DJYOS
图形编程
（上）

DJYOS 图形装饰平台用户手册

编写：刘巍 2015 年 02 月 12 日
Review：_年_月_日
审阅:罗侍田

1. GDD 概述

GDD 是依赖于 Djyos 内核与 **GK**(GUI Kernel)提供的一些基础功能函数来实现的更高一级的图形界面组件。它可向用户提供图形绘制，文字显示，窗口/控件管理，窗口定时器，消息通讯机制，以及对外部输入事件(键盘，鼠标)的处理功能。

在整个系统中，它们的层次关系如图 1-1 所示：

图 1-1 GDD 在 Djyos 系统中层次示意图

GDD 组件由窗口系统、绘图系统、消息系统、定时器系统、矩形区域运算以及外部输入系统等部分组成。窗口系统顾名思义就是图形、文字等信息显示的载体，每个窗口都有一个过程函数，即所谓的钩子函数，对窗口的管理及响应均是在过程函数中实现；绘图系统用于图形绘制；GDD 是基于消息事件驱动机制，消息系统是窗口间通讯以及窗口与用户交互的桥梁，没有它，用户界面只是表现在显示器上的一堆图片，无法响应用户给出的动作指令。通俗点来描述，对用户来讲：消息系统的主要功能就是系统告诉用户发生了什么和用户通知系统需要干什么，用户通过发送消息触发特定窗口的过程函数执行进而完成特定功能。定时器系统用于 GDD 定时服务；矩形区域计算用于 GDD 窗口管理及图形绘制过程对特定矩形区域的计算；外部输入用于 GDD 对外部信息(如鼠标、键盘等)的获取。本文会针对 GDD 各部分做出说明，在此之前先简单展示一下 Djyos GDD 组件的强大功能。

一个典型的 GDD 窗口应用程序步骤流程如下：

1. 创建主窗口：这里指定主窗口过程函数，设置位置，大小等参数；
2. 显示窗口：设置主窗口为可见状态；
3. 窗口消息处理：它包含三部分，从窗口消息中取出一条消息、将该消息派发给窗口的过程函数、窗口的过程函数处理消息。

实例代码如下：

GDD 是基于 GK 提供的部分基础函数实现，因此在使用 GDD 之前首先需要完成初始化 GK、初始化硬件显示屏、创建桌面等工作。

```
//在系统组件初始化函数中初始化 GK、初始化硬件显示屏、创建桌面。
void Sys_Module_Init(void)
{
ModuleInstall_GK (0);       //GK初始化
```

```
lcd_2440 =ModuleInstall_LCD(320,240,”LCD_S3C2440”);//硬件显示屏初始化
GK_ApiCreateDesktop(lcd_2440&desktop,0,0,     //创建桌面
                                CN_COLOR_RED+CN_COLOR_GREEN,CN_WINBUF_BUF,0,0);
}
```

在 Sys_Module_Init 函数中完成了 GK、硬件显示屏初始化及创建桌面工作后，应用程序只需调用系统函数 ModuleInstall_GDD 即可启动 GDD 组件。接下来应用程序即可完成自己所需的图形、文字处理工作了。

应用程序工作包含两部分：窗口入口函数及窗口过程函数。

```
//入口函数
void       WDD_Demo_ZigBee(void)
{
      HWND hwnd;
      MSG msg;
      RECT rc;
      int y;
      GetClientRect(GetDesktopWindow(),&rc);
      InflateRect(&rc,-20,-20);
      //创建主窗口
      hwnd = CreateWindow(win_proc,L"ZigBee 测试",WS_MAIN_WINDOW,
                                rc.left,rc.top,RectW(&rc),RectH(&rc),NULL,0x0000,NULL);
      ShowWindow(hwnd,TRUE);   //显示窗口
      while(GetMessage(&msg,hwnd))
      {
            DispatchMessage(&msg);
      }
}
//窗口过程函数
static    u32 win_proc(MSG *pMsg)
{
      HWND hwnd;
      HDC hdc;
      RECT rc,rc0;
      u32 i,x,y,w;
      hwnd =pMsg->hwnd;
      switch(pMsg->Code)
      {
            case       MSG_CREATE:
                        timer_500ms_count=0;
                        memset(ZigBeeTextBuf,0,sizeof(ZigBeeTextBuf));
                        memset(ZigBeeTimeBuf,0,sizeof(ZigBeeTimeBuf));
                        zigbee_ShowString("Zigbee Text");
                        hwndZigBee =hwnd;
                        GetClientRect(hwnd,&rc0);
                        CreateWindow(BUTTON,L"关闭
",WS_BORDER|WS_DLGFRAME|WS_CHILD|BS_SIMPLE|WS_VISIBLE,RectW(&rc0)-60,RectH(&rc0)-60,56,56,hwnd,
ID_CLOSE,NULL);
                        SetRect(&rcZigBeeString,4,4,RectW(&rc0)-4*2,60);
                        x=4;
                        y=RectH(&rc0)-70;
                        SetRect(&rcGroupBox_LED,x,y,100,66);
                        SetRect(&rcGroupBox_CTR,x+100+4,y,100,66);
                        x=rcGroupBox_LED.left+12;
                        y=rcGroupBox_LED.top+18;
CreateWindow(BUTTON,L"LED1",WS_BORDER|WS_CHILD|BS_HOLD|WS_VISIBLE,x,y+0*24,72,20,hwnd,ID_LED1
,NULL);
CreateWindow(BUTTON,L"LED2",WS_BORDER|WS_CHILD|BS_HOLD|WS_VISIBLE,x,y+1*24,72,20,hwnd,ID_LED2
,NULL);
                        x=rcGroupBox_CTR.left+12;
                        y=rcGroupBox_CTR.top+18;
                        CreateWindow(BUTTON,L"组网
",WS_BORDER|WS_CHILD|WS_VISIBLE,x,y+0*24,72,20,hwnd,ID_NET_ON,NULL);
                        CreateWindow(BUTTON,L"断开
",WS_BORDER|WS_CHILD|WS_VISIBLE,x,y+1*24,72,20,hwnd,ID_NET_OFF,NULL);
                        CreateWindow(BUTTON,L"发送
",WS_BORDER|WS_CHILD|BS_SIMPLE|WS_VISIBLE,RectW(&rc0)-60,rcGroupBox_CTR.top-(24+2),56,24,hwnd,ID_S
END,NULL);
```

```
                GDD_CreateTimer(hwnd,0,500,TMR_START);
                break;
        case    MSG_TIMER:
                {
                    timer_500ms_count++;
                    WDD_wsprintf(ZigBeeTimeBuf,L"TIME:%06d",timer_500ms_count);
                    InvalidateWindow(hwnd);
                }
                break;
        case    MSG_NOTIFY:
                {
                    u16 event,id;
                    event =HI16(pMsg->Param1);
                    id =LO16(pMsg->Param1);
                    switch(event)
                    {
                        case BTN_DOWN:   //按钮按下
                            if(id==ID_LED1)
                            {
                                zigbee_LED1_ON();
                            }
                            if(id==ID_LED2)
                            {
                                zigbee_LED2_ON();
                            }
                            if(id==ID_NET_ON)
                            {
                                zigbee_NET_ON();
                            }
                            if(id==ID_NET_OFF)
                            {
                                zigbee_NET_OFF();
                            }
                            if(id==ID_SEND)
                            {
                                char buf[64];
                                unicode_to_ansi(buf,ZigBeeTextBuf,64);
                                zigbee_send_string(buf);
                            }
                            break;
                        case BTN_UP:       //按钮弹起
                            if(id==ID_CLOSE)
                            {
                                PostMessage(hwnd,MSG_CLOSE,0,0);
                            }
                            if(id==ID_LED1)
                            {
                                zigbee_LED1_OFF();
                            }
                            if(id==ID_LED2)
                            {
                                zigbee_LED2_OFF();
                            }
                            break;
                    }
                }
                break;
        case    MSG_PAINT:
                {
                    hdc =BeginPaint(hwnd);
                    GetClientRect(hwnd,&rc0);
                    SetFillColor(hdc,RGB(150,150,150));
                    FillRect(hdc,&rc0);
                    SetTextColor(hdc,RGB(0,0,0));
                    SetDrawColor(hdc,RGB(100,100,100));
                    DrawGroupBox(hdc,&rcZigBeeString,L"信息接收区");
                    SetRect(&rc, rcGroupBox_LED.left, rcGroupBox_LED.top-(24+2),
```

```
                        RectW(&rcGroupBox_LED),24);
                        if(zigbee_is_ok())
                        {
                            SetTextColor(hdc,RGB(0,255,0));
                            SetDrawColor(hdc,RGB(0,200,0));
                            SetFillColor(hdc,RGB(0,128,0));
                            DrawText(hdc,L"连接成功
",-1,&rc,DT_CENTER|DT_VCENTER|DT_BORDER|DT_BKGND);
                        }
                        else
                        {
                            SetTextColor(hdc,RGB(255,0,0));
                            SetDrawColor(hdc,RGB(200,0,0));
                            SetFillColor(hdc,RGB(100,0,0));
                            if(timer_500ms_count&0x01)
                            {
                                DrawText(hdc,L"未连接
",-1,&rc,DT_CENTER|DT_VCENTER|DT_BORDER|DT_BKGND);
                            }
                            else
                            {
                                DrawText(hdc,L" ",-1,&rc,DT_CENTER|DT_VCENTER|DT_BORDER|DT_BKGND);
                            }
                        }
                        SetRect(&rc,rcGroupBox_CTR.left,rcGroupBox_CTR.top-(24+2),
                                RectW(&rcGroupBox_CTR),24);
                        SetTextColor(hdc,RGB(0,255,255));
                        SetDrawColor(hdc,RGB(0,200,200));
                        SetFillColor(hdc,RGB(0,80,80));
DrawText(hdc,ZigBeeTimeBuf,-1,&rc,DT_CENTER|DT_VCENTER|DT_BORDER|DT_BKGND);
                        SetTextColor(hdc,RGB(0,255,0));
                        SetDrawColor(hdc,RGB(0,200,0));
                        SetFillColor(hdc,RGB(0,80,0));
                        CopyRect(&rc,&rcZigBeeString);
                        InflateRectEx(&rc,-4,-20,-4,-4);
DrawText(hdc,ZigBeeTextBuf,-1,&rc,DT_LEFT|DT_VCENTER|DT_BORDER|DT_BKGND);
                        SetTextColor(hdc,RGB(0,0,128));
                        SetDrawColor(hdc,RGB(80,80,80));
                        DrawGroupBox(hdc,&rcGroupBox_LED,L"LED控制");
                        DrawGroupBox(hdc,&rcGroupBox_CTR,L"网络控制");
                        EndPaint(hwnd,hdc);
                    }
                    break;
            default:
                    return    DefWindowProc(pMsg);
    }
    return    0;
}
```

