

原 FFMpeg源代码简单分析：libavdevice的gdigrab

2015年03月25日 12:33:42 阅读数：9995

=====

FFmpeg的库函数源代码分析文章列表：

【架构图】

[FFmpeg源代码结构图 - 解码](#)

[FFmpeg源代码结构图 - 编码](#)

【通用】

[FFmpeg 源代码简单分析：av_register_all\(\)](#)

[FFmpeg 源代码简单分析：avcodec_register_all\(\)](#)

[FFmpeg 源代码简单分析：内存的分配和释放（av_malloc\(\)、av_free\(\)等）](#)

[FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）](#)

[FFmpeg 源代码简单分析：avio_open2\(\)](#)

[FFmpeg 源代码简单分析：av_find_decoder\(\)和av_find_encoder\(\)](#)

[FFmpeg 源代码简单分析：avcodec_open2\(\)](#)

[FFmpeg 源代码简单分析：avcodec_close\(\)](#)

【解码】

[图解FFMPEG打开媒体的函数avformat_open_input](#)

[FFmpeg 源代码简单分析：avformat_open_input\(\)](#)

[FFmpeg 源代码简单分析：avformat_find_stream_info\(\)](#)

[FFmpeg 源代码简单分析：av_read_frame\(\)](#)

[FFmpeg 源代码简单分析：avcodec_decode_video2\(\)](#)

[FFmpeg 源代码简单分析：avformat_close_input\(\)](#)

【编码】

[FFmpeg 源代码简单分析：avformat_alloc_output_context2\(\)](#)

[FFmpeg 源代码简单分析：avformat_write_header\(\)](#)

[FFmpeg 源代码简单分析：avcodec_encode_video\(\)](#)

[FFmpeg 源代码简单分析：av_write_frame\(\)](#)

[FFmpeg 源代码简单分析：av_write_trailer\(\)](#)

【其它】

[FFmpeg源代码简单分析：日志输出系统（av_log\(\)等）](#)

[FFmpeg源代码简单分析：结构体成员管理系统-AVClass](#)

[FFmpeg源代码简单分析：结构体成员管理系统-AVOption](#)

[FFmpeg源代码简单分析：libswscale的sws_getContext\(\)](#)

[FFmpeg源代码简单分析：libswscale的sws_scale\(\)](#)

FFmpeg源代码简单分析：libavdevice的avdevice_register_all()

FFmpeg源代码简单分析：libavdevice的gdigrab

【脚本】

FFmpeg源代码简单分析：makefile

FFmpeg源代码简单分析：configure

【H.264】

FFmpeg的H.264解码器源代码简单分析：概述

=====

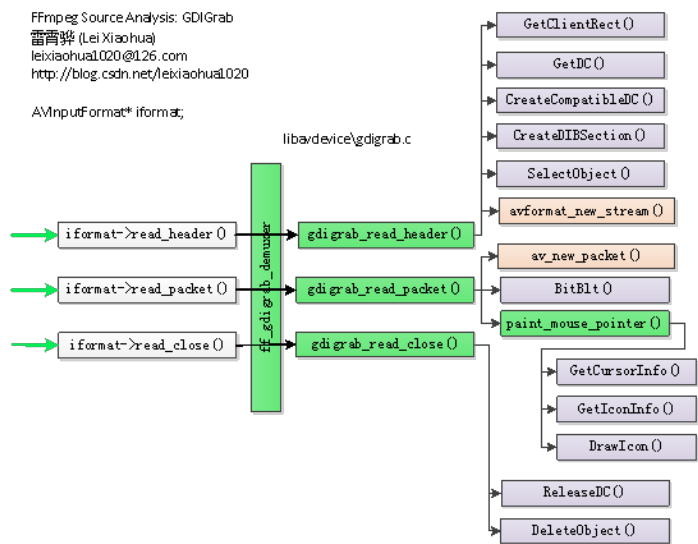
本文记录FFmpeg的libavdevice中GDIGrab组件的源代码。GDIGrab用于在Windows下屏幕录像（抓屏）。在ffmpeg.exe中使用可以参考文章：

