# 一、数据结构

## CreateMainWindow函数参数：PMAINWINCREATE pCreateInfo

结构体MAINWINCREATE 定义了被创建的窗口的位置、标题、类型等基本参数。实际上包含了创建窗口的UI风格和窗口处理函数两方面的内容。

PMAINWINCREATE为指向该结构体的指针。

typedef struct \_MAINWINCREATE

{

DWORD dwStyle; //主窗口的类型

DWORD dwExStyle; //主窗口的扩展类型

const char\* spCaption; //主窗口的标题

HMENU hMenu; //主窗口菜单句柄

HCURSOR hCursor; //主窗口光标句柄

HICON hIcon; //主窗口图标句柄

HWND hHosting; //主窗口的托管窗口The hosting main window

int (\*MainWindowProc)(HWND, int, WPARAM, LPARAM); // 窗口回调函数

int lx, ty, rx, by; //主窗口在屏幕坐标中的位置

int iBkColor; //主窗口颜色的像素值

DWORD dwAddData; //私有数据The first private data associated with the main window

DWORD dwReserved; //没有用到

}MAINWINCREATE;

typedef MAINWINCREATE\* PMAINWINCREATE;

## 2、MAINWIN结构体：主窗口的详细信息由该结构体给出

typedef struct \_MAINWIN

{

/\*

\* These fields are similiar with CONTROL struct.

\*/

short DataType; // the data type.

short WinType; // the window type.

int left, top; // the position and size of main window.

int right, bottom;

int cl, ct; // the position and size of client area.

int cr, cb;

DWORD dwStyle; // the styles of main window.

DWORD dwExStyle; // the extended styles of main window.

int iBkColor; // the background color.

HMENU hMenu; // handle of menu.

HACCEL hAccel; // handle of accelerator table.

HCURSOR hCursor; // handle of cursor.

HICON hIcon; // handle of icon.

HMENU hSysMenu; // handle of system menu.

PLOGFONT pLogFont; // pointer to logical font.

HDC privCDC; // the private client DC.

INVRGN InvRgn; // the invalid region of this main window.

PGCRINFO pGCRInfo; // pointer to global clip region info struct.

PZORDERNODE pZOrderNode;

PCARETINFO pCaretInfo;// pointer to system caret info struct.

DWORD dwAddData; // the additional data.

DWORD dwAddData2; // the second addtional data.

int (\*MainWindowProc)(HWND, int, WPARAM, LPARAM);

// the address of main window procedure.

char\* spCaption; // the caption of main window.

int id; // the identifier of main window.

SCROLLBARINFO vscroll;// the vertical scroll bar information.

SCROLLBARINFO hscroll;// the horizital scroll bar information.

struct \_MAINWIN\* pMainWin;

// the main window that contains this window.

// for main window, always be itself.

HWND hParent; // the parent of this window.

// for main window, always be HWND\_DESKTOP.

/\*

\* Child windows.

\*/

HWND hFirstChild; // the handle of first child window.

HWND hActiveChild; // the currently active child window.

HWND hOldUnderPointer; // the old child window under pointer.

HWND hPrimitive; // the premitive child of mouse event.

NOTIFPROC NotifProc; // the notification callback procedure.

/\*

\* window element data.

\*/

struct \_wnd\_element\_data\* wed;

/\*

\* Main Window hosting.

\* The following members are only implemented for main window.

\*/

struct \_MAINWIN\* pHosting; // the hosting main window.

struct \_MAINWIN\* pFirstHosted;// the first hosted main window.

struct \_MAINWIN\* pNextHosted;// the next hosted main window.

PMSGQUEUE pMessages;

// the message queue.

GCRINFO GCRInfo;

// the global clip region info struct.

// put here to avoid invoking malloc function.

} MAINWIN;

## 3、MSGQUEUE消息队列

**struct** \_MSGQUEUE

{

DWORD dwState; // message queue states

PQMSG pFirstNotifyMsg; // head of the notify message queue

PQMSG pLastNotifyMsg; // tail of the notify message queue

IDLEHANDLER OnIdle; // Idle handler

MSG\* msg; /\* post message buffer \*/

**int** len; /\* buffer len \*/

**int** readpos, writepos; /\* positions for reading and writing \*/

**int** FirstTimerSlot; /\* the first timer slot to be checked \*/

DWORD TimerMask; /\* timer slots mask \*/

**int** loop\_depth; /\* message loop depth, for dialog boxes. \*/

};

# 二、函数流程

## 1、init\_desktop\_win

static void init\_desktop\_win (void)  
{  
    static MAINWIN sg\_desktop\_win; // 静态量  
    PMAINWIN pDesktopWin;  
    pDesktopWin = &sg\_desktop\_win; // 作为默认的Desktop  
    pDesktopWin->pMessages         = \_\_mg\_dsk\_msg\_queue; // 自己的消息队列,其他窗口发送的SendMessage将消息压入该队列[luther.gliethttp]  
    pDesktopWin->MainWindowProc    = DesktopWinProc; // 桌面默认回调处理函数  
    pDesktopWin->DataType          = TYPE\_HWND;  
    pDesktopWin->WinType           = TYPE\_ROOTWIN;  
    pDesktopWin->pLogFont          = NULL;  
    pDesktopWin->spCaption         = "THE DESKTOP WINDOW";  
    pDesktopWin->pGCRInfo          = &sg\_ScrGCRInfo;  
    pDesktopWin->pMainWin          = pDesktopWin;  
    \_\_mg\_hwnd\_desktop = (HWND)pDesktopWin; // 登记到desktop的全局量中  
    // window.h  
    // #define HWND\_DESKTOP        \_\_mg\_hwnd\_desktop  
    \_\_mg\_dsk\_win  = pDesktopWin;  
}

## 2、main函数

**int** **main**(**void**) {

**puts**("!!!Hello World!!!"); /\* prints !!!Hello World!!! \*/

**int** i=0;

**if** (InitGUI () != 0) {

**return** 1;

}

// MyMiniGUIMain();

MSG Msg;

HWND hMainWnd;

MAINWINCREATE CreateInfo;

//设置主窗口风格，把主窗口设置为初始可见，具有边框和标题栏

CreateInfo.dwStyle = WS\_VISIBLE | WS\_BORDER | WS\_CAPTION;

CreateInfo.dwExStyle = WS\_EX\_NONE; //设置主窗口的扩展风格，该窗口没有扩展风格

CreateInfo.spCaption = "hello"; //设置主窗口的标题为hello

CreateInfo.hMenu = 0; //设置主窗口的主菜单，该窗口没有主菜单

CreateInfo.hCursor = GetSystemCursor(0); //设置主窗口的光标为系统缺省光标

CreateInfo.hIcon = 0; //设置主窗口的图标，该窗口没有图标

//设置主窗口的窗口过程函数为HelloWinProc，所有发往该窗口的消息由该函数处理

CreateInfo.MainWindowProc = HelloWinProc;

CreateInfo.lx = 0; //设置主窗口的坐标

CreateInfo.ty = 0;

CreateInfo.rx = 600;

CreateInfo.by = 600;

CreateInfo.iBkColor = COLOR\_lightwhite;//设置主窗口的背景颜色为白色，预定义

CreateInfo.dwAddData = 0;//设置主窗口的附加数据，该窗口没有附加数据

CreateInfo.hHosting = HWND\_DESKTOP;//设置主窗口的托管窗口为桌面窗口

hMainWnd = CreateMainWindow (&CreateInfo);

**if** (hMainWnd == HWND\_INVALID)

**return** -1;

// SW\_SHOWNORMAL正常显示，并且显示在桌面顶层

ShowWindow(hMainWnd, SW\_SHOWNORMAL);