图 1-2 GDD 效果示例图

2. 窗口系统

2.1. 窗口分类及关系

GDD 中窗口分为三大类：桌面窗口、主窗口、控件。

桌面窗口是整个系统唯一的一个全局根窗口，它在系统启动时，由系统自动创建，如果用户要对桌面窗口进行操作，可以先通过 GetDesktopWindow 获得桌面窗口句柄。可以使用相应的窗口 API 函数集对其进行操作。

主窗口由用户创建，是用户图形窗口程序必须创建的第一个窗口。是所有控件的载体，也负责整个用户窗口程序的消息事件收集与派发。

控件是在主窗口创建之后，由用户创建。控件不是必须的，一个窗口系统必须至少有一个主窗口，而控件则是用户按实际应用情况，是否使用；控件和主窗口一样，都是使用 CreateWindow 函数来创建它，与创建主窗口不同的是：当需要创建的窗口为控件时，必须指定该窗口为 WS_CHILD 风格，需要指明控件所属的父窗口，以及为每个控件指定一个同级窗口中的唯一 ID，具体请参考 CreateWindow： 创建窗口。GDD 提供了一些常用标准控件给用户使用，如按钮、复选框、单选框等。

三种不同类型窗口示例如图 2-1 所示：

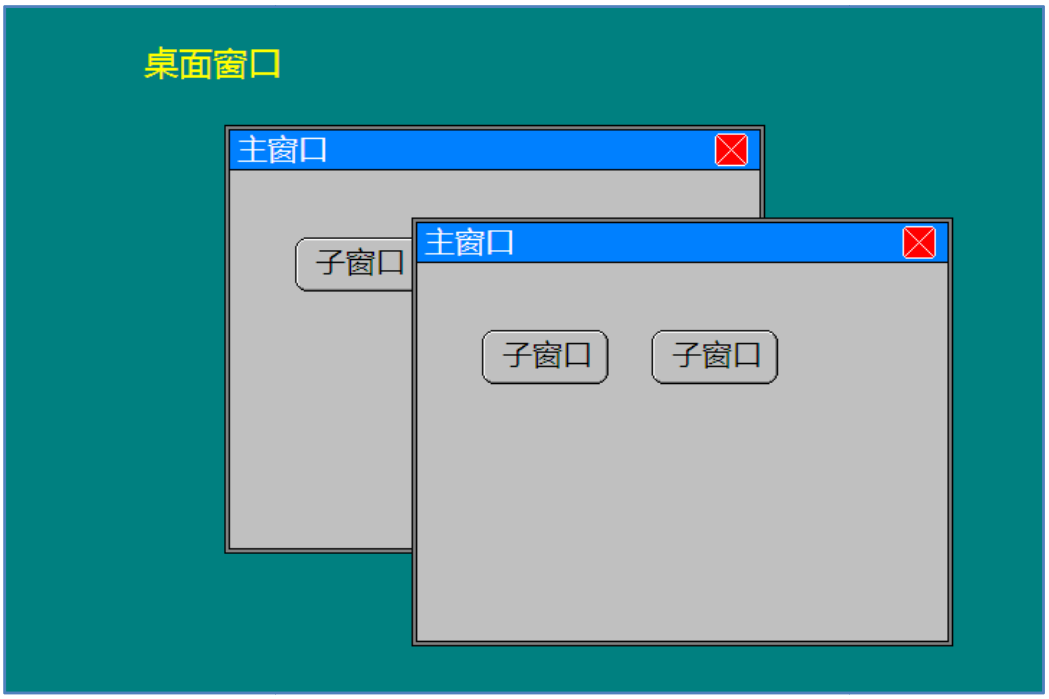

图 2-1GDD 三种窗口示意图

2.2. 窗口的客户区与非客户区

窗口区域分为客户区与非客户区两部分。在一个窗口中，标题栏，边框就是窗口的非客户区，中间区域称为客户区。它们的关系如图 2-2 所示：

图 2-2 客户区示意图

2.3. 坐标系统

GDD 的所有坐标都是像素数坐标，有三种坐标系统，相互之间是可以转换的，这三种坐标系统分别是：屏幕坐标，窗口坐标，客户坐标。

屏幕坐标以屏幕左上角为坐标原点，水平方向为 X 值，垂直方向为 Y 值构成坐标系。

窗口坐标以窗口左上角为坐标原点，水平方向为 X 值，垂直方向为 Y 值构成坐标系。

客户坐标以窗口客户区左上角为坐标原点，水平方向为 X 值、垂直方向为 Y 值构成坐标系。

三种坐标关系示意图 2-3 如下：

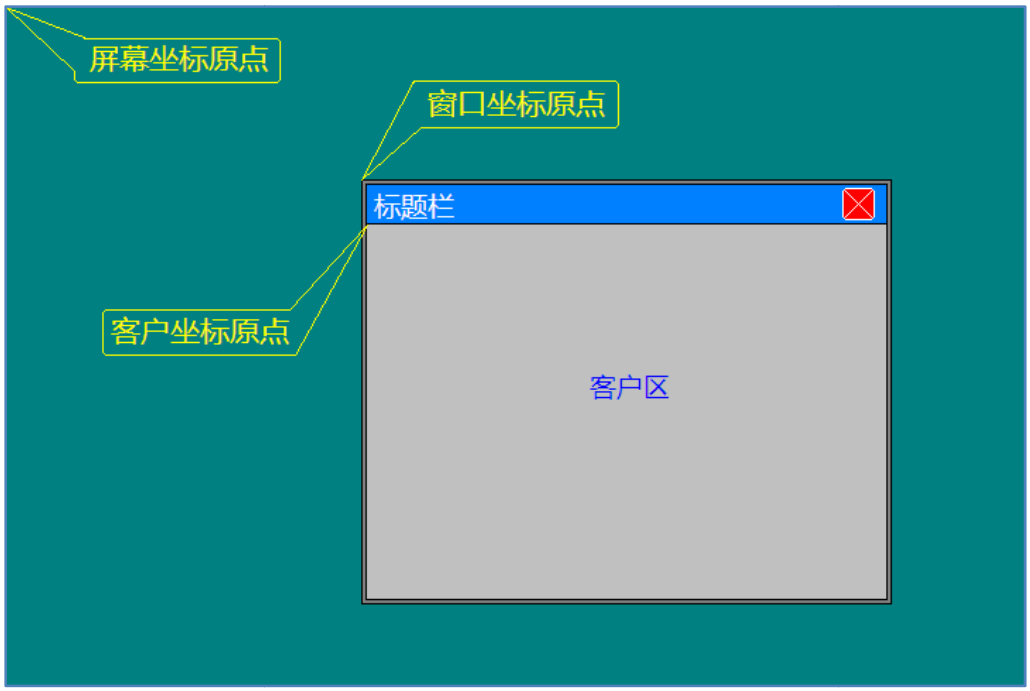

图 2-3 GDD 三种坐标示意图

2.4. 窗口句柄与窗口 ID 的作用与区别

在窗口程序中，用户会经常和窗口句柄，窗口 ID 打交道，窗口句柄是由系统自动分配的，无需用户干预，对窗口操作访问，都只能通过句柄进行;而窗口 ID，是在创建窗口时，由用户指定的一个任意 32 位数，一般用于对同级窗口(兄弟窗口)间的标识区分，同级窗口间，它们的 ID 需由用户创建窗口时保证唯一性，对于不同父窗口下的子窗口，它们的 ID 是可以重复的;例如一个窗口 A 和窗口 B，它们都可以创建一个指定 ID 为 1 的子窗口，这是合法的，另外补充一点，窗口 ID 是针对来标识子窗口的，因此，桌面窗口及所有主窗口是没有 ID 标识的概念的，当用户创建一个主窗口时，系统会忽略 CreateWindow 函数的 ID 参数值，具体请参数 CreateWindow 函数。

2.5. 窗口的关闭、销毁、退出过程

这个步骤流程如下：

1.发送 MSG_CLOSE 消息，该消息可以由系统产生(比如用户点击窗口的”×”按钮，也可以是用户主动发送 MSG_CLOSE 消息(使用 SendMessage 函数)。

2.用户在窗口过程函数中响应 MSG_CLOSE 消息，在这一步中，用户还能决定是否继续真正关闭和销毁窗口，如果不需要，则直接从该消息响应过程中返回，否则，用户需调用 DestroyWindow 函数来继续销毁窗口。

3.响应 MSG_DESTROY 消息，该消息是由 DestroyWindow 函数产生的，到了这里，窗口关闭、销毁已经是“无可挽回”的地步，用户可以在这里做一些资源释放工作，如果销毁的是主窗口，用户必须要在 MSG_DESTROY 消息响应过程中调用 PostQuitMessage 函数，如果是子窗口，则不需要调用，子窗口关闭、销毁过程也至此结束。

