|  |  |
| --- | --- |
| freetype | 字库 |
| gvfb/qvfb | 虚拟显存，主要是在PC机上模拟 |
| jpeg | 显示jpeg库 |
| libpng | 显示png库 |
| zlib | png压缩算法 |
| libminigui | 3.0核心库 |
| minigui\_res | 资源文件，包括字体，图标，位图等 |
| mg-samples | 实例 |

|  |  |
| --- | --- |
| mGi | 输入法组件，该组件目前提供了软键盘输入法和手写输入法框架，并提供给用户管理输入法的容器，通过这个容器，用户可以添加自己的输入法。此外，对于软键盘输入法，用户可以自定义键盘的位图并添加不同的输入翻译方式(自带中英文全拼输入法) |
| mGplus | 对minigui图形绘制接口的一个扩展和增强，主要是提供对二维矢量图形和高级图形算法的支持。如路径，渐变填充。 |
| mG3d | 3D接口 |
| mGutils | 提供一些常用额对话框 |
| mGp | 打印组件 |
| mGncs | 新的空间集合 |
| mGeff | 动画框架 |

|  |
| --- |
| Minigui的GDL分为GAL，DC和surface。位图管理，字体管理及图元绘制(包括矩形和圆形的绘制和填充) |

1. 创建主窗口

|  |  |
| --- | --- |
| **MAINWINCREATE结构体** |  |
| CreateInfo.dwStyle | 窗口风格 |
| CreateInfo.spCaption | 窗口的标题 |
| CreateInfo.dwExStyle | 窗口的附加风格 |
| CreateInfo.hMenu | 附加在窗口上的菜单句柄 |
| CreateInfo.hCursor | 在窗口中所使用的鼠标光标句柄 |
| CreateInfo.hIcon | 程序的图标 |
| CreateInfo.MainWindowProc | 该窗口的消息处理函数指针(回调函数指针) |
| CreateInfo.lx | 窗口左上角相对屏幕的绝对横坐标，以象素点表示 |
| CreateInfo.ty | 窗口左上角相对屏幕的绝对纵坐标，以象素点表示 |
| CreateInfo.rx | 窗口右下角相对屏幕的绝对横坐标，以象素点表示 |
| CreateInfo.by | 窗口右下角相对屏幕的绝对纵坐标，以象素点表示 |
| CreateInfo.iBkColor | 窗口背景颜色 |
| CreateInfo.dwAddData | 附带给窗口的一个 32 位值 |
| CreateInfo.hHosting | 窗口消息队列的托管窗口 |
|  |  |
| CreateMainWindow(PMAINWINCREATE pCreateInfo)---用来给MAINWIN赋值 | |

|  |  |  |
| --- | --- | --- |
| 1. **窗口的风格接扩展风格** | | |
| 风格标识 | 含义 | 备注 |
| WS\_NONE | 未指定任何风格 |  |
| WS\_VISIBLE | 创建初始可见的窗口 |  |
| WS\_DISABLED | 创建初始被禁止的窗口 |  |
| WS\_CAPTION | 创建含标题栏的主窗口 | 仅用于主窗口，无标题栏，无法直接通过拖动鼠标来移动窗口 |
| WS\_SYSMENU | 创建含系统菜单的主窗口 | 仅用于主窗口，右键标题栏 |
| WS\_BORDER | 创建有边框的窗口 |  |
| WS\_THICKFRAME | 创建具有厚边框的窗口 |  |
| WS\_THINFRAME | 创建具有薄边框的窗口 |  |
| WS\_VSCROLL | 创建带垂直滚动条的窗口 |  |
| WS\_HSCROLL | 创建带水平滚动条的窗口 |  |
| WS\_MINIMIZEBOX | 标题栏上带最小化按钮 | 仅用于主窗口 |
| WS\_MAXIMIZEBOX | 标题栏上带最大化按钮 | 仅用于主窗口 |
| WS\_EX\_NONE | 无扩展风格 |  |
| WS\_EX\_USEPRIVATECDC | 使用私有 DC | 仅用于主窗口 |
| WS\_EX\_TOPMOST | 建立始终处于顶层的主窗口 | 仅用于主窗口 |
| WS\_EX\_TOOLWINDOW | 建立Tooltip 主窗口 | 仅用于主窗口。Tooltip 主窗口将不会拥有输入焦点，但仍接收鼠标消息 |
| WS\_EX\_USEPARENTFONT | 使用父窗口字体作为默认字体 |  |
| WS\_EX\_USEPARENTCURSOR | 使用父窗口光标作为默认光标 |  |
| WS\_EX\_NOCLOSEBOX | 主窗口标题栏上不带关闭按钮 |  |
| WS\_EX\_CTRLASMAINWIN | 建立可显示在主窗口之外的控件 | 仅用于控件 |