FFmpeg获取DirectShow设备数据（摄像头，录屏）

编程使用可以参考文章：

最简单的基于FFmpeg的AVDevice例子（屏幕录制）

gdigrab的源代码位于libavdevice\gdigrab.c。关键函数的调用关系图如下图所示。图中绿色背景的函数代表源代码中自己声明的函数，紫色背景的函数代表Win32的API函数。



ff_gdigrab_demuxer

在FFmpeg中Device也被当做是一种Format，因为GDIGrab是一个输入设备，因此被当作一个AVInputFormat。GDIGrab对应的AVInputFormat结构体如下所示。

```
[cpp]
1. AVInputFormat ff_gdigrab_demuxer = {
2.     .name = "gdigrab",
3.     .long_name = NULL_IF_CONFIG_SMALL("GDI API Windows frame grabber"),
4.     .priv_data_size = sizeof(struct gdigrab),
5.     .read_header = gdigrab_read_header,
6.     .read_packet = gdigrab_read_packet,
7.     .read_close = gdigrab_read_close,
8.     .flags = AVFMT_NOFILE,
9.     .priv_class = &gdigrab_class,
10. };
```

从该结构体可以看出：

- 设备名称是"gdigrab"；
- 设备完整名称是"GDI API Windows frame grabber"；
- 初始化函数指针read_header()指向gdigrab_read_header()；
- 读取数据函数指针read_packet()指向gdigrab_read_packet()；
- 关闭函数指针read_close()指向gdigrab_read_close()；

Flags设置为AVFMT_NOFILE；

AVClass指定为gdigrab_class。

下面分析一下这些数据。

gdigrab_class

ff_gdigrab_demuxer指定它的AVClass为一个名称为"gdigrab_class"的静态变量。有关AVClass的概念之前已经记录过，在这里不再重复。gdigrab_class的定义如下。

```
[cpp]
1. static const AVClass gdigrab_class = {
2.     .class_name = "GDIgrab indev",
3.     .item_name  = av_default_item_name,
4.     .option     = options,
5.     .version    = LIBAVUTIL_VERSION_INT,
6. };
```

从gdigrab_class的定义可以看出，它指定了一个名称为"options"的数组作为它的选项数组（赋值给AVClass的option变量）。

options

下面看一下这个options数组的定义，如下所示。

```
[cpp]
1. #define OFFSET(x) offsetof(struct gdigrab, x)
2. #define DEC AV_OPT_FLAG_DECODING_PARAM
3. static const AVOption options[] = {
4.     { "draw_mouse", "draw the mouse pointer", OFFSET(draw_mouse), AV_OPT_TYPE_INT, {.i64 = 1}, 0, 1, DEC },
5.     { "show_region", "draw border around capture area", OFFSET(show_region), AV_OPT_TYPE_INT, {.i64 = 0}, 0, 1, DEC },
6.     { "framerate", "set video frame rate", OFFSET(framerate), AV_OPT_TYPE_VIDEO_RATE, {.str = "ntsc"}, 0, 0, DEC },
7.     { "video_size", "set video frame size", OFFSET(width), AV_OPT_TYPE_IMAGE_SIZE, {.str = NULL}, 0, 0, DEC },
8.     { "offset_x", "capture area x offset", OFFSET(offset_x), AV_OPT_TYPE_INT, {.i64 = 0}, INT_MIN, INT_MAX, DEC },
9.     { "offset_y", "capture area y offset", OFFSET(offset_y), AV_OPT_TYPE_INT, {.i64 = 0}, INT_MIN, INT_MAX, DEC },
10.    { NULL },
11. };
```

options数组中包含了该Device支持的选项。可以看出GDIGrab支持如下选项：

draw_mouse：画出鼠标指针。

show_region：划出抓屏区域的边界。

framerate：抓屏帧率。

video_size：抓屏的大小。

offset_x：抓屏起始点x轴坐标。

offset_y：抓屏起始点y轴坐标。

从宏定义"#define OFFSET(x) offsetof(struct gdigrab, x)"中可以看出，这些选项都存储在一个名称为"gdigrab"的结构体中。