4.如果是主窗口，由于在 MSG_DESTROY 中调用了 PostQuitMessage 函数，该函数会产生一条 MSG_QUIT 消息，当主窗口消息循环中的 GetMessage 函数获得到 MSG_QUIT 消息时，会在该函数返回前销毁主窗口，并返回 FALSE，如是，整个窗口消息循环至此结束。

2.6. API 说明

2.6.1. ScreenToClient： 屏幕坐标转换为客户区坐标

BOOL ScreenToClient(HWND hwnd，POINT *pt,s32 count);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

pt： 需要转换的坐标点，该参数兼作为转换的输入和输出参数。

count： 需要转换的坐标点数量。

返回值：

TRUE：成功； FALSE：失败。

说明：

如果函数返回失败，原因可能是窗口句柄无效，参数 pt 指针为 NULL；另外，该函数不关心需要转换的坐标是否在窗口可视范围内。

2.6.2. ClientToScreen： 客户区坐标转换为屏幕坐标

BOOL ClientToScreen(HWND hwnd, POINT *pt, s32 count);

头文件：

gdd.h

参数:

Hwnd： 窗口句柄。

pt： 需要转换的坐标点，该参数兼作为转换的输入和输出参数。

count： 需要转换的坐标点数量。

返回值：

TRUE：成功; FALSE：失败。

说明：

如果函数返回失败，原因可能是窗口句柄无效，参数 pt 指针为 NULL;另外，该函数不处理需要转换的坐标是否在窗口可视范围内。

2.6.3. ScreenToWindow：屏幕坐标转换为窗口坐标

BOOL ScreenToWindow(HWND hwnd,POINT *pt,s32 count);

头文件:

gdd.h

参数:

hwnd: 窗口句柄.

pt： 需要转换的坐标点，该参数兼作为转换的输入和输出参数。

count： 需要转换的坐标点数量。

返回值：

TRUE：成功; FALSE： 失败。

说明：

如果函数返回失败，原因可能是窗口句柄无效，参数 pt 指针为 NULL;另外，该函数不处理需要转换的坐标是否在窗口可视范围内。

2.6.4. WindowToScreen： 窗口坐标转换为屏幕坐标

BOOL WindowToScreen(HWND hwnd,POINT *pt,s32 count);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

pt： 需要转换的坐标点，该参数兼作为转换的输入和输出参数。

count： 需要转换的坐标点数量。

返回值：

TRUE：成功; FALSE：失败。

说明：

如果函数返回失败，原因可能是窗口句柄无效，参数 pt 指针为 NULL;另外，该函数不处理需要转换的坐标是否在窗口可视范围内。

2.6.5. GetDesktopWindow： 获得桌面窗口句柄

HWND GetDesktopWindow(void);

头文件：

gdd.h

参数： 无。

返回：

桌面窗口句柄。

2.6.6. GetWindowRect： 获得窗口矩形

BOOL GetWindowRect(HWND hwnd,RECT *prc);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

prc： 矩形缓冲区。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数获得窗口边界在屏幕上的矩形位置，以屏幕坐标表示。

2.6.7. GetClientRect： 获得窗口客户区矩形

BOOL GetClientRect(HWND hwnd,RECT *prc);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

prc： 矩形缓冲区。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数获得窗口客户区矩形，以客户区坐标表示，所以获得的矩形左上角坐标恒为 0，0;

矩形右下角坐标所指示的为客户区宽度和高度。

2.6.8. GetClientRectToScreen： 取窗口客户区矩形的屏幕坐标

BOOL GetClientRectToScreen(HWND hwnd,RECT *prc);

头文件：

gdd.h

参数：

hwnd： 窗口坐标。

prc： 矩形缓冲区。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数获得窗口客户区边界在屏幕上的矩形位置，以屏幕坐标表示。

2.6.9. GetWindowDC： 获得窗口绘图上下文

HDC GetWindowDC(HWND hwnd);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

返回值：

窗口绘图上下文句柄。

说明：

该函数获得的绘图上下文，可以在整个窗口范围内(非客户区+客户区)绘图。绘图上下文的原点坐标(0，0)为窗口边界最左上角，在结束绘图后，需调用 ReleaseDC 来释放该绘图上下文。

2.6.10. GetDC：获得窗口客户区绘图上下文

HDC GetDC(HWND hwnd);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

返回值：

窗口客户区绘图上下文句柄。

说明：

该函数获得的绘图上下文，只能在窗口客户区范围内绘图，绘图上下的原点坐标(0，0)为窗口客户区边界最左上角，在结束绘图后，需调用 ReleaseDC 来释放该绘图上下文。

2.6.11. ReleaseDC： 释放一个绘图上下文

BOOL ReleaseDC(HWND hwnd,HDC hdc);

头文件：

gdd.h

参数:：

hwnd： 绘图上下文所属的窗口句柄。

hdc： 需要释放的绘图上下文句柄。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数只能释放由 GetWindowDC，GetDC 所返回的绘图上下文句柄。

2.6.12. BeginPaint： 开始绘图

HDC BeginPaint(HWND hwnd);

头文件：

gdd.h

参数：

hwnd： 需要开始绘图的窗口句柄。

返回值：

绘图上下文句柄。

说明：

该函数只在窗口过程函数的 MSG_PAINT 消息中使用，指示客户区开始绘图;该函数返回的绘图上下文句柄，只能在窗口客户区范围内绘图， 原点坐标(0，0)为窗口客户区边界最左上角， 在结束绘图后，需调用 EndPaint 来释放该绘图上下文。

2.6.13. EndPaint： 结束绘图

BOOL EndPaint(HWND hwnd,HDC hdc);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

hdc： 需要结束绘图的绘图上下文句柄。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数只在窗口过程函数的 MSG_PAINT 消息中与 BeginPaint 成对使用，指示绘图结束并释放由 BeginPaint 所返回的绘图上下文句柄。

2.6.14. CreateWindow： 创建窗口

HWND CreateWindow(WNDPROC *pfWinProc,LPCWSTR Text,u32 Style,
s32 x, s32 y, s32 w, s32 h,HWND hParent,u32 WinId,const void *pdata);

头文件：

gdd.h

参数：

pfWinProc： 窗口过程函数地址。

Text： 窗口标题文字。

Style： 窗口风格属性标记，高 16 位为窗口公共风格属性标记，所有窗口适用；低 16 位窗口私有的风格属性标记，代表窗口的私有行为或外观，对于不同类型的窗口，私有风格属性值所代表的含义，仅作用于本身，不可以用于其它不同类型的窗口;不管是公共还是私有风格属性，系统都定义了相关的宏常量供用户使用，对于窗口公共风格属性，可以是以下值的组合：

WS_CHILD： 设定该标志，则创建为子窗口/控件；

WS_VISIBLE： 设定该标志，则窗口为可见的；

WS_DISABLE： 设定该标志，刚窗口为禁止状态，将不会响应外部的输入消息，如键盘，鼠标消息等等；

WS_BORDER： 设定该标志，则窗口会包含并显示外边框；
WS_DLGFRAME： 设定该标志，则窗口会包含并显示内边框；
WS_CAPTION： 设定该标志，刚窗口会包含并显示标题栏；
私有风格属性有以下定义，详情见 gdd.h。

// 按钮风格
BS_TYPE_MASK： 类型掩码；
BS_NORMAL： 常规按钮；
BS_HOLD： 自锁按钮；
BS_RADIO： 单选按钮；
BS_SURFACE_MASK： 外观掩码；
BS_NICE： 美观风格；
BS_SIMPLE： 简朴风格；
BS_FLAT： 平面风格；
BS_PUSHED： 按钮按下；

// 复选框风格
CBS_SELECTED： 复选框选中。

x，y，w，h： 指示窗口的位置和大小，位置是相对于该窗口所属的父窗口的客户区偏移量，即使用的是客户区坐标表示。大小表示窗口的水平和垂直方向所占的像素数量。

hParent：父窗口句柄，如果是创建主窗口(没有指定 WS_CHILD 属性)，该参数设为 NULL。

WinId： 用户指定的窗口 ID，如果是主窗口，该参数会被忽略; 如果是同一级子窗口，则需要为每个指定为不同 ID，如果指定了一个已有的同级窗口 ID，该函数将创建窗口失败，返回 NULL。

pData： 用户自定义的窗口私有数据，该参数会传入窗口过程的 MSG_CREATE 消息的 param1 中，如果不需要窗口私有数据，可以将该参数设为 NULL。

返回值：

新创建的窗口句柄;如果创建失败，该函数返回 NULL。

2.6.15. DestroyWindow： 销毁窗口

```
void DestroyWindow(HWND hwnd);
```

头文件：

gdd.h

参数：

hwnd： 需要销毁的窗口句柄。

说明：

该函数用来销毁一个窗口，除了销毁窗口本身，也会销毁窗口所属的所有子窗口和定时器，如果用户不调用该函数来销毁所创建的子窗口，当主窗口退出时，也会自行销毁所有子窗口和定时器，如果所销毁的窗口是焦点窗口，那么它所在的父窗口的焦点窗口对象会被设为 NULL。

2.6.16. MoveWindow： 移动窗口位置

```
BOOL MoveWindow(HWND hwnd,s32 x, s32 y);
```

头文件：

gdd.h

参数：

hwnd： 需要移动的窗口句柄。

x，y： 移动后，新的坐标位置。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数使用的坐标是相对于父窗口的客户区位置，即父窗口的客户区坐标。如果移动一个窗口并超出它所属的父窗口客户区范围,那么,超出的部分是会被裁剪的(不可见).如图 2-4 所示:

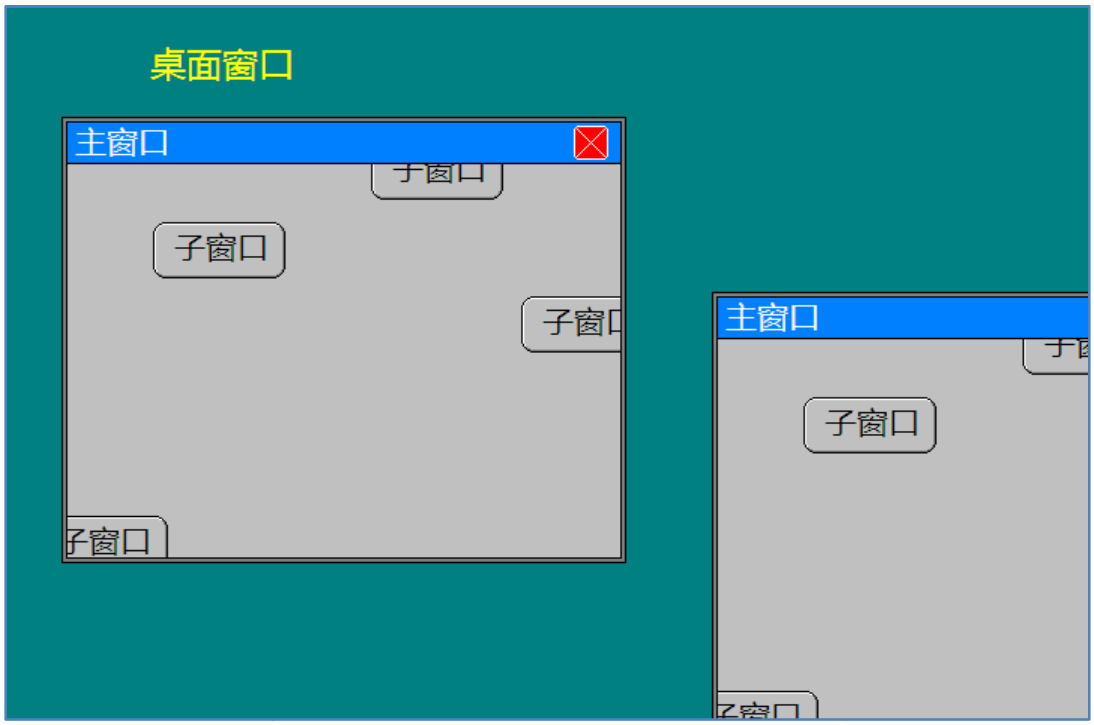

图 2-5 窗口裁剪示意图

2.6.17. OffsetWindow： 偏移窗口位置

BOOL OffsetWindow(HWND hwnd,int dx,int dy);

头文件：

gdd.h

参数：

hwnd： 需要偏移的窗口句柄。

dx，dy： 水平，垂直方向的偏移量。

返回值：

TRUE：成功; FALSE：失败。

说明：

当 dx 为负数时，窗口向左偏移，否则向右偏移;当 dy 为负数时，窗口向上偏移，否则向下偏移。

2.6.18. IsWindowVisible： 判断窗口是否可见

BOOL IsWindowVisible(HWND hwnd);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

返回值：

TRUE：窗口可见; FALSE：窗口不可见。

说明：

如果一个窗口为不可见，那么它所属的所以子窗口也都为不可见，即便子窗口设置了 WS_VISIBLE 标记。更多的内容，请参考 ShowWindow 函数说明。

2.6.19. InvalidateWindow： 设置窗口为无效状态

BOOL InvalidateWindow(HWND hwnd，BOOL bErase);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

bErase：是否重绘窗口背景。

返回值：

TRUE：成功; FALSE：失败。

说明：

设置窗口为无效状态，即意味着窗口需要进行重绘，该函数会给指定的窗口产生一条异步方式的窗口绘制消息：MSG_PAINT，MSG_PAINT 是属于低优先级处理的消息，只有当其它消息处理完成后(MSG_TIMER 除外，这个优先级更低)，才会处理窗口绘制操作，如果系统因繁忙导致前面的绘制消息未能及时处理，系统将对后续重复的同一个窗口的多个绘制消息合并为一个，以减少系统负担，以过滤掉不必要的多余的绘制过程。当 bErase 参数为 TRUE 时，窗口过程将会在执行 MSG_PAINT 前，发送一条 MSG_ERASEBKGND 消息，用于窗口背景重绘，用户可以在窗口过程函数中，响应 MSG_ERASEBKGND 消息来实现窗口背景的绘制工作。

2.6.20. ShowWindow： 设置窗口为显示或隐藏状态

BOOL ShowWindow(HWND hwnd,BOOL bShow);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

bShow： 显示标记，当为 TRUE 时，设置窗口为显示状态(可见状态);为 FALSE 时，将设置窗口为隐藏状态。

返回值：

TRUE： 成功; FALSE：失败。

说明：

该函数为设置窗口是否显示或隐藏，即设置或清除窗口的 WS_VISIBLE 标记，需要注意的是：如果该窗口所属的父窗口为隐藏状态(不可见)，那么它的所有子窗口都不会被显示，即便子窗口设置为显示状态(WS_VISIBLE);改变父窗口的显示状态标志时，并不会改变它所属的子窗口显示状态标志。例如，假设起先一个父窗口 A 和它的一个子窗口 B 都为隐藏状态(不可见，无 WS_VISIBLE 标记)，这时，如果将父窗口设置为显示状态(可见，设置了 WS_VISIBLE 标记)，那么此时，它的子窗口 B 仍然不会显示出来。总结来说，一个窗口真正可见，除了窗口本身具备可见属性(WS_VISIBLE)之外，还必须它的所有上级父窗口同时都具备可见属性。

2.6.21. EnableWindow： 设置窗口为使能或禁止状态

BOOL EnableWindow(HWND hwnd,BOOL bEnable);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

bEnable： 标记状态，当为 TRUE 时，设置窗口为使能状态;为 FALSE 时，设置窗口为禁止状态。

返回值：

TRUE：成功; FALSE：失败。

说明：

该函数为设置窗口是否使能或禁止，即设置或清除窗口的 WS_DISABLE 标记，当窗口为禁止状态(设置了 WS_DISABLE 标记)时，它将不能接收到外部输入事件产生的消息，如键盘，鼠标/触摸屏产生的消息，但如果用户通过 SendMessage/PostMessage 函数强制向窗口发送这类消息，窗口是可以正常接收到的;一个窗口只有当他的上级所有父窗口及窗口本身都使能时，该窗口才真正允许接收到外部输入事件产生的消息，这种关系与 ShowWindow 函数类似。需要注意的是禁止状态的窗口是可见的并且能绘制。

2.6.22. GetParent： 获得父窗口句柄

HWND GetParent(HWND hwnd);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

返回值：

父窗口句柄。

2.6.23. GetWindow： 获得与指定窗口有特定关系的窗口句柄

HWND GetWindow(HWND hwnd,int nCmd);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

nCmd： 说明指定窗口与要获得句柄的窗口之间的关系，该参数值可以是下列之一：

GW_CHILD： 获得 z 序顶端的子窗口。

GW_HWNDPREV： 获得 z 序上一个同级窗口。

GW_HWNDNEXT： 获得 z 序下一个同级窗口。

GW_HWNDFIRST： 获得 z 序顶层的同级窗口。

GW_HWNDLAST： 获得 z 序底层的同级窗口。

返回值：

特定关系的窗口句柄。

2.6.24. GetDlgItem： 获得窗口中指定 ID 的子窗口的句柄

HWND GetDlgItem(HWND hwnd,u32 id);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

id： 所属子窗口 ID。

返回值：

子窗口句柄。

2.6.25. SetWindowText：设置窗口文字

void SetWindowText(HWND hwnd,const char *text,s32 max_len);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

text：需要设置的字符串内容。

max_len：最大字符串字节长度，如果该值是负数,则使用整个字符串实际字节长度；如果该值小于实际的字符串字节长度,且大于 0，则字符串多余的部分将会被截断；如果该值大于实际的字符串字节长度，则按实际的字符串长度来处理。

返回值：

无。

2.6.26. GetWindowText：获得窗口文字

char* GetWindowText(HWND hwnd,char *text,s32 max_len);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

text：输出的字符串缓冲区。

max_len：最大字符串字节长度，如果该值是负数,则使用窗口整个字符串实际字节长度；如果该值小于实际的字符串字节长度,且大于 0，则字符串多余的部分将会被截断；如果该值大于窗口实际的字符串字节长度，则按实际的字符串长度来处理。

返回值：

字符串指针。

2.6.27. GetWindowFromPoint： 获得指定屏幕坐标位置所在的窗口

HWND GetWindowFromPoint(POINT *pt);

头文件：

gdd.h

参数：

pt： 坐标位置点参数，该坐标值使用屏幕坐标。

返回值：

坐标所在的窗口句柄。

说明：

在多级窗口从属关系中，该函数返回的窗口句柄，将会是坐标所在位置的最低级别的窗口。

2.6.28. SetFocusWindow：设置当前焦点窗口

HWND SetFocusWindow(HWND hwnd);

头文件：

gdd.h

参数：

hwnd： 新的焦点窗口句柄，如果该值为 NULL，则会清除当前焦点窗口。

返回值：

旧的焦点窗口句柄，如果当前无焦点窗口，返回 NULL。

说明：

调用该函数时，新的焦点窗口，将会收到 MSG_SETFOCUS 消息，旧的焦点窗口，将会收到 MSG_KILLFOCUS 消息。当一个窗口设为当前焦点窗口时，才能收到键盘产生的消息：MSG_KEY_DOWN 及 MSG_KEY_UP。

2.6.29. GetFocusWindow：获得当前焦点窗口

HWND SetFocusWindow(HWND hwnd);

头文件：

gdd.h

参数：

无。

返回值：

当前焦点窗口句柄，如果当前无焦点窗口，返回 NULL。

2.6.30. IsFocusWindow：判断指定窗口是否为当前焦点窗口

BOOL IsFocusWindow(HWND hwnd);

头文件：

gdd.h

参数：

无。

返回值：

TURE：指定的窗口是当前焦点窗口；FALSE：指定的窗口不是当前焦点窗口。

2.6.31. DefWindowProc： 系统默认的窗口消息处理函数

u32 DefWindowProc(MSG *pMsg);