//GetMessage(&Msg, hMainWnd)第二个参数为窗口句柄，用于表示从哪个窗口的消息队列中获取消息

**while** (GetMessage(&Msg, hMainWnd)) {

**if**(getIdle()==5)

**break**;

//把击键消息转换为MES\_CHAR消息，然后直接发送到窗口处理函数

TranslateMessage(&Msg);

//调用主窗口的窗口过程函数对发往该主窗口的消息进行处理

DispatchMessage(&Msg);

i++;

**if**(i==50)

**break**;

}

**sleep**(10);

//MainWindowThreadCleanup (hMainWnd);

**return** EXIT\_SUCCESS;

}

## 3、CreateMainWindow函数

HWND GUIAPI CreateMainWindow (PMAINWINCREATE pCreateInfo)

{

PMAINWIN pWin; //指针

**if** (pCreateInfo == NULL) //判断函数参数是否为空，若为空返回窗口无效信息

**return** HWND\_INVALID;

**if** (!(pWin = **calloc**(1, **sizeof**(MAINWIN)))) //为pWin动态分配内存空间

**return** HWND\_INVALID; //若分配空间失败，返回窗口无效信息

**#ifndef** \_LITE\_VERSION //没有定义\_LITE\_VERSION，运行模式为MiniGui-Threads

**if** (pCreateInfo->hHosting == HWND\_DESKTOP || pCreateInfo->hHosting == 0) {

/\*

\*\* Create thread infomation

//如果托管窗口为桌面窗口或者没有托管窗口，为新建的主窗口创建线程信息和消息队列

//获取本线程关联的消息队列结构体，若获取失败说明该窗口是最顶层的主窗口，目前还没有对应

//的消息队列，则为其创建一个新的消息队列结构体

**if** ((pWin->pMessages = GetMsgQueueThisThread ()) == NULL) {

**if** (!(pWin->pMessages = InitMsgQueueThisThread ()) ) {

//如果创建新消息队列结构体失败，释放为pWin分配的空间，返回窗口无效信息

free (pWin);

**return** HWND\_INVALID;

}

pWin->pMessages->pRootMainWin = pWin;//如果创建消息队列结构体成功，

//则设置当前窗口为根窗口

}

**else** { //获取消息队列结构体成功，说明当前创建的窗口已经有一个对应的顶层主窗口

/\* Already have a top level main window, in case of user have set

a wrong hosting window \*/

pWin->pHosting = pWin->pMessages->pRootMainWin; //设置托管口

}

}

**else** {//托管窗口为非桌面窗口

pWin->pMessages = GetMsgQueueThisThread ();//获取本线程关联的消息队列结构体

//如果当前窗口的消息队列结构体不等于托管主窗口的消息队列结构体

//或者消息队列结构体为空，释放为pWin分配的空间，返回窗口无效信息

**if** (pWin->pMessages != GetMsgQueue (pCreateInfo->hHosting) ||

pWin->pMessages == NULL) {

free (pWin);

**return** HWND\_INVALID;

}

}

**if** (pWin->pHosting == NULL)//如果当前窗口的托管主窗口为空，

//利用传递的函数参数获得托管主窗口信息

pWin->pHosting = GetMainWindowPtrOfControl (pCreateInfo->hHosting);

/\* leave the pHosting is NULL for the first window of this thread. \*/

**#else** //运行模式为非MiniGui-Threads

//获得托管窗口的句柄

pWin->pHosting = GetMainWindowPtrOfControl (pCreateInfo->hHosting);

**if** (pWin->pHosting == NULL) //托管窗口句柄为空

pWin->pHosting = \_\_mg\_dsk\_win; //设置托管窗口为默认桌面窗口

pWin->pMessages = \_\_mg\_dsk\_msg\_queue;//将消息队列设置为默认桌面消息队列

**#endif**

pWin->pMainWin = pWin;

pWin->hParent = 0; //当前窗口的父窗口不存在

pWin->pFirstHosted = NULL; //第一个托管主窗口

pWin->pNextHosted = NULL;//下一个托管主窗口

pWin->DataType = TYPE\_HWND; //数据类型

pWin->WinType = TYPE\_MAINWIN; //窗口类型

**#ifndef** \_LITE\_VERSION

pWin->th = pthread\_self();//创建主窗口的线程

**#endif**

pWin->hFirstChild = 0; //第一个子窗口的句柄为0，即不存在

pWin->hActiveChild = 0; //当前活动的子窗口的句柄为0，即不存在

pWin->hOldUnderPointer = 0; //旧的子窗口

pWin->hPrimitive = 0;

pWin->NotifProc = NULL;

pWin->dwStyle = pCreateInfo->dwStyle;

pWin->dwExStyle = pCreateInfo->dwExStyle;

pWin->hMenu = pCreateInfo->hMenu;

pWin->hCursor = pCreateInfo->hCursor;

pWin->hIcon = pCreateInfo->hIcon;

//如果当前窗口包含标题且包含系统菜单，则创建系统菜单

**if** ((pWin->dwStyle & WS\_CAPTION) && (pWin->dwStyle & WS\_SYSMENU))

pWin->hSysMenu= CreateSystemMenu ((HWND)pWin, pWin->dwStyle);

**else //否则系统菜单项为0**

pWin->hSysMenu = 0;

pWin->pLogFont = GetSystemFont (SYSLOGFONT\_WCHAR\_DEF); //获取系统字体

pWin->spCaption = FixStrAlloc (**strlen** (pCreateInfo->spCaption));//分配空间存放标题

**if** (pCreateInfo->spCaption [0]) //如果函数参数结构体的标题字符串的第一个字符非空

**strcpy** (pWin->spCaption, pCreateInfo->spCaption);//复制标题到结构体的标题项

pWin->MainWindowProc = pCreateInfo->MainWindowProc;//消息处理函数

pWin->iBkColor = pCreateInfo->iBkColor;

pWin->pCaretInfo = NULL;//指向系统插入字符信息结构体

pWin->dwAddData = pCreateInfo->dwAddData;

pWin->dwAddData2 = 0;

//运行模式为MiniGui-Threads或者MiniGui-Standalone，为pZOrderNode分配存储空间

**#if** !defined (\_LITE\_VERSION) || defined (\_STAND\_ALONE)

**if** ( !( pWin->pZOrderNode = **malloc** (**sizeof**(ZORDERNODE))) )

**goto** err;

**#endif**

/\* Scroll bar \*/滚动条

**if** (pWin->dwStyle & WS\_VSCROLL) { //垂直方向的滚动条

pWin->vscroll.minPos = 0;

pWin->vscroll.maxPos = 100;

pWin->vscroll.curPos = 0;

pWin->vscroll.pageStep = 0;

pWin->vscroll.barStart = 0;

pWin->vscroll.barLen = 10;

pWin->vscroll.status = SBS\_NORMAL;

}

**else**

pWin->vscroll.status = SBS\_HIDE | SBS\_DISABLED;

**if** (pWin->dwStyle & WS\_HSCROLL) { //水平方向的滚动条

pWin->hscroll.minPos = 0;

pWin->hscroll.maxPos = 100;

pWin->hscroll.curPos = 0;

pWin->hscroll.pageStep = 0;

pWin->hscroll.barStart = 0;

pWin->hscroll.barLen = 10;

pWin->hscroll.status = SBS\_NORMAL;

}

**else**

pWin->hscroll.status = SBS\_HIDE | SBS\_DISABLED;

// MSG\_NCCREATE表示窗口已经创建但是还没有向系统注册

**if** (SendMessage ((HWND)pWin, MSG\_NCCREATE, 0, (LPARAM)pCreateInfo))

**goto** err;

**#ifdef** \_\_TARGET\_MGDESKTOP\_\_

pCreateInfo->lx += \_\_mg\_mainwin\_offset\_x;

pCreateInfo->rx += \_\_mg\_mainwin\_offset\_x;

pCreateInfo->ty += \_\_mg\_mainwin\_offset\_y;

pCreateInfo->by += \_\_mg\_mainwin\_offset\_y;

**#endif**

SendMessage ((HWND)pWin, MSG\_SIZECHANGING, // **HelloWinProc**

(WPARAM)&pCreateInfo->lx, (LPARAM)&pWin->left);

SendMessage ((HWND)pWin, MSG\_CHANGESIZE, (WPARAM)&pWin->left, 0); // **HelloWinProc**

pWin->pGCRInfo = &pWin->GCRInfo;

**#if** defined (\_LITE\_VERSION) && !defined (\_STAND\_ALONE)

**if** (SendMessage (HWND\_DESKTOP, MSG\_ADDNEWMAINWIN, (WPARAM) pWin, 0) < 0)

**goto** err;

**#else //**

SendMessage (HWND\_DESKTOP, MSG\_ADDNEWMAINWIN, (WPARAM) pWin,

(LPARAM) pWin->pZOrderNode);

**#endif**

/\* There was a very large bug.