Gdigrab 上下文结构体

Gdigrab上下文结构体中存储了GDIGrab设备用到的各种变量，定义如下所示。

```

1.  /**
2.   * GDI Device Demuxer context
3.   */
4.  struct gdigrab {
5.      const AVClass *class;  /**< Class for private options */
6.
7.      int      frame_size;  /**< Size in bytes of the frame pixel data */
8.      int      header_size; /**< Size in bytes of the DIB header */
9.      AVRational time_base; /**< Time base */
10.     int64_t    time_frame; /**< Current time */
11.
12.     int      draw_mouse;  /**< Draw mouse cursor (private option) */
13.     int      show_region; /**< Draw border (private option) */
14.     AVRational framerate; /**< Capture framerate (private option) */
15.     int      width;       /**< Width of the grab frame (private option) */
16.     int      height;      /**< Height of the grab frame (private option) */
17.     int      offset_x;    /**< Capture x offset (private option) */
18.     int      offset_y;    /**< Capture y offset (private option) */
19.
20.     HWND     hwnd;        /**< Handle of the window for the grab */
21.     HDC       source_hdc;  /**< Source device context */
22.     HDC       dest_hdc;    /**< Destination, source-compatible DC */
23.     BITMAPINFO bmi;       /**< Information describing DIB format */
24.     HBITMAP   hbmp;       /**< Information on the bitmap captured */
25.     void      *buffer;     /**< The buffer containing the bitmap image data */
26.     RECT      clip_rect;  /**< The subarea of the screen or window to clip */
27.
28.     HWND     region_hwnd; /**< Handle of the region border window */
29.
30.     int cursor_error_printed;
31. };

```

gdigrab_read_header()

gdigrab_read_header()用于初始化gdigrab。函数的定义如下所示。

```

1.  /**
2.   * Initializes the gdi grab device demuxer (public device demuxer API).
3.   *
4.   * @param s1 Context from avformat core
5.   * @return AVERROR_IO error, 0 success
6.   */
7.  static int
8.  gdigrab_read_header(AVFormatContext *s1)
9.  {
10.     struct gdigrab *gdigrab = s1->priv_data;
11.     //窗口句柄
12.     HWND hwnd;
13.     HDC source_hdc = NULL;
14.     HDC dest_hdc = NULL;
15.     BITMAPINFO bmi;
16.     HBITMAP hbmp = NULL;
17.     void *buffer = NULL;
18.
19.     const char *filename = s1->filename;
20.     const char *name = NULL;
21.     AVStream *st = NULL;
22.
23.     int bpp;
24.     RECT virtual_rect;
25.     //窗口的位置和大小
26.     RECT clip_rect;
27.     BITMAP bmp;
28.     int ret;
29.     //filename为窗口名称
30.     if (!strcmp(filename, "title=", 6)) {
31.         name = filename + 6;
32.         //查找窗口的句柄
33.         hwnd = FindWindow(NULL, name);
34.         if (!hwnd) {
35.             av_log(s1, AV_LOG_ERROR,
36.                 "Can't find window '%s', aborting.\n", name);
37.             ret = AVERROR(EIO);
38.             goto error;
39.         }
40.         if (gdigrab->show_region) {
41.             av_log(s1, AV_LOG_WARNING,
42.                 "Can't show region when grabbing a window.\n");
43.             gdigrab->show_region = 0;
44.         }
45.         //filename为desktop
46.     } else if (!strcmp(filename, "desktop")) {
47.         //窗口句柄为NULL
48.         hwnd = NULL;
49.     } else {