头文件：

gdd.h

参数：

pMsg： 需要处理的消息。

返回值：

消息处理结果。

说明：

在窗口过程中函数中，对于用户不感兴趣的消息，需调用该函数，交由系统默认处理。

3. 绘图系统

3.1. 概述

3.1.1. 绘图上下文

用户所有的绘图操作，都需要在一个绘图上下文上进行(以下简称 DC)。DC 中记录着一些绘制参数，如各类颜色值，当前使用字体等等，用户通过设置这些参数，来改变绘图行为和效果。多个绘图上下文可同时使用，通过信号量保证了线程使用绘图上下文的安全性。

3.1.2. DrawColor，FillColor，TextColor 的作用与区别

绘图上下文中，使用了三种颜色参数，分别如下：

DrawColor： 绘制色(画笔)，用于绘制线条，空心图形，如 DrawLine，DrawCircle;

FillColor： 填充色(画刷)，用于复辅音实心图形，如 FillRect，FillCircle;

TextColor： 文字颜色，用于文字绘制时指定字体的颜色;

绘图上下文中的颜色值，用户可以使用 RGB 宏来表示，该宏的 3 个参数依次分别代表红，绿，蓝三基色，分量范围为 0~255.比如可以用以下方式来描述：

红色：RGB(255,0,0)

绿色：RGB(0,255,0)

黄色：RGB(255,255,0)

白色：RGB(255,255,255)

黑色：RGB(0,0,0)

之所以使用 3 个颜色来分别作用于不同的绘图输出，好处是在用户编程时，当绘制的图形种类多样，与字体混合绘制时，可以减少用户代码反复修改/切换颜色的情况，从用户角度来看，降低了使用上的繁琐度，增强程序的可读性。

三种颜色使用对象如图 3-1 所示：

图 3-1 三种颜色效果示意图

3.2. API 说明

3.2.1. SetRopCode： 设置当前光栅码

u32 SetRopCode(HDC hdc,u32 rop_code);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

rop_code： 新的光栅码，关于光栅码含义，请参考 gkernel.h。

返回值：

旧的光栅码。

3.2.2. GetRopCode： 获得当前光栅码

u32 GetRopCode(HDC hdc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

返回值：

当前光栅码。

3.2.3. MoveTo： 设置当前坐标位置

void MoveTo(HDC hdc,int x,int y,POINT *old_pt);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

x，y： 新的坐标位置。

old_pt： 输出旧的坐标位置，如果该参数为 NULL，则忽略该参数。

返回值： 无。

3.2.4. SetDrawColor： 设备当前画笔颜色

u32 SetDrawColor(HDC hdc,u32 color);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

color： 新的画笔颜色。

返回值：

旧的画笔颜色。

说明：

当前画笔颜色会被绘制类绘图函数使用，如 DrawLine，DrawRect…

3.2.5. GetDrawColor： 获得当前画笔颜色

u32 GetDrawColor(HDC hdc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

返回值：

当前画笔颜色。

3.2.6. SetFillColor： 设置当前填充颜色

u32 SetFillColor(HDC hdc,u32 color);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

color： 新的填充颜色。

返回值：

旧的填充颜色。

说明：

当前填充颜色会被填充类绘图函数使用，如 FillRect，FillCircle…

3.2.7. GetFillColor： 获得当前填充颜色

u32 GetFillColor(HDC hdc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

返回值：

当前填充颜色。

3.2.8. SetTextColor： 设置当前文字颜色

u32 SetTextColor(HDC hdc,u32 color);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

color： 新的文字颜色。

返回值：

旧的文字颜色。

说明：

当前文字颜色会被文字绘制类函数使用，如 TextOut，DrawText…

3.2.9. GetTextColor： 获得当前文字颜色

u32 GetTextColor(HDC hdc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

返回值：

当前文字颜色。

3.2.10. SetFont： 设置当前字体

HFONT SetFont(HDC hdc,HFONT hFont);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

hFont： 新的字体句柄。

返回值：

旧的字体句柄。

说明：

当前字体会被文字绘制类函数使用，如 TextOut，DrawText…

3.2.11. GetFont： 获得当前字体

HFONT GetFont(HDC hdc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

返回值：

当前字体句柄。

3.2.12. SetPixel： 绘制像素

void SetPixel(HDC hdc,s32 x,s32 y,u32 color);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

x，y： 像素点的坐标。

color： 颜色值。

返回值： 无。

3.2.13. DrawLine： 画线

void DrawLine(hdc hdc,s32 x0，s32 y0，s32 x1，s32 y1);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

x0，y0： 起始坐标。

x1，y1： 结束坐标，该点也会被绘制。

返回值： 无。

该函数只绘制单个像素宽度的任意直线，使用绘图上下文中的 DrawColor 作为颜色值，结束坐标点，也将会被绘制。

3.2.14. DrawLineTo： 使用当前位置画线

void DrawLineTo(HDC hdc,s32 x,s32 y);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

x，y： 结束坐标位置，该点也会被绘制。

返回值： 无。

该函数只绘制单个像素宽度的任意直线，使用绘图上下文中的 DrawColor 作为颜色值。绘制完成后，该函数会将绘图上下文中的当前坐标位置更新为本次画线的结束坐标值。

3.2.15. TextOut： 在指定位置绘制字符串

BOOL TextOut(HDC hdc, s32 x, s32 y,LPCWSTR text,s32 count);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

text： 需要绘制的字符串。

count： 需要绘制的字符数量，该参数小于 0 时，将绘制整个字符串。

x，y 两个参数没说明，是字符串的左上角坐标还是左下角坐标？

返回值：

TRUE：成功; FALSE：失败。

说明：

输出的字符串，使用绘图上下文中的 TextColor 作为颜色值，支持回车和换行符格式，当字符串超出屏幕范围时，不会自动换行，而是直接裁剪掉。

3.2.16. DrawText： 在指定矩形内绘制字符串

BOOL DrawText(HDC hdc,LPWSTR text,s32 count,const RECT *prc,u32 flag);

头文件:：

gdd.h

参数：

hdc： 绘图上下文句柄。

text： 需要绘制的字符串。

count： 需要绘制的字符数量，该参数小于 0 时，将绘制整个字符串。

prc： 字符串输出的矩形。

flag： 绘制标记，指定字符串在竖直方向位置有以下三种情形：

DT_VCENTER： 文字在矩形内垂直居中对齐；

DT_TOP： 文字在矩形内顶部对齐；

DT_BOTTOM： 文字在矩形内底部对齐。

指定字符串在水平方向位置也有以下三种情形：

DT_CENTER： 文字在矩形内水平居中对齐；

DT_LEFT： 文字在矩形内左对齐；

DT_RIGHT： 文字在矩形内右对齐；

其他情形有：

DT_BORDER： 绘制矩形边框。

DT_BKGND： 绘制矩形背景。

绘制标记 flag 从竖直方向及水平方向位置三种情形中各选取其一与其他情形中两个任意组合。

返回值：

TRUE：成功; FALSE：失败。

说明：

输出的字符串，使用绘图上下文中的 TextColor 作为颜色值，支持回车和换行符格式。;绘制矩形边框，使用绘图上下文中的 DrawColor 作为颜色值;填充矩形背景，使用绘图上下文中的 FillColor 作为颜色值。

3.2.17. DrawRect： 绘制矩形

void DrawRect(HDC hdc,const RECT *prc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

prc： 需要绘制的矩形参数。

返回值： 无

说明：

该函数使用绘图上下文中的 DrawColor 作为颜色值，绘制一个空心矩形。

3.2.18. FillRect： 填充矩形

void FillRect(HDC hdc,const RECT *prc);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

prc： 需要填充的矩形参数。

返回值： 无

说明：

该函数使用绘图上下文中的 FillColor 作为颜色值，填充一个实心矩形。

3.2.19. GradientFillRect： 渐变填充矩形

void GradientFillRect(HDC hdc,const RECT *prc,u32 color1,u32 color2,u32 mode);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

prc： 需要绘制的矩形参数。

color1： 起始颜色值。

color2： 结束颜色值。

mode： 填充模式，可以是以下值之一：

CN_FILLRECT_MODE_H： 水平填充，Color0 表示左边颜色，Color1 右边；

CN_FILLRECT_MODE_V： 垂直填充，Color0 表示上边颜色，Color1 下边；

CN_FILLRECT_MODE_SP： 倾斜填充，Color0 表示左上角颜色，Color1 右下角；

CN_FILLRECT_MODE_SN： 倾斜填充，Color0 表示右上角颜色，Color1 左下角。

返回值： 无。

3.2.20. DrawPolyLine： 绘制折线

void DrawPolyLine(HDC hdc,const POINT *pt,int count);

头文件：

gdd.h

参数：

hdc： 绘图上下文句柄。

pt： 需要绘制的曲线的坐标点。

count： 需要绘制的曲线的坐标点数量。

返回值： 无。

说明：

该函数使用绘图上下文中的 DrawColor 作为颜色值。

3.2.21. DrawGroupBox： 绘制组合框

void DrawGroupBox(HDC hdc,const RECT *prc,const WCHAR *text);

头文件:：

gdd.h

参数：

hdc： 绘图上下文句柄。

prc： 需要绘制的组合框的矩形参数。

text： 组合框文字内容。

返回值： 无。

说明：

组合框是将文本框和列表框的功能融合在一起的一种控件。用户既可以在文本框中输入，也可以从列表框中选择一个列表项来完成输入。

该函数使用设备上下文中的 TextColor 作为组合框的文字颜色值；使用 DrawColor 作为组合框边框颜色值。 组合框可用于对用户绘制的内容进行分组标识，这使得界面在视觉效果上更加整洁美观;界面内容也更加直观明了。

3.2.22. AlphaBlendColor： 计算两个颜色按 Alpha 混合后的颜色值

u32 AlphaBlendColor(u32 bk_c,u32 fr_c,u8 alpha);