\* We should add the new main window in system and then

\* SendMessage MSG\_CREATE for application to create

\* child windows.

\*/

**if** (SendMessage ((HWND)pWin, MSG\_CREATE, 0, (LPARAM)pCreateInfo)) {

SendMessage(HWND\_DESKTOP, MSG\_REMOVEMAINWIN, (WPARAM)pWin, 0);

**goto** err;

}

/\* Create private client dc. \*/

**if** (pWin->dwExStyle & WS\_EX\_USEPRIVATECDC)

pWin->privCDC = CreatePrivateClientDC ((HWND)pWin);

**else**

pWin->privCDC = 0;

**return** (HWND)pWin;

err:

**#ifndef** \_LITE\_VERSION

**if** (pWin->pMessages && pWin->pHosting == NULL) {

FreeMsgQueueThisThread ();

}

**#endif**

**#if** !defined (\_LITE\_VERSION) || defined (\_STAND\_ALONE)

**if** (pWin->pZOrderNode) **free** (pWin->pZOrderNode);

**#endif**

**if** (pWin->privCDC) DeletePrivateDC (pWin->privCDC);

**free** (pWin);

**return** HWND\_INVALID;

}

### 1、判断传入的参数pCreateInfo是否为空

Case NULL：若参数为空，返回HWND\_INVALID

Case NOT NULL：若参数不为空，继续执行2

### 2、为PMAINWIN类型的 pWin分配内存空间，并判断pWin是否为空

Case NULL：分配空间失败，返回HWND\_INVALID

Case NOT NULL：分配空间成功，继续执行3

### 3、是否定义\_LITE\_VERSION：

3.a没有定义\_LITE\_VERSION，代表minigui的运行模式为MiniGui-Threads

设置pWin的成员pWin->pMessages和pWin->pHosting

3.b定义了\_LITE\_VERSION，代表minigui的运行模式为非MiniGui-Threads

设置pWin的成员pWin->pMessages和pWin->pHosting

### 4、设置pWin的成员：

pWin->pMainWin = pWin;

pWin->hParent = 0;

pWin->pFirstHosted = NULL;

pWin->pNextHosted = NULL;

pWin->DataType = TYPE\_HWND;

pWin->WinType = TYPE\_MAINWIN;

**#ifndef** \_LITE\_VERSION

pWin->th = pthread\_self();

**#endif**

pWin->hFirstChild = 0;

pWin->hActiveChild = 0;

pWin->hOldUnderPointer = 0;

pWin->hPrimitive = 0;

pWin->NotifProc = NULL;

pWin->dwStyle = pCreateInfo->dwStyle;

pWin->dwExStyle = pCreateInfo->dwExStyle;

pWin->hMenu = pCreateInfo->hMenu;

pWin->hCursor = pCreateInfo->hCursor;

pWin->hIcon = pCreateInfo->hIcon;

**if** ((pWin->dwStyle & WS\_CAPTION) && (pWin->dwStyle & WS\_SYSMENU))

pWin->hSysMenu= CreateSystemMenu ((HWND)pWin, pWin->dwStyle);

**else**

pWin->hSysMenu = 0;

pWin->pLogFont = GetSystemFont (SYSLOGFONT\_WCHAR\_DEF);

pWin->spCaption = FixStrAlloc (**strlen** (pCreateInfo->spCaption));

**if** (pCreateInfo->spCaption [0])

**strcpy** (pWin->spCaption, pCreateInfo->spCaption);

pWin->MainWindowProc = pCreateInfo->MainWindowProc;

pWin->iBkColor = pCreateInfo->iBkColor;

pWin->pCaretInfo = NULL;

pWin->dwAddData = pCreateInfo->dwAddData;

pWin->dwAddData2 = 0;

**#if** !defined (\_LITE\_VERSION) || defined (\_STAND\_ALONE)

**if** ( !( pWin->pZOrderNode = **malloc** (**sizeof**(ZORDERNODE))) )

**goto** err;

**#endif**

/\* Scroll bar \*/

**if** (pWin->dwStyle & WS\_VSCROLL) {

pWin->vscroll.minPos = 0;

pWin->vscroll.maxPos = 100;

pWin->vscroll.curPos = 0;

pWin->vscroll.pageStep = 0;

pWin->vscroll.barStart = 0;

pWin->vscroll.barLen = 10;

pWin->vscroll.status = SBS\_NORMAL;

}

**else**

pWin->vscroll.status = SBS\_HIDE | SBS\_DISABLED;

**if** (pWin->dwStyle & WS\_HSCROLL) {

pWin->hscroll.minPos = 0;

pWin->hscroll.maxPos = 100;

pWin->hscroll.curPos = 0;

pWin->hscroll.pageStep = 0;

pWin->hscroll.barStart = 0;

pWin->hscroll.barLen = 10;

pWin->hscroll.status = SBS\_NORMAL;

}

**else**

pWin->hscroll.status = SBS\_HIDE | SBS\_DISABLED;

### 5、SendMessage ((HWND)pWin, MSG\_NCCREATE, 0, (LPARAM)pCreateInfo)

作用：什么也没做

#define MSG\_NCCREATE 0x0061

表示该窗口已经创建但是还没有向系统进行注册，当收到这种类型的消息时可以对自己创建的对象进行初始化，但不能创建子窗口，也不能进行绘图。如果函数返回值为非零值，创建的窗口将被销毁。

\* \code

\* MSG\_NCCREATE for main windows:

\* PMAINWINCREATE create\_info = (PMAINWINCREATE)lParam;

\*

\* MSG\_NCCREATE for controls:

\* DWORD add\_data = (DWORD)lParam;

\* \endcode

\*

\* \param create\_info The pointer to the MAINWINCREATE structure which is

\* passed to CreateMainWindow function.

\* \param add\_data The first additional data passed to CreateWindowEx function.

\*

\* \sa CreateMainWindow, CreateWindowEx, MAINWINCREATE

main🡪CreateMainWindow(&CreateInfo)

🡪 SendMessage ((HWND)pWin, MSG\_NCCREATE, 0, (LPARAM)pCreateInfo)

🡪 HelloWinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)

🡪 DefaultMainWinProc(hWnd, message, wParam, lParam)

(message >= MSG\_FIRSTCREATEMSG && message <= MSG\_LASTCREATEMSG)

🡪 DefaultCreateMsgHandler(pWin, message, wParam, lParam)

返回0，什么也没做

### 6、SendMessage ((HWND)pWin, MSG\_SIZECHANGING,(WPARAM)&pCreateInfo->lx, (LPARAM)&pWin->left);

#define MSG\_SIZECHANGING 0x0025

指示了将要被更改的窗口的大小，当窗口大小将要发生改变时，该消息会发送给窗口。如果你想要控制窗口改变后的实际位置和大小（窗口改变可能是MoveWindow或者其他函数引起的），你需要使用MSG\_SIZECHANGING作为SendMessage函数的第二个参数，并且通过第二个参数返回位置和大小信息。

\* \code

\* MSG\_SIZECHANGING

\* const RECT\* rcExpect = (const RECT\*)wParam;

\* RECT\* rcResult = (RECT\*)lParam;

\* \endcode

\*

\* \param rcExpect The expected size of the window after changing.