```

```

50.     av_log(s1, AV_LOG_ERROR,
51.           "Please use \"desktop\" or \"title=<windowname>\" to specify your target.\n");
52.     ret = AERROR(EIO);
53.     goto error;
54. }
55.
56. if (hwnd) {
57.     GetClientRect(hwnd, &virtual_rect);
58. } else {
59.     //窗口句柄为NULL, 代表是全屏
60.     virtual_rect.left = GetSystemMetrics(SM_XVIRTUALSCREEN);
61.     virtual_rect.top = GetSystemMetrics(SM_YVIRTUALSCREEN);
62.     virtual_rect.right = virtual_rect.left + GetSystemMetrics(SM_CXVIRTUALSCREEN);
63.     virtual_rect.bottom = virtual_rect.top + GetSystemMetrics(SM_CYVIRTUALSCREEN);
64. }
65.
66. /* If no width or height set, use full screen/window area */
67. if (!gdigrab->width || !gdigrab->height) {
68.     clip_rect.left = virtual_rect.left;
69.     clip_rect.top = virtual_rect.top;
70.     clip_rect.right = virtual_rect.right;
71.     clip_rect.bottom = virtual_rect.bottom;
72. } else {
73.     clip_rect.left = gdigrab->offset_x;
74.     clip_rect.top = gdigrab->offset_y;
75.     clip_rect.right = gdigrab->width + gdigrab->offset_x;
76.     clip_rect.bottom = gdigrab->height + gdigrab->offset_y;
77. }
78.
79. if (clip_rect.left < virtual_rect.left ||
80.     clip_rect.top < virtual_rect.top ||
81.     clip_rect.right > virtual_rect.right ||
82.     clip_rect.bottom > virtual_rect.bottom) {
83.     av_log(s1, AV_LOG_ERROR,
84.           "Capture area (%li,%li),(%li,%li) extends outside window area (%li,%li),(%li,%li)",
85.           clip_rect.left, clip_rect.top,
86.           clip_rect.right, clip_rect.bottom,
87.           virtual_rect.left, virtual_rect.top,
88.           virtual_rect.right, virtual_rect.bottom);
89.     ret = AERROR(EIO);
90.     goto error;
91. }
92.
93. /* This will get the device context for the selected window, or if
94.  * none, the primary screen */
95. //得到某个窗口句柄的DC
96. source_hdc = GetDC(hwnd);
97. if (!source_hdc) {
98.     WIN32_API_ERROR("Couldn't get window device context");
99.     ret = AERROR(EIO);
100.    goto error;
101. }
102. bpp = GetDeviceCaps(source_hdc, BITSPIXEL);
103.
104. if (name) {
105.     av_log(s1, AV_LOG_INFO,
106.           "Found window %s, capturing %lix%lix%i at (%li,%li)\n",
107.           name,
108.           clip_rect.right - clip_rect.left,
109.           clip_rect.bottom - clip_rect.top,
110.           bpp, clip_rect.left, clip_rect.top);
111. } else {
112.     av_log(s1, AV_LOG_INFO,
113.           "Capturing whole desktop as %lix%lix%i at (%li,%li)\n",
114.           clip_rect.right - clip_rect.left,
115.           clip_rect.bottom - clip_rect.top,
116.           bpp, clip_rect.left, clip_rect.top);
117. }
118.
119. if (clip_rect.right - clip_rect.left <= 0 ||
120.     clip_rect.bottom - clip_rect.top <= 0 || bpp%8) {
121.     av_log(s1, AV_LOG_ERROR, "Invalid properties, aborting\n");
122.     ret = AERROR(EIO);
123.     goto error;
124. }
125. //创建一个与指定设备兼容的HDC
126. dest_hdc = CreateCompatibleDC(source_hdc);
127. if (!dest_hdc) {
128.     WIN32_API_ERROR("Screen DC CreateCompatibleDC");
129.     ret = AERROR(EIO);
130.     goto error;
131. }
132.
133. /* Create a DIB and select it into the dest_hdc */
134. //BMP
135. bmi.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
136. bmi.bmiHeader.biWidth = clip_rect.right - clip_rect.left;
137. bmi.bmiHeader.biHeight = -(clip_rect.bottom - clip_rect.top);
138. bmi.bmiHeader.biPlanes = 1;
139. bmi.bmiHeader.biBitCount = bpp;
140. bmi.bmiHeader.biCompression = BI_RGB;
141. bmi.bmiHeader.biReserved = 0;