头文件：

gdd.h

参数：

bk_c： 背景色(XRGB8888 格式)。

fr_c： 前景色(XRGB8888 格式)。

apha： Alpha 分量值(0~255 范围)。

返回值：

混合后的颜色值(XRGB8888 格式)。

说明：

Alpha 颜色混合公式为： 显示颜色=源颜色×alpha/255+背景颜色×(255-alpha)/255;

3.2.23. UpdateDisplay： 立即更新显示到屏幕

void UpdateDisplay(void);

头文件：

gdd.h

参数： 无。

返回值： 无。

4. 消息系统

4.1. 概述

消息系统在整个系统中的关系层次如所示：

图 4-1 消息系统在 GDD 中层次示意图

每个主窗口均会有一个消息队列，在调用 CreateWindow 函数创建主窗口时内部会创建一个该主窗口所属的消息队列，消息队列长度为 32。子窗口通过主窗口的消息队列获取消息。

4.2. 同步消息与异步消息

消息发送方式分为同步发送与异步发送两种，系统提供两个 API 函数：SendMessage，PostMessage，分别用于两种不同的消息发送方式，它们的区别如下：

SendMessage： 用于同步消息发送，消息直接发送到目标窗口的窗口过程函数中，直到该条消息被窗口过程函数处理完成后，该函数才会返回。

PostMessage： 用于异步消息发送，消息发送到目标窗口所属的消息队列中后，不等待处理完成，便立即返回。

4.3. 消息队列及消息循环与窗口的关系

因为异步消息不需要立即处理，所有主窗口和桌面窗口，都有自己的消息队列，用于缓存异步消息。对子窗口发送异步消息时，实际是将消息发送到了它所属的主窗口消息队列中。在窗口入口函数中，调用 GetMessage 函数从主窗口消息队列中取出一条消息，若成功取出一条消息，则调用 DispatchMessage 函数派发该消息，所谓派发消息其实就是间接调用主窗口的窗口过程函数处理该消息。在窗口入口函数中获取消息及派发消息实例代码如下：

```
while(GetMessage(&msg,hwnd))     //从指定的主窗口消息队列中取出一条消息
    {
        DispatchMessage(&msg);   //派发该消息
    }
```

4.4. 窗口过程函数及消息数据结构

所有的成功发送出来的消息，无论是同步还异步方式发送，最终都会到达窗口过程函数中，窗口过程函数是一个由用户指定的回调函数，这里也是窗口程序的主体部分，用户在这里可以对接收到的各类消息进行响应处理，对于某些不需要处理的消息必须调用 DefWindowProc 函数，将该消息交给系统默认处理。窗口过程函数会传入当前接收到的消息数据结构。该数据结构如下：

```
typedef struct  tagMSG
{
```

```
    HWND hwnd;
    u32 Code;
    u32 Param1;
    u32 Param2;
    const void* ExData;
}MSG;
```

该结构各成员作用如下：

hwnd： 该条消息所属的目标窗口句柄，窗口过程函数必然是所属该窗口；

Code： 消息代码值；

Param1： 消息参数 1，不同消息代码，该参数意义不同；

Param1： 消息参数 2，不同消息代码，该参数意义不同；

ExData： 消息扩展数据，不同消息代码，该参数意义不同。

上面已提及窗口过程函数主要功能就是根据不同消息做相应的处理，遇到不需要特别处理的消息交给系统做默认处理，一个典型的窗口过程函数实例代码如下：

```
static   u32 win_proc(MSG *pMsg)
{
    HWND hwnd;
    HDC hdc;
    RECT rc,rc0;
    hwnd =pMsg->hwnd;
    switch(pMsg->Code)
    {
        case    MSG_CREATE:
                GetClientRect(hwnd,&rc0);
                CreateWindow(BUTTON,L"关闭
",WS_CHILD|BS_NORMAL|WS_BORDER|WS_VISIBLE,RectW(&rc0)-64,RectH(&rc0)-28,60,24,hwnd,IDB
_CLOSE,NULL);
                GDD_CreateTimer(hwnd,1,5000,TMR_START);
                break;
        case    MSG_TIMER:
              PostMessage(hwnd,MSG_CLOSE,0,0);
                break;
        case    MSG_NOTIFY:
                {
                    u16 event,id;
                    event =HI16(pMsg->Param1);
                    id =LO16(pMsg->Param1);
                    if(event==BTN_UP && id==IDB_CLOSE)
                    {
                        PostMessage(hwnd,MSG_CLOSE,0,0);
                    }
                }
                break;
        case    MSG_PAINT:
                {
                    hdc =BeginPaint(hwnd);
                    GetClientRect(hwnd,&rc0);
                    SetFillColor(hdc,RGB(170,160,135));
                    FillRect(hdc,&rc0);
                    SetTextColor(hdc,RGB(0,0,0));
                    TextOut(hdc,2,4,wstr,-1);
                    EndPaint(hwnd,hdc);
                }
                break;
        case    MSG_CLOSE:
```

```
                DestroyWindow(hwnd);
                return 1;
        case    MSG_DESTROY:
                PostQuitMessage(hwnd,0);
                return 1;
        default:
                return  DefWindowProc(pMsg);
    }
    return  0;
}
```

该窗口过程函数包含了对创建窗口消息(MSG_CREATE)、定时器超时消息(MSG_TIMER)、控件状态变化告知信息(MSG_NOTIFY)、绘图消息(MSG_PAINT)、关闭窗口消息(MSG_CLOSE)及销毁窗口消息等消息对应的处理方法。除了关闭窗口消息及销毁窗口消息处理方法相同，其他消息应根据应用程序实际需求编写。上述实例只是定义了部分消息处理方法，GDD 还定义了其他窗口消息类型具体参见 gdd.h。应用程序可根据实际需求对各消息类型做相应的处理。

4.5. 消息代码及参数说明

系统为用户定义了一系列通用消息代码，以下是这些消息的详细说明。

4.5.1. MSG_CREATE： 窗口创建消息

参数：

Param1： 由 CreateWindow 的 pdata 传入。

Param2： 忽略。

说明：

该消息在窗口创建时，由 CreateWindow 函数产生，用户可以在该消息响应中作一些初始化的工作。

4.5.2. MSG_ERASEBKGND： 窗口背景重绘消息

参数：

Param1： 绘图上下文句柄。

Param2： 忽略。

说明：

在窗口重绘之前，如果有对窗口背景进行重绘的请求，那么 BeginPaint 函数内部将会先以同步方式发送该消息，用于先对窗口进行背景进行重绘；背景重绘请求产生的条件为：调用了 InvalidateWindow 函数，第二个参数 bErase 为 TURE；具体参考：InvalidateWindow： 设置窗口为无效状态

4.5.3. MSG_NCPAINT： 窗口非客户区绘制消息

参数：

Param1： 忽略。

Param2： 忽略。

说明：

当窗口非客户区需要重新绘制时（比如窗口由不可见状态变为可见状态或用户主动发送了 MSG_NCPAINT 消息），当窗口过程收到该消息时，指示窗口非客户区需要重新绘制，如果用户不需要自己绘制窗口非客户，可以调用 DefWindowProc 函数，交由系统默认处理。

4.5.4. MSG_PAINT： 窗口客户区绘制消息

参数：

Param1： 忽略。

Param2： 忽略。

说明：

当窗口客户区需要重新绘制时（比如窗口由不可见状态变为可见状态或用户主动发送了 MSG_PAINT 消息）当窗口过程收到该消息时，指示窗口客户区需要重新绘制。

4.5.5. MSG_TIMER： 定时器超时消息

参数：

Param1： 定时器 ID。

Param2： 忽略。

说明：

当一个定时器定时时间到来时，便会发送该消息到定时器所属的窗口，如果该定时器产生的消息未被窗口处理完成，那么该定时器将不会再重复产生 MSG_TIMER 消息。

4.5.6. MSG_CLOSE： 窗口请求关闭消息

参数：

Param1： 忽略。

Param2： 忽略。

说明：

如果接收该消息，表示窗口请求关闭，用户在这里可以按实际情况处理是否要真正关闭窗口，如果用户需要继续关闭窗口，则需调用 DestroyWindow 函数，否则直接返回。

4.5.7. MSG_DESTROY： 窗口销毁消息

参数：

Param1： 忽略。

Param2： 忽略。

说明：

当用户调用了 DestroyWindow 函数时，会产生该消息，表示窗口需要被销毁，用户可以在这里做一些资源善后工作，然后必需调用 PostQuitMessage 函数来指示窗口需要退出消息循环。

4.5.8. MSG_NOTIFY： 控件通知消息

参数：

Param1： 低 16 位：控件 ID;高 16 位：控件通知码。

Param2： 控件窗口句柄。

说明：

该消息由控件向所属的父窗口发送的消息，用于通知父窗口，控件本身发生了状态变化，通知码用于描述控件状态及其变化等，例如：

控件名称	控件码	描述
BUTTON	BTN_DOWN	按钮被按下
	BTN_UP	按钮弹起
CHECKBOX	CBN_SELECTED	复选框为选中状态
	CBN_UNSELECTED	复选框为未选中状态
LISTBOX	LBN_SELCHANGE	列表框当前选择项被改变

4.5.9. MSG_LBUTTON_DOWN： 客户区鼠标左键点击消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区按下左键时，该窗口会自动收到该消息，如果是触摸屏设备的笔针点击屏幕的动作，也会产生该消息。

4.5.10. MSG_LBUTTON_UP： 客户区鼠标左键弹起消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区松开左键时，该窗口会自动收到该消息，如果是触摸屏设备的笔针抬起的动作，也会产生该消息。

4.5.11. MSG_SETFOCUS： 窗口获得焦点

参数：

Param1： 忽略。

Param2： 忽略。

说明：

当调用 SetFocusWindow 函数时，新设置的焦点窗口，将会收到 MSG_SETFOCUS 消息，当窗口收到该消息时，说明该窗口已经被设置为当前焦点窗口。对于键盘产生的消息，将会自动发送到当前焦点窗口。