\* \param rcResult The actual size of the window after changing.

\*

main🡪CreateMainWindow(&CreateInfo)

🡪SendMessage((HWND)pWin,MSG\_SIZECHANGING,(WPARAM)&pCreateInfo->lx,(LPARAM)&pWin->left)

🡪 HelloWinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)

🡪 DefaultMainWinProc(hWnd, message, wParam, lParam)

( message >= MSG\_FIRSTPOSTMSG && message <= MSG\_LASTPOSTMSG)

🡪 DefaultPostMsgHandler(pWin, message, wParam, lParam)

memcpy ((PRECT)lParam, (PRECT)wParam, sizeof (RECT))

将wParam的信息复制给lParam，返回0

### 7、SendMessage ((HWND)pWin, MSG\_CHANGESIZE, (WPARAM)&pWin->left, 0)

#define MSG\_CHANGESIZE 0x0022

改变窗口大小

main🡪CreateMainWindow(&CreateInfo)

🡪SendMessage((HWND)pWin,MSG\_CHANGESIZE,(WPARAM)&pWin->left,0)

🡪 HelloWinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)

🡪 DefaultMainWinProc(hWnd, message, wParam, lParam)

(message >= MSG\_FIRSTPOSTMSG && message <= MSG\_LASTPOSTMSG)

(1)🡪 OnChangeSize (pWin, (PRECT)wParam, (PRECT)lParam)确定边界、标题、滚动条等的大小

(2)🡪 RecalcClientArea ((HWND)pWin) 确定客户区域的坐标和大小

### 8、SendMessage (HWND\_DESKTOP, MSG\_ADDNEWMAINWIN, (WPARAM) pWin, (LPARAM) pWin->pZOrderNode);

#define MSG\_ADDNEWMAINWIN 0x00F0

main🡪CreateMainWindow(&CreateInfo)

🡪 SendMessage (HWND\_DESKTOP, MSG\_ADDNEWMAINWIN, (WPARAM) pWin, (LPARAM) pWin->pZOrderNode)

🡪 DesktopWinProc (HWND hWnd, int message, WPARAM wParam, LPARAM lParam)

(message >= MSG\_FIRSTWINDOWMSG && message <= MSG\_LASTWINDOWMSG)

🡪 WindowMessageHandler (message, (PMAINWIN)wParam, lParam)

case MSG\_ADDNEWMAINWIN

🡪 dskAddNewMainWindow(pWin, (PZORDERNODE)lParam)

1🡪 dskUpdateGCRInfoOnShowNewMainWin (pWin)

2🡪 SendAsyncMessage ((HWND)pWin, MSG\_NCPAINT, 0, 0)(相当于SendMessage)

\* MSG\_NCPAINT:绘制非客户区域brief Indicates that paints non-client area.

**#define** MSG\_NCPAINT 0x00B2

2🡪 HelloWinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)

2🡪 DefaultMainWinProc(hWnd, message, wParam, lParam)

(message >= MSG\_FIRSTPAINTMSG && message <= MSG\_LASTPAINTMSG)

2🡪 DefaultPaintMsgHandler(pWin, message, wParam, lParam)

2🡪 wndDrawNCFrame (pWin, (HDC)wParam, (const RECT\*)lParam)

(1)🡪 wndDrawNCArea (pWin, hdc)(边框)

(2)🡪 wndDrawScrollBar (pWin, hdc)(滚动条)

(3)🡪 wndDrawCaption (pWin, hdc, !(pWin->dwStyle & WS\_DISABLED) && (GetActiveWindow() == (HWND)pWin));(标题)

(4)🡪 DrawMenuBarHelper (pWin, hdc, prcInvalid)

3🡪 SendNotifyMessage ((HWND)pWin, MSG\_SHOWWINDOW, SW\_SHOWNORMAL, 0)

4🡪InvalidateRect ((HWND)pWin, NULL, TRUE)

5🡪dskChangActiveWindow (pWin)

## 4、GetMainWindowPtrOfControl

PMAINWIN **GetMainWindowPtrOfControl** (HWND hWnd)

{

PMAINWIN pWin;

MG\_CHECK\_RET (MG\_IS\_NORMAL\_WINDOW(hWnd), NULL);

pWin = MG\_GET\_WINDOW\_PTR (hWnd);

**return** pWin->pMainWin;

}

## 5、SendMessage：

**int** GUIAPI **SendMessage** (HWND hWnd, **int** iMsg, WPARAM wParam, LPARAM lParam)

{

WNDPROC WndProc;

MG\_CHECK\_RET (MG\_IS\_WINDOW(hWnd), -1);

**if** ( !(WndProc = GetWndProc(hWnd)) )

**return** ERR\_INV\_HWND;

**return** (\*WndProc)(hWnd, iMsg, wParam, lParam);

}

## 6、HelloWinProc

**int** **HelloWinProc**(HWND hWnd, **int** message, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

BITMAP mybmp;

HWND hwnd;

**switch** (message) {/\*

hwnd = CreateWindow (CTRL\_STATIC, "Double-click me!",

WS\_VISIBLE | SS\_CENTER | SS\_NOTIFY,

50, 55, 95, 200, 20, hWnd, 0);\*/

**case** MSG\_PAINT:

hdc = BeginPaint (hWnd);

//LoadBitmapFromMem(hdc,&mybmp,g\_testbmp,90,"bmp");

//TextOut (hdc, 0, 0, "我不知道");

//TextOut (hdc, 0, 100, "1 2 3\n 4 5 6 7 8 9 0\r ");

//TextOut (hdc, 0, 200, "a b c d\r\n e f g h i j k l m n o p q r s t u v w x y z ");

//TextOut (hdc, 0, 300, "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z");

//FillBoxWithBitmap(hdc,0,0,60,70,&mybmp);

//FillBoxWithBitmap(hdc, 10, 10, 100, 100, &mybmp);

TextOut (hdc, 0, 100, "A");

EndPaint (hWnd, hdc);

**return** 0;

**case** MSG\_CLOSE:

DestroyMainWindow (hWnd);

PostQuitMessage (hWnd);

**return** 0;

}

**return** DefaultMainWinProc(hWnd, message, wParam, lParam);

}

## 7、PreDefMainWinProc（DefaultMainWinProc）

**int** **PreDefMainWinProc** (HWND hWnd, **int** message, WPARAM wParam, LPARAM lParam)

{

PMAINWIN pWin = (PMAINWIN)hWnd;

**if** (message > MSG\_DT\_MOUSEOFF && message <= MSG\_DT\_RBUTTONDBLCLK)

**return** DefaultDTMouseMsgHandler(pWin, message,

wParam, LOSWORD (lParam), HISWORD (lParam));

**else** **if** (message >= MSG\_FIRSTMOUSEMSG && message <= MSG\_NCMOUSEOFF)

**return** DefaultMouseMsgHandler(pWin, message,

wParam, LOSWORD (lParam), HISWORD (lParam));

**else** **if** (message > MSG\_NCMOUSEOFF && message <= MSG\_LASTMOUSEMSG)

**return** DefaultNCMouseMsgHandler(pWin, message,

(**int**)wParam, LOSWORD (lParam), HISWORD (lParam));

