlyname; //Get the player's nickname char

**formalname; //Get the player's formal name Return

value: None

instruction:

Citation Tips

The relevant files are xplay.cpp and xplay.h.

(ten) **CDROM Audio Track Control Module (CD Music)**

Module Function

The CDROM audio track playback module uses WINDOWS's MCI function as the underlying layer to provide a convenient application interface to application users.

By using this module, users can easily realize functions such as playing CD music, stopping playing CD music, pausing, and continuing playing.

Data interface

There is currently no external data interface.

Function interface

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

The above three functions are the module loading functions of the CDROM audio track control module.

EXPORT void FNBACK pause_cdrom_music(void);

achievement Function: Pause the playing of CDROM music.

ginseng Number: None

Return value: None

Note: By calling resume_cdrom_music(), you can resume playing CDROM music from the paused position.

EXPORT void FNBACK resume_cdrom_music(void);

achievement Yes: Continue playing CDROM music.

ginseng Number: None

Return value: None

Description: Used in conjunction with pause_cdrom_music().

EXPORT void FNBACK stop_cdrom_music(void);

achievement Function: Stop playing CDROM music.

ginseng Number: None

Return value: None

instruction:

EXPORT void FNBACK play_cdrom_music(SLONG track);

achievement Function: Play the music of the specified track.

ginseng Number: SLONG track; //Track number, starting from 0.

Return value: None

Note: Note that this function will play the audio track from the beginning.

EXPORT SLONG FNBACK status_cdrom_music(void);

achievement Function: Check the playing status of CDROM music.

ginseng Number: None

Return value: If the CDROM music has stopped, it returns 0; otherwise, it returns other values.

EXPORT void FNBACK loop_cdrom_music(void);

achievement Function: Play CDROM music in a loop.

ginseng Number: None

Return value: None

instruction:

If we need to loop CDROM music, we simply need to call this function repeatedly and periodically in our program. This will allow us to loop the current music. A typical approach is to check this in a function like idle_loop(). However, for flexibility, the idle_loop() function provided in the RAYSSDK currently does not call this function.

In fact, it is recommended to use MP3 or WAV as the music for our game. The corresponding function of using WAV is implemented in the sound control module. For the method of using MP3, please refer to the MP3 control module.

EXPORT SLONG FNBACK total_cdrom_music(void);

achievement Function: Get the number of tracks on the current CDROM.

ginseng Number: None

Return value: Returns the number of tracks obtained.

instruction:

Citation Tips

The relevant files are xcdrom.cpp and xcdrom.h.

(eleven) **Chinese character display module (Font)**

Module Function

For displaying Chinese characters, the previous method used was to directly use a bitmap font file. The advantage of this method is that regardless of whether the operating system version of the application is Simplified Chinese, Traditional Chinese, or English, Chinese characters can be displayed normally and conveniently (the module name is font).

However, for maximum compatibility and scalability with the WINDOWS system, the current font used is the operating system-based TrueTypeFont (TTF) text display (module name is winfont).

The usage of the two modules is generally the same. Here, we will use the winfont module for illustration. It is recommended that you use the winfont module when developing applications to display Chinese characters.

Data interface

This module currently does not provide any external data interface.