4.5.12. MSG_KILLFOCUS： 窗口失去焦点

参数：

Param1： 忽略。

Param2： 忽略。

说明：

当调用 SetFocusWindow 函数时，当前旧的焦点窗口，将会收到 MSG_KILLFOCUS 消息，当窗口收到该消息时，说明该窗口失去焦点。当一个窗口失去焦点后，它将不能接收到键盘产生的消息。

4.5.13. MSG_RBUTTON_DOWN： 客户区鼠标右键点击消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区按下右键时，该窗口会自动收到该消息。

4.5.14. MSG_LBUTTON_UP： 客户区鼠标右键弹起消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区松开右键时，该窗口会自动收到该消息。

4.5.15. MSG_MBUTTON_DOWN： 客户区鼠标中键点击消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区按下中键时，该窗口会自动收到该消息。

4.5.16. MSG_MBUTTON_UP： 客户区鼠标中键弹起消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区松开中键时，该窗口会自动收到该消息。

4.5.17. MSG_MOUSE_MOVE： 客户区鼠标移动消息

参数：

Param1： 鼠标按键状态。可能是以下值组合：

MK_LBUTTON： 鼠标左键为按下状态。

MK_MBUTTON： 鼠标中键为按下状态。

MK_RBUTTON： 鼠标右键为按下状态。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用客户区坐标表示。

说明：

当鼠标在窗口的客户区移动时，该窗口会自动收到该消息。

4.5.18. MSG_NCLBUTTON_DOWN： 非客户区鼠标左键点击消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区按下左键时，该窗口会自动收到该消息，如果是触摸屏设备的笔针点击屏幕的动作，也会产生该消息。

4.5.19. MSG_NCLBUTTON_UP： 非客户区鼠标左键弹起消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区松开左键时，该窗口会自动收到该消息，如果是触摸屏设备的笔针抬起的动作，也会产生该消息。

4.5.20. MSG_NCRBUTTON_DOWN： 非客户区鼠标右键点击消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区按下右键时，该窗口会自动收到该消息。

4.5.21. MSG_NCRBUTTON_UP： 非客户区鼠标右键弹起消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区松开右键时，该窗口会自动收到该消息。

4.5.22. MSG_NCMBUTTON_DOWN： 非客户区鼠标中键点击消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区按下中键时，该窗口会自动收到该消息。

4.5.23. MSG_NCMBUTTON_UP： 非客户区鼠标中键弹起消息

参数：

Param1： 忽略。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区松开中键时，该窗口会自动收到该消息。

4.5.24. MSG_NCMOUSE_MOVE： 非客户区鼠标移动消息

参数：

Param1： 鼠标按键状态。可能是以下值组合：

MK_LBUTTON： 鼠标左键为按下状态。

MK_MBUTTON： 鼠标中键为按下状态。

MK_RBUTTON： 鼠标右键为按下状态。

Param2： 低 16 位： X 坐标值; 高 16 位：Y 坐标值; 使用屏幕坐标表示。

说明：

当鼠标在窗口的非客户区移动时，该窗口会自动收到该消息。

4.5.25. MSG_KEY_DOWN： 按键按下消息

参数：

Param1： 低 16 位：按键值；高 16 位：保留。

Param2： 该按键消息产生的时间，单位为毫秒。

说明：

当键盘有按键按下时，会产生一次该消息。

4.5.26. MSG_KEY_UP： 按键弹起消息

参数：

Param1： 低 16 位：按键值；高 16 位：保留。

Param2： 该按键消息产生的时间，单位为毫秒。

说明：

当键盘有按键弹起时，会产生一次该消息。

4.5.27. PBM_SETDATA：进度条设置数据

参数：

Param1： 进度条数据结构指针。

Param2： 忽略。

说明：

无。

4.5.28. PBM_GETDATA：进度条获得数据

参数：

Param1：进度条数据结构指针。

Param2： 忽略。

说明：

无。

4.5.29. PBM_SETRANGE：进度条设置量程值

参数：

Param1：量程值。

Param2：忽略。

说明：

无。

4.5.30. PBM_GETRANGE： 进度条获得量程值

参数：

Param1：忽略。

Param2：忽略。

返回：

量程值。

说明：

无。

4.5.31. PBM_SETPOS： 进度条设置当前位置

参数：

Param1：当前位置。

Param2：忽略。

说明：

进度条当前位置值，是相对于量程值。

4.5.32. PBM_GETPOS： 进度条获得当前位置

参数：

Param1：忽略。

Param2：忽略。

返回：

当前位置。

说明：

进度条当前位置值，是相对于量程值。

4.5.33. LBM_ADDSTRING： 列表框增加一个字符项

参数：

Param1：项目索引值。

Param2：字符指针。

返回：

实际的项目索引值。

4.5.34. LBM_DELSTRING： 列表框删除一个字符项

参数：

Param1：项目索引值。

Param2：忽略。

返回：

忽略。

4.5.35. LBM_SETCURSEL： 列表框设置当前选择项

参数：

Param1：项目索引值。

Param2：忽略。

返回：

忽略。

说明：

当前选择项是指当前被选中的项目。

4.5.36. LBM_GETCURSEL： 列表框获得当前选择项

参数：

Param1：忽略。

Param2：忽略。

返回：

当前选择项。

4.5.37. LBM_SETTOPINDEX： 列表框设置顶部首个可见项

参数：

Param1：项目索引值。

Param2：忽略。

返回：

忽略。

说明：

列表框显示时，是从首个可见项开始，在这前面的项目，是不会显示出来的。

4.5.38. LBM_GETTOPINDEX： 列表框获得顶部首个可见项

参数：

Param1：忽略。

Param2：忽略。

返回：

顶部首个可见项索引值。

说明：

列表框显示时，是从首个可见项开始，在这前面的项目，是不会显示出来的。

4.5.39. LBM_GETCOUNT： 列表框获得当前项目数量

参数：

Param1：忽略。

Param2：忽略。

返回：

当前项目数量值。

4.5.40. LBM_RESETCONTENT：列表框的删除所有项目

参数：

Param1：忽略。

Param2：忽略。

返回：

忽略。

4.5.41. LBM_GETTEXTLEN：列表框获得指定项目的字符字节长度

参数：

Param1：项目索引。

Param2：忽略。

返回：

指定项目的字符字节长度。

4.5.42. LBM_GETTEXT：列表框获得指定项目的字符内容

参数：

Param1：项目索引。

Param2：输出的字符内容缓冲区。

返回：

忽略。

4.5.43. LBM_SETITEMHEIGHT：列表框设置指定项目的高度

参数：

Param1：项目索引。

Param2：高度值（像素单位）。

返回：

忽略。

4.5.44. LBM_GETITEMHEIGHT：列表框获得指定项目的高度

参数：

Param1：项目索引。

Param2：忽略。

返回：

指定项目的高度值（像素单位）。

4.5.45. LBM_SETITEMDATA：列表框设置指定项目的数据值

参数：

Param1：项目索引。

Param2：数据值；这个数据值并不会列表框内部使用，作为用户自定义使用。

返回：

忽略。

4.5.46. LBM_GETITEMDATA：列表框获得指定项目的数据值

参数：

Param1：项目索引。

Param2：忽略。

返回：

指定项目的数据值。

4.6. API 说明

4.6.1. DispatchMessage： 派发消息

u32 DispatchMessage(MSG *pMsg);

头文件：

gdd.h

参数：

pMsg： 需要派发的消息。

返回值：

消息处理结果。

说明：

该函数只在主窗口消息循环中使用。

4.6.2. SendMessage： 以同步方式发送消息

u32 SendMessage(HWND hwnd,u32 msg,u32 param1,u32 param2);

头文件：：

gdd.h

参数：

hwnd： 消息发送的目的窗口句柄。

msg： 消息代码。

param1： 消息参数 1，不同消息，该参数意义不同。

param2： 消息参数 2，不同消息，该参数意义不同。

说明：

该函数以同步方式发送消息，支持跨线程发送消息到指定窗口，直到该消息被窗口处理完成后，该函数才会返回。

4.6.3. PostMessage： 以异步方式发送消息

BOOL PostMessage(HWND hwnd,u32 msg,u32 param1,u32 param2);

头文件：：

gdd.h

参数：

hwnd： 消息发送的目的窗口句柄。

msg： 消息代码。

param1： 消息参数 1，不同消息，该参数意义不同。

param2： 消息参数 2，不同消息，该参数意义不同。

返回值：

TRUE： 成功; FALSE： 失败。

说明：

该函数以异步该方式发送消息，支持跨线程发送消息到指定窗口，用该函数发送一条消息后，便立即返回，不会等待窗口是否处理完成。

4.6.4. PostQuitMessage： 异步方式发送退出消息

BOOL PostQuitMessage(HWND hwnd,u32 exit_code);

头文件：

gdd.h

参数：

hwnd： 窗口句柄。

exit_code： 用户自定义的窗口退出码。

返回值：

TRUE： 成功; FALSE： 失败。

说明：

该函数会产生一条 MSG_QUIT 消息，当主窗口消息循环接收到 MSG_QUIT 消息时，便会自动销毁主窗口，并退出消息循环，这意味着一个主窗口过程的结束。 该函数只在主窗口过程函数的 MSG_DESTROY 消息中使用，在其它地方使用，将可能产生不可预知的后果;详细内容可参考窗口的关闭、销毁、退出过程

4.6.5. PeekMessage： 以非阻塞方式获取一条消息

BOOL PeekMessage(MSG *pMsg, HWND hwnd);