**else** **if** (message >= MSG\_FIRSTKEYMSG && message <= MSG\_LASTKEYMSG)

**return** DefaultKeyMsgHandler(pWin, message, wParam, lParam);

**else** **if** (message >= MSG\_FIRSTPOSTMSG && message <= MSG\_LASTPOSTMSG)

**return** DefaultPostMsgHandler(pWin, message, wParam, lParam);

**else** **if** (message >= MSG\_FIRSTCREATEMSG && message <= MSG\_LASTCREATEMSG)

**return** DefaultCreateMsgHandler(pWin, message, wParam, lParam);

**else** **if** (message >= MSG\_FIRSTPAINTMSG && message <= MSG\_LASTPAINTMSG)

**return** DefaultPaintMsgHandler(pWin, message, wParam, lParam);

**else** **if** (message >= MSG\_FIRSTSESSIONMSG && message <= MSG\_LASTSESSIONMSG)

**return** DefaultSessionMsgHandler(pWin, message, wParam, lParam);

**else** **if** (message >= MSG\_FIRSTCONTROLMSG && message <= MSG\_LASTCONTROLMSG)

**return** DefaultControlMsgHandler(pWin, message, wParam, lParam);

**else** **if** (message >= MSG\_FIRSTSYSTEMMSG && message <= MSG\_LASTSYSTEMMSG)

**return** DefaultSystemMsgHandler(pWin, message, wParam, lParam);

**return** 0;

}

## 8、DesktopWinProc

int DesktopWinProc (HWND hWnd, int message, WPARAM wParam, LPARAM lParam)

{

static PBITMAP bg\_bmp;

static int pic\_x, pic\_y;

static HDC hDesktopDC;

static HMENU hDesktopMenu;

RECT\* pInvalidRect;

int flags, x, y;

if (message >= MSG\_FIRSTWINDOWMSG && message <= MSG\_LASTWINDOWMSG)

return WindowMessageHandler (message, (PMAINWIN)wParam, lParam);

if (message == MSG\_CHAR && \_\_mg\_active\_mainwnd) {

PostMessage ((HWND)\_\_mg\_active\_mainwnd, message, wParam, lParam);

}

else if (message >= MSG\_FIRSTKEYMSG && message <= MSG\_LASTKEYMSG) {

if (do\_drag\_drop\_window (message, 0, 0))

return 0;

if (wParam == SCANCODE\_PRINTSCREEN && message == MSG\_KEYDOWN) {

#ifdef \_MISC\_SAVESCREEN

static int n = 1;

char buffer[20];

sprintf (buffer, "%x-%d.bmp", (lParam & KS\_CTRL)?

(HWND)\_\_mg\_active\_mainwnd:

0, n);

if (SaveMainWindowContent ((lParam & KS\_CTRL)?

(HWND)\_\_mg\_active\_mainwnd:

HWND\_DESKTOP,

buffer)) {

Ping ();

n ++;

}

#endif

}

else if (wParam == SCANCODE\_ESCAPE && lParam & KS\_CTRL) {

dskUpdateDesktopMenu (hDesktopMenu);

TrackPopupMenu (hDesktopMenu, TPM\_DEFAULT,

0, g\_rcScr.bottom, HWND\_DESKTOP);

}

if (\_\_mg\_ptmi) {

if (PopupMenuTrackProc (\_\_mg\_ptmi, message, wParam, lParam))

return KeyMessageHandler (message, (int)wParam, (DWORD)lParam);

}

else

return KeyMessageHandler (message, (int)wParam, (DWORD)lParam);

}

if (message >= MSG\_FIRSTMOUSEMSG && message <= MSG\_LASTMOUSEMSG) {

flags = (int)wParam;

x = LOSWORD (lParam);

y = HISWORD (lParam);

if (do\_drag\_drop\_window (message, x, y))

return 0;

if (\_\_mg\_ptmi) {

if (PopupMenuTrackProc (\_\_mg\_ptmi, message, x, y))

return MouseMessageHandler (message, flags, x, y);

}

else

return MouseMessageHandler (message, flags, x, y);

}

if (message == MSG\_COMMAND) {

if (wParam <= MAXID\_RESERVED && wParam >= MINID\_RESERVED)

return dskDesktopCommand (hDesktopMenu, (int)wParam);

#ifndef \_LITE\_VERSION

else

return CustomDesktopCommand ((int)wParam);

#endif

}

switch (message) {

case MSG\_STARTSESSION:

\_\_mg\_init\_local\_sys\_text ();

hDesktopDC = CreatePrivateDC (HWND\_DESKTOP);

hDesktopMenu = dskCreateDesktopMenu ();

if (dskGetBgPicturePos () < 0)

bg\_bmp = NULL;

else

bg\_bmp = dskLoadBgPicture ();

if (bg\_bmp)

dskGetBgPictureXY (dskGetBgPicturePos (),

bg\_bmp->bmWidth, bg\_bmp->bmHeight,

&pic\_x, &pic\_y);

break;

case MSG\_REINITSESSION:

if (wParam)

\_\_mg\_init\_local\_sys\_text ();

DestroyMenu (hDesktopMenu);

hDesktopMenu = dskCreateDesktopMenu ();

SendMessage (HWND\_DESKTOP, MSG\_ERASEDESKTOP, 0, 0);

break;

case MSG\_ENDSESSION:

#ifndef \_LITE\_VERSION

if ((sg\_MainWinZOrder.nNumber + sg\_TopMostWinZOrder.nNumber) == 0) {

if (hDesktopDC) DeletePrivateDC (hDesktopDC);

hDesktopDC = 0;

if (hDesktopMenu) {

DestroyMenu (hDesktopMenu);

hDesktopMenu = 0;

}

if (bg\_bmp)

UnloadBitmap (bg\_bmp);

PostQuitMessage (HWND\_DESKTOP);

return 1;

}

break;

#else

if (hDesktopDC) DeletePrivateDC (hDesktopDC);

hDesktopDC = 0;

if (hDesktopMenu) {

DestroyMenu (hDesktopMenu);

hDesktopMenu = 0;

}

if (bg\_bmp)

UnloadBitmap (bg\_bmp);

return 1;

#endif

case MSG\_ERASEDESKTOP:

SetBrushColor (hDesktopDC, GetWindowElementColor (BKC\_DESKTOP));

pInvalidRect = (PRECT)lParam;

if (pInvalidRect) {

SelectClipRect (hDesktopDC, pInvalidRect);

FillBox (hDesktopDC, pInvalidRect->left, pInvalidRect->top,

RECTWP (pInvalidRect), RECTHP (pInvalidRect));

}

else {

SelectClipRect (hDesktopDC, &g\_rcDesktop);

FillBox(hDesktopDC, g\_rcDesktop.left, g\_rcDesktop.top,

g\_rcDesktop.right,

g\_rcDesktop.bottom);

}

if (bg\_bmp) {

FillBoxWithBitmap (hDesktopDC, pic\_x, pic\_y,

bg\_bmp->bmWidth, bg\_bmp->bmHeight, bg\_bmp);

}

break;

#ifdef \_LITE\_VERSION

case MSG\_SETFOCUS:

if (\_\_mg\_active\_mainwnd) {

SendNotifyMessage ((HWND)\_\_mg\_active\_mainwnd, MSG\_SETFOCUS, 0, 0);

}

break;

#endif

case MSG\_PAINT:

{

PZORDERNODE pNode;

PMAINWIN pTemp;

RECT invrc;

BOOL is\_empty\_invrc = FALSE;

invrc.left = LOWORD(wParam);

invrc.top = HIWORD(wParam);

invrc.right = LOWORD(lParam);

invrc.bottom = HIWORD(lParam);

if (invrc.top == invrc.bottom || invrc.left == invrc.right)

is\_empty\_invrc = TRUE;

// close the active menu first

dskForceCloseMenu ();