Function interface

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

The above three functions are the module loading interface functions of the Chinese character display module.

```
void print_640x480x16M(SLONG x,SLONG y,USTR *data,SLONG
        put_type,BMP *bit_map); print_range_640x480x16M(SLONG
void 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); The above function is an application function used when displaying Chinese characters.

However, for compatibility with the xfont module and to facilitate future implementations of the application on other platforms, it is strongly recommended that you use the macros provided by this module to display Chinese characters. The following will focus on how to use these macros.

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)

They realize Chinese character display with sizes of 16X16, 20X20, 24X24, 28X28, and 32X32 points respectively.

Parameter Description:

(SLONG) x is the X coordinate of the starting point of the string display.

(SLONG) y is the Y coordinate of the starting point of the string display.

(USTR*) str is the Chinese character string to be displayed.

(SLONG) type is the way to display Chinese characters.

(BMP*) bitmap is the destination bitmap.

There are a few important points to note here:

(1) The alignment of Chinese characters is top-left alignment. For example, if we want to display the Chinese character "希望" (hope) with a size of 16 at the top-left corner (0,0) of the bitmap and keep the top-left corner of the displayed Chinese character at that point, we can do this as follows:

print16(0,0,"Hope",PEST_PUT,bitmap);

(2) The displayed Chinese character string can contain some control words. These control words can achieve functions such as changing the foreground color of the Chinese character display, changing the effect of the Chinese character display, etc. The control words that can be used are:

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

This will change the foreground color of Chinese characters. C stands for Color. The following 0, 1, .. 9 represent different colors, as shown below:

0 SYSTEM_WHITE White
1 SYSTEM_RED Red
2 SYSTEM_GREEN Green
3 SYSTEM_BLUE Blue
4 SYSTEM_YELLOW Yellow
5 SYSTEM_CYAN cyan
6 SYSTEM_PINK Pink

7 SYSTEM_WHITE White, same as 0
8 SYSTEM_BLACK Black
9 SYSTEM_DARK0 Gray

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

This will change the stroke effect of the Chinese characters. 0 disables the stroke; the other 1-9 values represent different stroke colors, with the same color definitions as above.

(3) (SLONG)type specifies the way to display Chinese characters. Its values are defined as follows:

PEST_PUT Display Chinese characters by filtering out the transparent color

(PEST) COPY_PUT Display Chinese characters by directly copying them

Or the following macro

COPY_PUT_COLOR(a)

This macro specifies the background color for Chinese characters. The macro parameter a is a color value. To ensure compatibility with different graphics cards, it is strongly recommended that this parameter be a system color, such as SYSTEM_RED or SYSTEM_BLUE. For the definition of system colors, please refer to the engine's drawing function topic.

For example, if we want to display the Chinese character "hope" in white with a red border and a blue background, we can use it like this:

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

If we don't specify a background color, we can do this:

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

Citation Tips

The files related to Chinese character display are xfont.cpp, xfont.h and winfont.cpp, winfont.h.

(twelve) **MP3 player module (MP3)**

Module Function

This module dynamically decompresses and plays MP3 music files. It should be noted that this module currently only works under Windows 2000. It will be updated to support Windows 98, Windows 95, and other platforms in the next release.

Data interface

There is currently no external data interface.

Function interface

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

The above three functions provide the loading interface of the MP3 module.

EXPORT void FNBACK play_mp3(SLONG loop, USTR *filename);

achievement Can: Play MP3 music.

ginseng Number: SLONG loop; // Loop play flag, 1 = loop play, 0 = play once
USTR *filename; //MP3 file name Return
value: None

instruction:

EXPORT void FNBACK stop_mp3(void);

achievement Function: Stop MP3 music playing.

ginseng Number: None

Return value: None

instruction:

EXPORT void FNBACK set_mp3_volume(SLONG volume);

achievement Function: Set the volume for playing MP3.

ginseng Number: SLONG volume; //Specified volume
Please refer to the definition of volume in the music control module (xsound.h). Return
value: None

instruction:

EXPORT void FNBACK set_mp3_pan(SLONG pan);

achievement Function: Set the equalizer value for playing MP3.

ginseng Number: SLONG pan; //Specified balance value

Please refer to the definition of equalizer value in the music control module (xsound.h).

Return value: None

instruction:

EXPORT SLONG FNBACK is_mp3_playing(void);

achievement Can: Check if MP3 music is playing.

ginseng Number: None

Return value: If the video is playing, returns TRUE; otherwise, returns FALSE.

Citation Tips

The files related to this module are mp3.h and a series of other files, which are not listed here one by one.

(Thirteen) **Windows BMP image format support**

File format introduction

BMP is the bitmap format of Windows. The file

header format is as follows:

```
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;
```

Data interface

No external data interface is currently provided.

Function interface

EXPORT BMP* FNBACK BMP_load_file(char *filename);

achievement Function: Load a Windows BMP bitmap file into the engine's bitmap. Parameters:

ginseng char *filename; //Windows BMP bitmap file name

Return value: Returns a pointer to the engine bitmap data.

For information about the engine bitmap data structure (BMP), please refer to the Bitmap Operation section of Engine Image Operation.

instruction:

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

achievement Function: Store the engine bitmap data as a WINDOWS BMP format file.

ginseng Parameters: BMP *bmp; //Pointer to the engine bitmap data

char *filename; //Target file name Return

value: 0 if successful, otherwise other values.

instruction:

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

achievement Function: Store the data in the specified memory area as a Windows BMP format file.

ginseng Parameters: UHINT *buffer; //Pointer to the image data buffer

SLONG xl; //bitmap width

SLONG yl; //bitmap height

char *filename; //Target file name Return

value: 0 if successful, otherwise other values.

instruction:

Citation

The relevant files are vbmp.cpp and vbmp.h.

(fourteen) **PCX image format support**

File format introduction

The file header structure is as follows:

```
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;
```

Data interface

Currently no external data interface is provided.

Function interface

EXPORT BMP* FNBACK PCX_load_file(char *filename);

achievement Function: Load a PCX image file into the engine bitmap structure.

ginseng Parameters: char *filename; //PCX image file name

Return value: Returns a pointer to the engine bitmap structure data.

For the definition of the engine bitmap data structure, please refer to the bitmap operation section of the engine image operation.

instruction:

Citation

The corresponding files are vpcx.cpp and vpcx.h.

(fifteen) **TGA image format support**

File format introduction

The file header structure is as follows:

```
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;
```

Data interface

Currently no external data interface is provided.

Function interface

EXPORT BMP* FNBACK TGA_load_file(char *filename);

achievement Can: Load TGA images into the engine bitmap.

ginseng Number: char *filename; //PCX file name

Return value: Returns a pointer to the engine bitmap data structure.

Description: When we load a TGA image, this function will automatically filter the Alpha channel of the TGA and

It turns into the system transparent color (pure black).

The filter threshold can be set via TGA_set_transparency_level(). Alpha values above this threshold will retain their original color, while values below this threshold will be replaced with the system transparency color.

However, when we need to load a TGA image with an alpha channel, in order to keep the original alpha information, it is recommended to use the TGA_load_file_with_alpha() function.

EXPORT ABMP* FNBACK TGA_load_file_with_alpha(char *filename);

achievement Function: Load TGA image into the engine's alpha bitmap.

ginseng Parameters: char *filename; //TGA file name

Return value: Returns a pointer to the engine's alpha bitmap data.

For the definition of the engine alpha bitmap data structure, please refer to the bitmap operation section of the engine image operation.

instruction:

EXPORT void FNBACK TGA_set_transparency_level(SLONG level);

achievement Function: Set the transparency threshold (Alpha threshold) when loading

ginseng TGA. Parameter: SLONG level; //Alpha threshold, 0~255

Return value: None

Description: Used with TGA_load_file().

Citation

(sixteen) **JPG image format support**

File format introduction

The JPG format is relatively complex, please refer to relevant books.

Data interface

Function interface

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

achievement Yes: Load JPG images into the memory buffer.

ginseng Number: const char *filename; //JPG file name

unsigned char **pic; //Pointer to the memory buffer int

*width; //Store image width

int *height; //Store image height

Return value: Returns 0 if successful, otherwise other values.

say bright:

After loading is complete, the data in the corresponding memory buffer is the restored JPG image information. Each image element corresponds to 4 consecutive bytes in the memory buffer, which are the r, g, b, and a values of the element, respectively.

We can combine these values through operations and then obtain other engine bitmap data suitable for our needs.

Fan Example: Here is an example to illustrate:

```
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;

//Load JPG image file

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

    fore_bitmap = create_bitmap(jpg_width, jpg_height);
```

```
        if(fore_bitmap)
                {
//Convert the contents of jpg_pic into
fore_bitmap //and convert it to high 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); }

//Release the memory occupied by jpg_pic that is no longer needed
                if(*jpg_pic)
                        {
                        free(*jpg_pic);
                        *jpg_pic =
                        NULL; }
//Subsequent operations on fore_bitmap
//? (omitted here)
                }
        }
```

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

This function is currently not recommended.

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

This function is currently not recommended.

Citation

The relevant file is jpeg.h.

(Seventeen) **PhotoShop PSD format support**

File format introduction

For more information about PSD file formats, please refer to relevant books.

Data interface

Function interface

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

achievement Can: Open a PSD file.

ginseng Number: USTR *filename; //PSD file name

SLONG *layers; //Number of layers for storing PSD

files SLONG *xl; //Width of PSD image

SLONG *yl; //Height of PSD image Return value: If the file is opened successfully, a pointer to the structure type PSDFILE is returned. Otherwise,

Returns NULL.

instruction:

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

achievement Function: Read the specified layer information from the opened PSD

ginseng file. Parameter: SLONG layer; //Specify the PSD layer to be read

SLONG *sx; //The starting x coordinate of the layer image

SLONG *sy; //The starting y coordinate of the layer image

SLONG *xl; //The width of the layer image

SLONG *yl; //The height of the layer image

PSDFILE *f; //PSDFILE file pointer Return value: If the read is successful, it returns 0; otherwise, it returns other values.

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

achievement Function: Read the image of a specified layer from the opened PSD file.

ginseng Parameter: SLONG layer; //specify the layer

UHINT *buffer; //Buffer for storing images USTR

*alpha_buffer; //Buffer for storing image alpha PSDFILE

*f; //PSDFILE file pointer

Return value: Returns 0 if reading is successful, otherwise returns other values.

say Note: When reading an image of a layer in a PSD file, we must first call read_psd_layer_info()

To obtain the information of the layer, allocate a buffer to store the image and a buffer to store the image Alpha, and then call this function

Read_psd_layer_image(). Example: Display

Fan each layer of a PSD file in sequence