```

```

141.     bmi.bmiHeader.biSizeImage = 0;
142.     bmi.bmiHeader.biXPelsPerMeter = 0;
143.     bmi.bmiHeader.biYPelsPerMeter = 0;
144.     bmi.bmiHeader.biClrUsed = 0;
145.     bmi.bmiHeader.biClrImportant = 0;
146.     hbmp = CreateDIBSection(dest_hdc, &bmi, DIB_RGB_COLORS,
147.         &buffer, NULL, 0);
148.     if (!hbmp) {
149.         WIN32_API_ERROR("Creating DIB Section");
150.         ret = AVERROR(EIO);
151.         goto error;
152.     }
153.
154.     if (!SelectObject(dest_hdc, hbmp)) {
155.         WIN32_API_ERROR("SelectObject");
156.         ret = AVERROR(EIO);
157.         goto error;
158.     }
159.
160.     /* Get info from the bitmap */
161.     GetObject(hbmp, sizeof(BITMAP), &bmp);
162.     //创建AVStream
163.     st = avformat_new_stream(s1, NULL);
164.     if (!st) {
165.         ret = AVERROR(ENOMEM);
166.         goto error;
167.     }
168.     avpriv_set_pts_info(st, 64, 1, 1000000); /* 64 bits pts in us */
169.     //保存信息到GDIGrab上下文结构体
170.     gdigrab->frame_size = bmp.bmWidthBytes * bmp.bmHeight * bmp.bmPlanes;
171.     gdigrab->header_size = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +
172.         (bpp <= 8 ? (1 << bpp) : 0) * sizeof(RGBQUAD) /* palette size */;
173.     gdigrab->time_base = av_inv_q(gdigrab->framerate);
174.     gdigrab->time_frame = av_gettime() / av_q2d(gdigrab->time_base);
175.
176.     gdigrab->hwnd = hwnd;
177.     gdigrab->source_hdc = source_hdc;
178.     gdigrab->dest_hdc = dest_hdc;
179.     gdigrab->hbmp = hbmp;
180.     gdigrab->bmi = bmi;
181.     gdigrab->buffer = buffer;
182.     gdigrab->clip_rect = clip_rect;
183.
184.     gdigrab->cursor_error_printed = 0;
185.
186.     if (gdigrab->show_region) {
187.         if (gdigrab_region_wnd_init(s1, gdigrab)) {
188.             ret = AVERROR(EIO);
189.             goto error;
190.         }
191.     }
192.
193.     st->codec->codec_type = AVMEDIA_TYPE_VIDEO;
194.     st->codec->codec_id = AV_CODEC_ID_BMP;
195.     st->codec->time_base = gdigrab->time_base;
196.     st->codec->bit_rate = (gdigrab->header_size + gdigrab->frame_size) * 1/av_q2d(gdigrab->time_base) * 8;
197.
198.     return 0;
199.
200. error:
201.     //如果出错了
202.     if (source_hdc)
203.         ReleaseDC(hwnd, source_hdc);
204.     if (dest_hdc)
205.         DeleteDC(dest_hdc);
206.     if (hbmp)
207.         DeleteObject(hbmp);
208.     if (source_hdc)
209.         DeleteDC(source_hdc);
210.     return ret;
211. }

```

从源代码可以看出，gdigrab_read_header()的流程大致如下所示：

- (1) 确定窗口的句柄hwnd。如果指定了“title=”的话，调用FindWindow()获取hwnd；如果指定了“desktop”，则设定hwnd为NULL。
- (2) 根据窗口的句柄hwnd确定抓屏的矩形区域。如果抓取指定窗口，则通过GetClientRect()函数；否则就抓取整个屏幕。
- (3) 调用GDI的API完成抓屏的一些初始化工作。包括：
 - a)通过GetDC()获得某个窗口句柄的HDC（在这里是source_hdc）。
 - b)通过CreateCompatibleDC()创建一个与指定设备兼容的HDC（在这里是dest_hdc）
 - c)通过CreateDIBSection()创建HBITMAP
 - d)通过SelectObject()绑定HBITMAP和HDC（指的是dest_hdc）
- (4) 通过avformat_new_stream()创建一个AVStream。
- (5) 将初始化时候的一些参数保存至GDIGrab的上下文结构体。

gdigrab_read_packet()