SendMessage (HWND\_DESKTOP,

MSG\_ERASEDESKTOP, 0,

(LPARAM)(is\_empty\_invrc?0:&invrc));

pNode = sg\_TopMostWinZOrder.pTopMost;

while (pNode) {

pTemp= (PMAINWIN)(pNode->hWnd);

if (pTemp->WinType != TYPE\_CONTROL

&& pTemp->dwStyle & WS\_VISIBLE) {

if (is\_empty\_invrc) {

SendAsyncMessage ((HWND)pTemp, MSG\_NCPAINT, 0, 0);

InvalidateRect ((HWND)pTemp, NULL, TRUE);

}

else {

RECT rcTemp, rcInv;

if (IntersectRect (&rcTemp, (RECT\*)(&pTemp->left),

&invrc)) {

dskScreenToWindow (pTemp, &rcTemp, &rcInv);

SendAsyncMessage ((HWND)pTemp,

MSG\_NCPAINT, 0, (LPARAM)(&rcInv));

dskScreenToClient (pTemp, &rcTemp, &rcInv);

InvalidateRect ((HWND)pTemp, &rcInv, TRUE);

}

}

}

pNode = pNode->pNext;

}

pNode = sg\_MainWinZOrder.pTopMost;

while (pNode) {

pTemp= (PMAINWIN)(pNode->hWnd);

if (pTemp->WinType != TYPE\_CONTROL

&& pTemp->dwStyle & WS\_VISIBLE) {

if (is\_empty\_invrc) {

SendAsyncMessage ((HWND)pTemp, MSG\_NCPAINT, 0, 0);

InvalidateRect ((HWND)pTemp, NULL, TRUE);

}

else {

RECT rcTemp, rcInv;

if (IntersectRect (&rcTemp, (RECT\*)(&pTemp->left),

&invrc)) {

dskScreenToWindow (pTemp, &rcTemp, &rcInv);

SendAsyncMessage ((HWND)pTemp,

MSG\_NCPAINT, 0, (LPARAM)(&rcInv));

dskScreenToClient (pTemp, &rcTemp, &rcInv);

InvalidateRect ((HWND)pTemp, &rcInv, TRUE);

}

}

}

pNode = pNode->pNext;

}

}

break;

case MSG\_BROADCASTMSG:

return dskBroadcastMessage ((PMSG)lParam);

case MSG\_REGISTERWNDCLASS:

return AddNewControlClass ((PWNDCLASS)lParam);

case MSG\_UNREGISTERWNDCLASS:

return DeleteControlClass ((const char\*)lParam);

case MSG\_NEWCTRLINSTANCE:

dskOnNewCtrlInstance ((PCONTROL)wParam, (PCONTROL)lParam);

break;

case MSG\_REMOVECTRLINSTANCE:

if (!dskOnRemoveCtrlInstance ((PCONTROL)wParam, (PCONTROL)lParam))

return -1;

break;

case MSG\_GETCTRLCLASSINFO:

return (int)GetControlClassInfo ((const char\*)lParam);

case MSG\_CTRLCLASSDATAOP:

return (int)ControlClassDataOp (wParam, (WNDCLASS\*)lParam);

case MSG\_REGISTERKEYHOOK:

return (int)dskRegisterKeyHook ((void\*)wParam,

(MSGHOOK)lParam);

case MSG\_REGISTERMOUSEHOOK:

return (int)dskRegisterMouseHook ((void\*)wParam,

(MSGHOOK)lParam);

case MSG\_IME\_REGISTER:

return dskRegisterIMEWnd ((HWND)wParam);

case MSG\_IME\_UNREGISTER:

return dskUnregisterIMEWnd ((HWND)wParam);

case MSG\_IME\_SETSTATUS:

return dskSetIMEStatus ((int)wParam, (int)lParam);

case MSG\_IME\_GETSTATUS:

return dskGetIMEStatus ((int)wParam);

#ifdef \_LITE\_VERSION

case MSG\_SRVNOTIFY:

BroadcastMessage (MSG\_SRVNOTIFY, wParam, lParam);

break;

#endif

case MSG\_TIMEOUT:

BroadcastMessage (MSG\_IDLE, wParam, 0);

break;

case MSG\_DT\_KEYLONGPRESS:

case MSG\_DT\_KEYALWAYSPRESS:

case MSG\_DT\_KEYDOWN:

case MSG\_DT\_CHAR:

case MSG\_DT\_KEYUP:

case MSG\_DT\_SYSKEYDOWN:

case MSG\_DT\_SYSCHAR:

case MSG\_DT\_SYSKEYUP:

case MSG\_DT\_LBUTTONDOWN:

case MSG\_DT\_LBUTTONUP:

case MSG\_DT\_LBUTTONDBLCLK:

case MSG\_DT\_MOUSEMOVE:

case MSG\_DT\_RBUTTONDOWN:

case MSG\_DT\_RBUTTONDBLCLK:

break;

case MSG\_DT\_RBUTTONUP:

x = LOSWORD (lParam);

y = HISWORD (lParam);

dskUpdateDesktopMenu (hDesktopMenu);

TrackPopupMenu (hDesktopMenu, TPM\_DEFAULT, x, y, HWND\_DESKTOP);

break;

case MSG\_TIMER: // per 0.01s

{

static UINT uCounter = 0;

#ifdef \_LITE\_VERSION

static UINT blink\_counter = 0;

static UINT sg\_old\_counter = 0;

if (sg\_old\_counter == 0)

sg\_old\_counter = \_\_mg\_timer\_counter;

DispatchTimerMessage (\_\_mg\_timer\_counter - sg\_old\_counter);

sg\_old\_counter = \_\_mg\_timer\_counter;

if (\_\_mg\_timer\_counter < (blink\_counter + 10))

break;

uCounter += (\_\_mg\_timer\_counter - blink\_counter) \* 10;

blink\_counter = \_\_mg\_timer\_counter;

#else

DispatchTimerMessage (1);

if (\_\_mg\_timer\_counter % 10 != 0)

break;

uCounter += 100;

#endif

if (sg\_hCaretWnd != 0

&& GetMainWindowPtrOfControl (sg\_hCaretWnd) == \_\_mg\_active\_mainwnd

&& uCounter >= sg\_uCaretBTime) {

PostMessage (sg\_hCaretWnd, MSG\_CARETBLINK, 0, 0);

uCounter = 0;

}

}

break;

}

return 0;

}

## 10、wndDrawNCFrame

**static** **void** **wndDrawNCFrame**(MAINWIN\* pWin, HDC hdc, **const** RECT\* prcInvalid)

{

BOOL fGetDC = FALSE;

/\* to avoid some unexpected event... \*/

**if** (!MG\_IS\_WINDOW ((HWND)pWin))

**return**;

**if** (hdc == 0) {

hdc = GetDC ((HWND)pWin);

fGetDC = TRUE;

}

**if** (prcInvalid)

ClipRectIntersect (hdc, prcInvalid);

wndDrawNCArea (pWin, hdc);

wndDrawScrollBar (pWin, hdc);

**if** (pWin->WinType == TYPE\_MAINWIN) {

wndDrawCaption (pWin, hdc, !(pWin->dwStyle & WS\_DISABLED)

&& (GetActiveWindow() == (HWND)pWin));

DrawMenuBarHelper (pWin, hdc, prcInvalid);

}

**else** {

wndDrawCaption (pWin, hdc, !(pWin->dwStyle & WS\_DISABLED) &&

((PCONTROL)pWin)->pParent->active == (PCONTROL)pWin);

}

**if** (fGetDC)

ReleaseDC (hdc);

}

## 11、wndDrawNCArea

**static** **void** **wndDrawNCArea** (**const** MAINWIN\* pWin, HDC hdc)

{

/\* Draw window frame \*/

**if** (pWin->dwStyle & WS\_BORDER) {

**#if** defined(\_FLAT\_WINDOW\_STYLE)

**int** i, iBorder;

iBorder = GetMainWinMetrics (MWM\_BORDER);