头文件：

gdd.h

参数：

pMsg： 存放一条所获得的消息缓冲区。

hwnd： 主窗口句柄。

返回值：

TRUE： 成功获得了一条消息; FALSE： 没有获得消息。

说明：

从消息队列头取出一条消息，该消息将会从消息队列中删除，该函数只在主窗口入口函数中获取消息时使用。

4.6.6. GetMessage： 以阻塞方式获取一条消息

BOOL GetMessage(MSG *pMsg,HWND hwnd);

头文件：

gdd.h

参数：

pMsg： 存放一条所获得的消息缓冲区。

hwnd： 主窗口句柄。

返回值：

TRUE： 成功获得了一条消息; FALSE： 获得了一条 MSG_QUIT 消息。

说明：

该函数只在主窗口消息循环中使用。

5. 定时器

5.1. 概述

GDD 的定时器是以系统 TICK 为基准进行计时，是属于一种软件模拟性质的虚拟定时器，这些特性使得定时器不会有很高的精度和准确度，适用于一些对时间要求不是很严格的场合。定时器是作为窗口的资源形式存在，它的服务对象是直接面向窗口。每个窗口都可以由用户创建最多 65536 个独立的定时器，在窗口运行过程中，用户可以动态去创建和删除定时器，也可以动态去修改已有的定时器运行参数。当一个定时器超时发生后，定时器所属的窗口过程将会收到一条 MSG_TIMER 消息，MSG_TIMER 参数会附带产生该消息的定时器 ID，以便用户对多个定时器产生的超时消息加以识别。用户需要注意的是： MSG_TIMER 消息是优先级最低的一类，只有当窗口所属的消息队列中没有其它消息时，系统才会将 MSG_TIMER 派发到它的目标窗口过程函数。

5.2. API 说明

5.2.1. GDD_CreateTimer： 创建定时器

TIMER* GDD_CreateTimer(HWND hwnd,u16 Id,u32 IntervalMS,u16 Flag);

头文件:：

gdd.h

参数：

hwnd： 定时器所属窗口。

Id： 定时器 ID，由用户指定一个 0~65535 之间的数，所以理论上，每个窗口都可以创建 65536 个定时器，用户在创建定时器时，必须保证同一窗口下的定时器 ID 是唯一的，否则该函数会创建失败。

IntervalMS： 定时间隔，单位：毫秒。

Flag： 定时器标志属性，可以是以下值组合：

TMR_START： 创建成功后，便立即启动定时器。

TMR_SINGLE： 如果指定了该标记，则为单次定时模式，否则为循环定时模式。

返回值：

成功则返回新创建的定时器对象指针;失败则返回 NULL。

说明：

当定时器定时时间到来时，将为所属的窗口产生一条异步处理的定时器超时消息 MSG_TIMER，在所有的消息类型中，MSG_TIMER 属于优先级最低的，只有当所有其它消息处理完成后，系统才会真正将 MSG_TIMER 消息发送到窗口过程函数，交由用户处理响应，当因系统繁忙导致未能及时处理 MSG_TIMER 消息时，同一定时器将不会重复产生 MSG_TIMER 消息，以避免这些没有意义冗余操作影响系统响应性能。

5.2.2. GDD_FindTimer： 查找窗口指定 ID 的定时器

TIMER* GDD_FindTimer(HWND hwnd,u16 Id);

头文件：

gdd.h

参数：

hwnd： 定时器所属窗口。

Id： 需要查找的定时器 Id。

返回值：

查找到的定时器对象指针;如果窗口不存在指定 Id 的定时器，则返回 NULL。

5.2.3. GDD_ResetTimer： 重新设置定时器参数

BOOL GDD_ResetTimer(TIMER *ptmr,u32 IntervalMS,u32 Flag);

头文件：

gdd.h

参数：

ptmr： 需要重新设置的定时器对象指针。

IntervalMS： 定时间隔，单位：毫秒。

Flag： 定时器标志属性，可以是以下值组合：

TMR_START： 创建成功后，便立即启动定时器。

TMR_SINGLE： 如果指定了该标记，则为单次定时模式，否则为循环定时模式。

返回值：

TRUE：成功; FALSE：失败。

说明：

当指定的定时器被重置时，如果计时器正在运行中，那么它的计时时间将会强制重新从 0 开始计数，如果之前产生了定时器消息未被及时处理，也将会被移除。

5.2.4. GDD_DeleteTimer： 删除定时器

BOOL GDD_DeleteTimer(TIMER *ptmr);

头文件：

gdd.h

参数：

ptmr： 需要删除的定时器对象指针。

返回值：

TRUE：成功; FLASE：失败。

说明：

当一个定时器不需要再使用时，用户可以使用该函数来删除指定的定时器；当然，定时器属于窗口的私有资源对象，即便用户不调用该函数来删除那些不再使用的定时器，那么，在窗口退出(销毁)前，系统也将会自动删除这些定时器，以避免因资源泄漏而产生系统崩溃。另外，定时器已产生但还未及时处理的消息在删除定时器时也会将该消息从消息队列中删除。

6. 矩形区域运算

说明下矩形的位置基准。

6.1. API 说明

6.1.1. RectW： 获得矩形宽度

s32 RectW(const RECT *prc);

头文件：

gdd.h

参数：

prc： 矩形对象指针。

返回值：

矩形宽度。

6.1.2. RectH： 获得矩形高度

int RectH(const RECT *prc);

头文件：

gdd.h

参数：

prc： 矩形对象指针。

返回值：

矩形高度。

6.1.3. SetRect： 设置矩形参数

void SetRect(RECT *prc,s32 x, s32 y, s32 w, s32 h);

头文件:：

gdd.h

参数：

prc： 需要设置的矩形对象指针。

x，y： 矩形起始位置(左上角坐标)。

w，h： 矩形宽度和高度。

返回值： 无。

6.1.4. SetRectEmpty： 设置一个矩形为空矩形

void SetRectEmpty(RECT *prc);

头文件：

gdd.h

参数：

prc： 需要设置的矩形对象指针。

返回值： 无。

说明：

当一个矩形的宽度或高度等于 0 时， 即表示这个矩形为空矩形。

6.1.5. IsRectEmpty： 判断一个矩形是否为空矩形

BOOL IsRectEmpty(const RECT *prc);

头文件：

gdd.h

参数：

prc： 需要判断的矩形对象指针。

返回值：

TRUE： 该矩形为空矩形; FALSE： 该矩形为非空矩形。

6.1.6. CopyRect： 复制矩形参数

BOOL CopyRect(RECT *dst,const RECT *src);

头文件：

gdd.h

参数：

dst： 目标矩形对象。

src： 源矩形对象。

返回值：

TRUE： 成功; FALSE：失败。

说明：

该函数将源矩形对象的参数复制到目标矩形对象中。

6.1.7. OffsetRect： 偏移矩形位置

BOOL OffsetRect(RECT *prc,s32 dx,s32 dy);

头文件：

gdd.h

参数：

prc： 需要偏移的矩形对象指针。

dx： 水平方向的偏移值，当为负值时，向左偏移，否则向右偏移。

dy： 垂直方向的偏移值，当为负值时，向上偏移，否则向下偏移。

返回值： TRUE： 成功; FALSE：失败。

6.1.8. InflateRect： 扩大或缩小矩形

BOOL InflateRect(RECT *prc, s32 dx, s32 dy);

头文件：

gdd.h

参数：

prc： 矩形对象指针。

dx： 矩形左边和右边，分别扩大或缩小的值，当该参数为负值时，为减小，否则为增大。

dy： 矩形上边和下边，分别扩大或缩小的值，当该参数为负值时，为减小，否则为增大。

返回值：

TRUE：成功; FALSE：失败.

6.1.9. InflateRectEx： 扩大或缩小矩形

BOOL InflateRectEx(RECT *prc, s32 l, s32 t, s32 r, s32 b);

头文件：

gdd.h

参数：

prc： 矩形对象指针。

l： 矩形左边扩大或缩小的值，当该参数为负值时，为减小，否则为增大。

t： 矩形上边扩大或缩小的值，当该参数为负值时，为减小，否则为增大。

r： 矩形右边扩大或缩小的值，当该参数为负值时，为减小，否则为增大。

b： 矩形下边扩大或缩小的值，当该参数为负值时，为减小，否则为增大。

返回值：

TRUE：成功; FALSE：失败。

6.1.10. PtInRect： 判断坐标点是否在矩形范围内

BOOL PtInRect(const RECT *prc,const POINT *pt);

头文件：

gdd.h

参数：

prc： 矩形对象指针。

pt： 坐标点对象指针。

返回值：

TRUE：坐标点在矩形范围内；FALSE：坐标点不在矩形范围内。

7. 输入

7.1. API 说明

7.1.1. GetMouseKeyState： 获得鼠标按键状态

u8 GetMouseKeyState(void);

头文件：

gdd.h

参数： 无。

返回值：

鼠标按键状态，可以是以下多个按键组合值：

MK_LBUTTON： 左键被按下。

MK_MBUTTON： 中键被按下。

MK_RBUTTON： 右键被按下。

7.1.2. MouseInput： 设置鼠标输入事件

void MouseInput(s32 x,s32 y,u32 key_state);

头文件：

gdd.h

参数：

x，y： 鼠标坐标位置。

key_satate： 鼠标按键状态，可以是以下值组合：

MK_LBUTTON： 左键被按下。

MK_MBUTTON： 中键被按下。

MK_RBUTTON： 右键被按下。

返回值： 无。

说明：

该函数可以用来模拟一次鼠标的动作，会产生鼠标输入事件，并自动发送消息到符合条件的窗口中。

7.1.3. KeyboardInput： 设置键盘输入事件

void KeyboardInput(u16 key_val,EN_GDD_KEY_EVENT key_event);

头文件：

gdd.h

参数：

key_val：按键值。

key_event： 按键事件类型：

KEY_EVENT_DOWN： 按键按下。

KEY_EVENT_UP： 按键弹起。

返回值： 无。

说明：

该函数可以用来模拟一次键盘的动作，会产生键盘输入事件，并自动发送键盘消息到当前焦点窗口中。