`gdigrab_read_packet()`用于读取一帧抓屏数据。该函数的定义如下所示。

[cpp]  

```
1.  /**
2.   * Grabs a frame from gdi (public device demuxer API).
3.   *
4.   * @param s1 Context from avformat core
5.   * @param pkt Packet holding the grabbed frame
6.   * @return frame size in bytes
7.   */
8.  static int gdigrab_read_packet(AVFormatContext *s1, AVPacket *pkt)
9.  {
10.     struct gdigrab *gdigrab = s1->priv_data;
11.     //读取参数
12.     HDC     dest_hdc   = gdigrab->dest_hdc;
13.     HDC     source_hdc = gdigrab->source_hdc;
14.     RECT     clip_rect = gdigrab->clip_rect;
15.     AVRational time_base = gdigrab->time_base;
16.     int64_t   time_frame = gdigrab->time_frame;
17.
18.     BITMAPFILEHEADER bfh;
19.     int file_size = gdigrab->header_size + gdigrab->frame_size;
20.
21.     int64_t curtime, delay;
22.
23.     /* Calculate the time of the next frame */
24.     time_frame += INT64_C(1000000);
25.
26.     /* Run Window message processing queue */
27.     if (gdigrab->show_region)
28.         gdigrab_region_wnd_update(s1, gdigrab);
29.
30.     /* wait based on the frame rate */
31.     //延时
32.     for (;;) {
33.         curtime = av_gettime();
34.         delay = time_frame * av_q2d(time_base) - curtime;
35.         if (delay <= 0) {
36.             if (delay < INT64_C(-1000000) * av_q2d(time_base)) {
37.                 time_frame += INT64_C(1000000);
38.             }
39.             break;
40.         }
41.         if (s1->flags & AVFMT_FLAG_NONBLOCK) {
42.             return AVERROR(EAGAIN);
43.         } else {
44.             av_usleep(delay);
45.         }
46.     }
47.     //新建一个AVPacket
48.     if (av_new_packet(pkt, file_size) < 0)
49.         return AVERROR(ENOMEM);
50.     pkt->pts = curtime;
51.
52.     /* Blit screen grab */
53.     //关键: BitBlt()完成抓屏功能
54.     if (!BitBlt(dest_hdc, 0, 0,
55.                 clip_rect.right - clip_rect.left,
56.                 clip_rect.bottom - clip_rect.top,
57.                 source_hdc,
58.                 clip_rect.left, clip_rect.top, SRCCOPY | CAPTUREBLT)) {
59.         WIN32_API_ERROR("Failed to capture image");
60.         return AVERROR(EIO);
61.     }
62.     //画鼠标指针?
63.     if (gdigrab->draw_mouse)
64.         paint_mouse_pointer(s1, gdigrab);
65.
66.     /* Copy bits to packet data */
67.     //BMP文件头BITMAPFILEHEADER
68.     bfh.bfType = 0x4d42; /* "BM" in little-endian */
69.     bfh.bfSize = file_size;
70.     bfh.bfReserved1 = 0;
71.     bfh.bfReserved2 = 0;
72.     bfh.bfOffBits = gdigrab->header_size;
73.     //往AVPacket中拷贝数据
74.     //拷贝BITMAPFILEHEADER
75.     memcpy(pkt->data, &bfh, sizeof(bfh));
76.     //拷贝BITMAPINFOHEADER
77.     memcpy(pkt->data + sizeof(bfh), &gdigrab->bmi.bmiHeader, sizeof(gdigrab->bmi.bmiHeader));
78.     //不常见
79.     if (gdigrab->bmi.bmiHeader.biBitCount <= 8)
80.         GetDIBColorTable(dest_hdc, 0, 1 << gdigrab->bmi.bmiHeader.biBitCount,
81.                           (RGBQUAD *) (pkt->data + sizeof(bfh) + sizeof(gdigrab->bmi.bmiHeader)));
82.     //拷贝像素数据
83.     memcpy(pkt->data + gdigrab->header_size, gdigrab->buffer, gdigrab->frame_size);
84.
85.     gdigrab->time_frame = time_frame;
86.
87.     return gdigrab->header_size + gdigrab->frame_size;
88. }
```


从源代码可以看出，gdigrab_read_packet()的流程大致如下所示：

- (1) 从GDIGrab上下文结构体读取初始化时候设定的参数。
- (2) 根据帧率参数进行延时。
- (3) 通过av_new_packet()新建一个AVPacket。
- (4) 通过BitBlt()完成抓屏功能。
- (5) 如果需要画鼠标指针的话，调用paint_mouse_pointer()，这里不做分析。
- (6) 按照顺序拷贝以下3项内容至AVPacket的data指向的内存：
 - a)BITMAPFILEHEADER
 - b)BITMAPINFOHEADER
 - c)抓屏的到的像素数据

gdigrab_read_close()

gdigrab_read_close()用于关闭gdigrab。该函数的定义如下所示。

```
[cpp]
1.  /**
2.   * Closes gdi frame grabber (public device demuxer API).
3.   *
4.   * @param s1 Context from avformat core
5.   * @return 0 success, !0 failure
6.   */
7.  static int gdigrab_read_close(AVFormatContext *s1)
8.  {
9.      struct gdigrab *s = s1->priv_data;
10.
11.      if (s->show_region)
12.          gdigrab_region_wnd_destroy(s1, s);
13.
14.      if (s->source_hdc)
15.          ReleaseDC(s->hwnd, s->source_hdc);
16.      if (s->dest_hdc)
17.          DeleteDC(s->dest_hdc);
18.      if (s->hbmpp)
19.          DeleteObject(s->hbmpp);
20.      if (s->source_hdc)
21.          DeleteDC(s->source_hdc);
22.
23.      return 0;
24.  }
```

从源代码可以看出，gdigrab_read_close ()完成了各种变量的清理工作。

雷霄骅

leixiaohua1020@126.com

<http://blog.csdn.net/leixiaohua1020>

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/44597955>

文章标签： [FFmpeg](#) [GDI](#) [抓屏](#) [屏幕录像](#) [源代码](#)

个人分类： [FFMPEG](#)

所属专栏： [FFmpeg](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com