SetPenColor (hdc, GetWindowElementColorEx ((HWND)pWin, WEC\_FLAT\_BORDER));

**for** (i = 0; i < iBorder; i++)

Rectangle (hdc, i, i,

pWin->right - pWin->left - i - 1,

pWin->bottom - pWin->top - i - 1);

**#elif** defined \_PHONE\_WINDOW\_STYLE

**int** i, iBorder;

iBorder = GetMainWinMetrics (MWM\_BORDER);

SetPenColor (hdc, GetWindowElementColorEx ((HWND)pWin, WEC\_FLAT\_BORDER));

**for** (i = 0; i < iBorder; i++){

Rectangle (hdc, i, i,

pWin->right - pWin->left - i - 1,

pWin->bottom - pWin->top - i - 1);

SetPenColor (hdc,RGB2Pixel(hdc, 0x9c, 0xa1, 0xac));/\* set inner border color \*/

}

**#else**

**if** (pWin->dwStyle & WS\_CHILD)

Draw3DThickFrameEx (hdc, (HWND)pWin,

0, 0,

pWin->right - pWin->left - 1,

pWin->bottom - pWin->top - 1,

DF\_3DBOX\_PRESSED | DF\_3DBOX\_NOTFILL, 0);

**else**

Draw3DThickFrameEx (hdc, (HWND)pWin,

0, 0,

pWin->right - pWin->left,

pWin->bottom - pWin->top,

DF\_3DBOX\_NORMAL | DF\_3DBOX\_NOTFILL, 0);

**#endif**

}

**else** **if** ((pWin->dwStyle & WS\_THICKFRAME) ||

(pWin->dwStyle & WS\_THINFRAME))

{

SetPenColor(hdc, GetWindowElementColorEx ((HWND)pWin, WEC\_FRAME\_NORMAL));

Rectangle(hdc, 0, 0,

pWin->right - pWin->left - 1,

pWin->bottom - pWin->top - 1);

}

}

## 12 wndDrawScrollBar

**static** **void** **wndDrawScrollBar** (MAINWIN\* pWin, HDC hdc)

{

**int** start = 0;

RECT rcHBar, rcVBar, rcSlider;

PBITMAP bmp;

**int** xo, yo, bw, bh, w , h , xos, yos, r;

**float** scaleX = 0.0f, scaleY = 0.0f;

wndGetVScrollBarRect (pWin, &rcVBar);

rcVBar.left -= pWin->left;

rcVBar.top -= pWin->top;

rcVBar.right -= pWin->left;

rcVBar.bottom -= pWin->top;

wndGetHScrollBarRect (pWin, &rcHBar);

rcHBar.left -= pWin->left;

rcHBar.top -= pWin->top;

rcHBar.right -= pWin->left;

rcHBar.bottom -= pWin->top;

bmp = GetSystemBitmap (SYSBMP\_ARROWS);

bw = bmp->bmWidth >> 2;

bh = bmp->bmHeight >> 1;

**if** ((pWin->dwStyle & WS\_HSCROLL) && (pWin->dwStyle & WS\_VSCROLL)

&& !(pWin->hscroll.status & SBS\_HIDE

&& pWin->vscroll.status & SBS\_HIDE)) {

/\* Always erase the bottom-right corner \*/

**#ifdef** \_FLAT\_WINDOW\_STYLE

SetBrushColor (hdc, PIXEL\_lightwhite);

**#else**

SetBrushColor (hdc, GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

**#endif**

FillBox (hdc, rcVBar.left, rcHBar.top, RECTW (rcVBar), RECTH (rcHBar));

}

**if** (pWin->dwStyle & WS\_HSCROLL

&& !(pWin->hscroll.status & SBS\_HIDE) && RectVisible (hdc, &rcHBar)) {

r = GetMainWinMetrics (MWM\_CXHSCROLL);

scaleX = (**float**)r/(**float**)bw;

scaleY = (**float**)r/(**float**)bh;

//mjgfb ceil

w = (**int**)(bmp->bmWidth\*scaleX);

h = (**int**)(bmp->bmHeight\*scaleY);

/\* draw left and right buttons. \*/

**if** (pWin->hscroll.status & SBS\_DISABLED) {

xo = bw << 1; yo = bh;

}

**else** {

xo = bw << 1; yo = 0;

}

**if** (GetMainWinMetrics (MWM\_CXHSCROLL) != 0) {

//mjgfb ceil

xos = (**int**)(xo\*scaleX);

yos = (**int**)(yo\*scaleY);

FillBoxWithBitmapPart (hdc, rcHBar.left, rcHBar.top,

r, r, w, h, bmp, xos, yos);

}

**if** (pWin->hscroll.status & SBS\_DISABLED) {

xo = (bw << 1) + bw; yo = bh;

}

**else** {

xo = (bw << 1) + bw; yo = 0;

}

**if** (GetMainWinMetrics (MWM\_CXHSCROLL) != 0) {

//mjgfb ceil

xos = (**int**)(xo\*scaleX);

yos = (**int**)(yo\*scaleY);

FillBoxWithBitmapPart (hdc,

rcHBar.right - GetMainWinMetrics (MWM\_CXHSCROLL), rcHBar.top,

r, r, w, h, bmp, xos, yos);

}

/\* draw slider. \*/

start = rcHBar.left +

GetMainWinMetrics (MWM\_CXHSCROLL) +

pWin->hscroll.barStart;

**if** (start + pWin->hscroll.barLen > rcHBar.right)

start = rcHBar.right - pWin->hscroll.barLen;

rcSlider.left = start;

rcSlider.top = rcHBar.top;

rcSlider.right = start + pWin->hscroll.barLen;

rcSlider.bottom = rcHBar.bottom;

**#ifdef** \_FLAT\_WINDOW\_STYLE

SetBrushColor (hdc, GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

FillBox (hdc, start + 1, rcHBar.top + 1,

pWin->hscroll.barLen - 2, RECTH (rcHBar) - 1);

SetPenColor (hdc, PIXEL\_black);

Rectangle (hdc, rcSlider.left, rcSlider.top, rcSlider.right - 1, rcSlider.bottom);

xo = (rcSlider.left + rcSlider.right) >> 1;

MoveTo (hdc, xo, rcSlider.top + 2);

LineTo (hdc, xo, rcSlider.bottom - 1);

MoveTo (hdc, xo - 2, rcSlider.top + 3);

LineTo (hdc, xo - 2, rcSlider.bottom - 2);

MoveTo (hdc, xo + 2, rcSlider.top + 3);

LineTo (hdc, xo + 2, rcSlider.bottom - 2);

SetPenColor (hdc, PIXEL\_black);

Rectangle (hdc, rcHBar.left - 1, rcHBar.top, rcHBar.right, rcHBar.bottom);

**#elif** defined(\_PHONE\_WINDOW\_STYLE)

DrawBoxFromBitmap (hdc, &rcSlider, GetSystemBitmap (SYSBMP\_SCROLLBAR\_HSLIDER), TRUE, FALSE);

**#else**

Draw3DThickFrameEx (hdc, (HWND)pWin,

rcSlider.left, rcHBar.top,

rcSlider.right, rcHBar.bottom,

DF\_3DBOX\_NORMAL | DF\_3DBOX\_FILL,

GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

**#endif**

rcHBar.left += GetMainWinMetrics (MWM\_CXHSCROLL);

rcHBar.right -= GetMainWinMetrics (MWM\_CXHSCROLL);

/\* Exclude the rect of slider \*/

ExcludeClipRect (hdc, &rcSlider);

/\* Draw back bar \*/

**#ifdef** \_FLAT\_WINDOW\_STYLE

SetBrushColor (hdc, PIXEL\_lightwhite);