```
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);

achievement Function: Read the display GRAPH image of the PSD file.

ginseng Parameters: UHINT *buffer; //Store the buffer for display image

PSDFILE *f; //PSDFILE file pointer return value: returns 0 if reading is successful, otherwise returns other values.

instruction:

EXPORT void FNBACK close_psd_file(PSDFILE *f);

achievement Function: Close the opened PSD file.

ginseng Number: PSDFILE *f; //PSD file pointer

Return value: None

instruction:

EXPORT void FNBACK set_psd_transparency_level(SLONG level);

achievement Function: Set the penetration threshold when reading PSD files.

ginseng Number: SLONG level; //threshold value, 0~255

Return value: None

instruction:

Citation

The relevant files are vpsd.cpp and vpsd.h.

(eighteen) **FLC animation format support**

File format introduction

FLC is an early 256-color animation format and is widely used in AnimatorPro. Please refer to relevant books for its specific file format.

Data interface

There is currently no external data interface.

Function interface

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

achievement: Can: Open an FLC animation file.

ginseng Number: USTR *name; //FLC file name

SLONG mode_flag; //Turn on FLC mode

OPEN_FLIC_MODE_640X480: 640X480 mode

OPEN_FLIC_MODE_320X200: 320X200 mode USTR *memory_plan; //memory buffer

Return value: If the opening is successful, the number of frames of the FLC animation is returned; otherwise, TTN_ERROR is returned.

EXPORT void FNBACK FLIC_close_flic_file(void);

achievement: Yes: Close the FLC file.

ginseng Number: None

Return value: None

instruction:

EXPORT SHINT FNBACK FLIC_read_flic_data(void);

achievement: Yes: Read FLC data.

ginseng Number: None

Return value: If the reading is successful, it returns 0; otherwise, it returns -1 (for example, when reading the last frame of FLC animation back).

instruction:

EXPORT SHINT FNBACK FLIC_play_flic_frame(void);

achievement Yes: Decompress the read data into the specified memory buffer.

ginseng Number: None

Return value: Returns 0.

say bright:

Fan Example: Read and display each frame of FLC.

```
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);

achievement Yes: Export the color palette of FLC files.

ginseng Number: USTR *palette; //Pointer to the palette buffer

Return value: None

instruction:

Citation

The relevant files are vflic.cpp and vflic.h.

(nineteen) **AVI animation format support**

File format introduction

AVI is the animation format for Windows. In RAYSSDK, DirectMedia is used to play AVI files.

Data interface

extern RECT rcXMedia;

The size of the animated image.

extern STREAM_TIME stStartTime;

The animation start time.

extern STREAM_TIME stEndTime;

Animation end time.

extern STREAM_TIME stCurrentTime;

The current animation playback time.

extern STREAM_TIME stDuration;

The total running time of the animation.

Function interface

void set_xmedia_over_draw(PFNREDRAW my_redraw);

achievement: Function: Set the animation playback

ginseng Number:

Return value:

instruction:

void play_xmedia_movie(SLONG sx,SLONG sy,SLONG ex,SLONG ey,USTR * filename,SLONG wait_flag);

achievement: able:

ginseng Number:

Return value:

instruction:

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

achievement able:

ginseng Number:

Return value:

instruction:

SLONG play_xmedia_frame(BMP *bitmap,SLONG left_top_flag);

achievement able:

ginseng Number:

Return value:

instruction:

SLONG close_xmedia_file(void);

achievement able:

ginseng Number:

Return value:

instruction:

void seek_xmedia_file(STREAM_TIME st);

achievement able:

ginseng Number:

Return value:

instruction:

Citation

The relevant files are xmedia.cpp and xmedia.h.