FillBox (hdc, rcHBar.left + 1, rcHBar.top + 1, RECTW(rcHBar) - 1, RECTH (rcHBar) - 1);

**#elif** defined(\_PHONE\_WINDOW\_STYLE)

DrawBoxFromBitmap (hdc, &rcHBar, GetSystemBitmap (SYSBMP\_SCROLLBAR\_HBG), TRUE, FALSE);

**#else**

SetBrushColor (hdc, GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

FillBox (hdc, rcHBar.left, rcHBar.top, RECTW(rcHBar), RECTH (rcHBar));

**#endif**

}

**if** (pWin->dwStyle & WS\_VSCROLL

&& !(pWin->vscroll.status & SBS\_HIDE) && RectVisible (hdc, &rcVBar)) {

r = GetMainWinMetrics (MWM\_CYVSCROLL);

scaleX = (**float**)r/(**float**)bw;

scaleY = (**float**)r/(**float**)bh;

w = (**int**)(bmp->bmWidth\*scaleX);

h = (**int**)(bmp->bmHeight\*scaleY);

/\* draw top and bottom arrow buttons. \*/

**if** (pWin->vscroll.status & SBS\_DISABLED) {

xo = 0; yo = bh;

}

**else** {

xo = 0; yo = 0;

}

**if** (GetMainWinMetrics (MWM\_CYVSCROLL) != 0) {

//mjgfb ceil

xos = (**int**)(xo\*scaleX);

yos = (**int**)(yo\*scaleY);

FillBoxWithBitmapPart (hdc, rcVBar.left, rcVBar.top,

r, r, w, h, bmp, xos, yos);

}

**if** (pWin->vscroll.status & SBS\_DISABLED) {

xo = bw; yo = bh;

}

**else** {

xo = bw; yo = 0;

}

**if** (GetMainWinMetrics (MWM\_CYVSCROLL) != 0) {

//mjgfb ceil

xos = (**int**)(xo\*scaleX);

yos = (**int**)(yo\*scaleY);

FillBoxWithBitmapPart (hdc,

rcVBar.left, rcVBar.bottom - GetMainWinMetrics (MWM\_CYVSCROLL),

r, r, w, h, bmp, xos, yos);

}

/\* draw slider \*/

start = rcVBar.top +

GetMainWinMetrics (MWM\_CYVSCROLL) +

pWin->vscroll.barStart;

**if** (start + pWin->vscroll.barLen > rcVBar.bottom)

start = rcVBar.bottom - pWin->vscroll.barLen;

rcSlider.left = rcVBar.left;

rcSlider.top = start;

rcSlider.right = rcVBar.right;

rcSlider.bottom = start + pWin->vscroll.barLen;

**#ifdef** \_FLAT\_WINDOW\_STYLE

SetBrushColor (hdc, GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

FillBox (hdc, rcVBar.left + 1, start + 1,

RECTW (rcVBar) - 1, pWin->vscroll.barLen - 2);

SetPenColor (hdc, PIXEL\_black);

Rectangle (hdc, rcSlider.left, rcSlider.top, rcSlider.right, rcSlider.bottom - 1);

yo = (rcSlider.top + rcSlider.bottom) >> 1;

MoveTo (hdc, rcSlider.left + 2, yo);

LineTo (hdc, rcSlider.right - 1, yo);

MoveTo (hdc, rcSlider.left + 3, yo - 2);

LineTo (hdc, rcSlider.right - 2, yo - 2);

MoveTo (hdc, rcSlider.left + 3, yo + 2);

LineTo (hdc, rcSlider.right - 2, yo + 2);

SetPenColor (hdc, PIXEL\_black);

Rectangle (hdc, rcVBar.left, rcVBar.top - 1, rcVBar.right, rcVBar.bottom);

**#elif** defined(\_PHONE\_WINDOW\_STYLE)

DrawBoxFromBitmap (hdc, &rcSlider, GetSystemBitmap (SYSBMP\_SCROLLBAR\_VSLIDER), FALSE, FALSE);

**#else**

Draw3DThickFrameEx (hdc, (HWND)pWin,

rcVBar.left, rcSlider.top, rcVBar.right, rcSlider.bottom,

DF\_3DBOX\_NORMAL | DF\_3DBOX\_FILL,

GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

**#endif**

rcVBar.top += GetMainWinMetrics (MWM\_CYVSCROLL);

rcVBar.bottom -= GetMainWinMetrics (MWM\_CYVSCROLL);

/\* Exclude the rect of slider \*/

ExcludeClipRect (hdc, &rcSlider);

/\* draw back bar \*/

**#ifdef** \_FLAT\_WINDOW\_STYLE

SetBrushColor (hdc, PIXEL\_lightwhite);

FillBox (hdc, rcVBar.left + 1, rcVBar.top + 1, RECTW (rcVBar) - 1, RECTH (rcVBar) - 1);

**#elif** defined(\_PHONE\_WINDOW\_STYLE)

DrawBoxFromBitmap (hdc, &rcVBar, GetSystemBitmap (SYSBMP\_SCROLLBAR\_VBG), FALSE, FALSE);

**#else**

SetBrushColor (hdc, GetWindowElementColorEx ((HWND)pWin, BKC\_CONTROL\_DEF));

FillBox (hdc, rcVBar.left, rcVBar.top, RECTW (rcVBar), RECTH (rcVBar));

**#endif**

}

}

## 、dskAddNewMainWindow

static PMAINWIN dskAddNewMainWindow (PMAINWIN pWin, PZORDERNODE pNode)

{

// Handle main window hosting.

if (pWin->pHosting)

dskAddNewHostedMainWindow (pWin->pHosting, pWin);

// Init Global Clip Region info.

init\_gcrinfo (pWin);

// Init Invalid Region info.

init\_invrgn (pWin);

// Update Z Order info.

pNode->hWnd = (HWND)pWin;

add\_new\_window (pWin, pNode);

// show and active this main window.

if ( pWin->dwStyle & WS\_VISIBLE ) {

dskUpdateGCRInfoOnShowNewMainWin (pWin);

SendAsyncMessage ((HWND)pWin, MSG\_NCPAINT, 0, 0);

SendNotifyMessage ((HWND)pWin, MSG\_SHOWWINDOW, SW\_SHOWNORMAL, 0);

InvalidateRect ((HWND)pWin, NULL, TRUE);

return dskChangActiveWindow (pWin);

}

## dskUpdateGCRInfoOnShowNewMainWin

static void dskUpdateGCRInfoOnShowNewMainWin (MAINWIN\* pWin)

{

RECT rcWin;

dskGetWindowRectInScreen (pWin, &rcWin);

if (pWin->dwExStyle & WS\_EX\_TOPMOST) {

clip\_windows\_under\_this (&sg\_TopMostWinZOrder, pWin, &rcWin);

clip\_windows (&sg\_MainWinZOrder, &rcWin);

}

else {

start\_clip\_window (pWin);

clip\_by\_windows (&sg\_TopMostWinZOrder, pWin);

end\_clip\_window (pWin);

clip\_windows\_under\_this (&sg\_MainWinZOrder, pWin, &rcWin);

}

clip\_desktop (&rcWin);

}

## dskGetWindowRectInScreen

static void dskGetWindowRectInScreen (PMAINWIN pWin, RECT\* prc)

{

PCONTROL pParent;

PCONTROL pCtrl;

pParent = pCtrl = (PCONTROL)pWin;

prc->left = pCtrl->left;

prc->top = pCtrl->top;

prc->right = pCtrl->right;

prc->bottom = pCtrl->bottom;

while ((pParent = pParent->pParent)) {

prc->left += pParent->cl;

prc->top += pParent->ct;

prc->right += pParent->cl;

prc->bottom += pParent->ct;

}

}