|  |  |  |
| --- | --- | --- |
| 主窗口的实体指针MAINWIN结构体 | | |
| 结构体 | 解释 | 备注 |
| Unsigned char DataType | 数据类型 | TYPE\_HWND  TYPE\_HMENU  TYPE\_HACCEL  TYPE\_HCURSOR  TYPE\_HICON  TYPE\_HDC  TYPE\_WINTODEL |
| Unsigned char WinType | 区分主窗口和控件窗口 | TYPE\_MAINWIN  TYPE\_CONTROL  TYPE\_ROOTWIN |
| Int left, top, right, bottom |  |  |
| DWORD dwStyle | 窗口模式 |  |
| DWORD dwExStyle | 扩展模式 |  |
| int cl, ct, cr, cb | 位置和客户区域大小 |  |
| int iBkColor | 背景色 |  |
| HMENU hMenu | 菜单句柄 |  |
| HACCEL hAccel | handle of accelerator table | ？？？ |
| HICON hIcon | 图标句柄 |  |
| HCURSOR hCursor | 鼠标句柄 |  |
| HENU hSysMenu | 系统菜单句柄 |  |
| POLGFONT pLogFont | 逻辑字体指针 |  |
| char \*spCaption | 标题 |  |
| int id | ID号 |  |
| LFSCROLLBARINFO vscroll | 垂直滚动条信息 |  |
| LFSCROLLBARINFO hscroll | 水平滚动条信息 |  |
| WINDOW\_ELEMENT\_RENDERER\* we\_rdr | 窗口渲染器 | ？？？ |
| HDC privCDC | the private client DC |  |
| INVRGN InvRgn | 无效区域 |  |
| PGCRINFO pGCRInfo | 全局裁剪去结构体信息 |  |
| int idx\_znode | Z序 |  |
| PCARETINFO pCaretInfo | the address of main window procedure | ？？？ |
| DWORD dwAddData | 附加数据1 |  |
| DWORD dwAddData2 | 附加数据2 |  |
| int (\*MainWindowProc)(HWND, int, WPARAM, LPARAM) |  |  |
| struct \_MAINWIN\* pMainWin | 包含改窗口的窗口指针，对于主窗口就是自己 | pMainWin必须是窗口树的根 |
| HWND hParent | 父窗口句柄，对于主窗口是HWND\_DESKTOP | hParent可以是一个主窗口也可以使控件窗口 |
| HWND hFirstChild |  |  |
| HWND hActiveChild |  |  |
| HWND hOldUnderPointer |  |  |
| HWND hPrimitive |  |  |
| NOTIFPROC NotifProc | 通知回调函数 | typedef void (\* NOTIFPROC) (HWND hwnd, int id, int nc, DWORD add\_data); |
| struct \_wnd\_element\_data\* wed | window element data | ？？ |
| The following members are only implemented for main window. | | |
| struct \_MAINWIN\* pHosting |  |  |
| struct \_MAINWIN\* pFirstHosted |  |  |
| struct \_MAINWIN\* pNextHosted |  |  |
| PMSGQUEUE pMessages | 消息队列 |  |
| GCRINFO GCRInfo | global clip region info struct | ??? |
| HWND hFirstChildAsMainWin |  |  |
| HDC secondaryDC | the secondary window dc | ?? |
| ON\_UPDATE\_SECONDARYDC update\_secdc | the callback of secondary window dc |  |
| RECT update\_rc |  | ？？？ |

主窗口的销毁

|  |
| --- |
| DestroyMainWindow (hWnd) 函数，发送 MSG\_DESTROY 消息。不会销毁主窗口所使用的消息队列以及窗口对象本身。  MainWindowCleaup 最终清除主窗口所使用的消息队列以及窗口对象本身 |

对话框窗口

|  |
| --- |
| 对话框模板  CTRLDATA结构体(10个元素)----用来初始化CONTROL结构体  typedef struct {  char\* class\_name; // control class  DWORD dwStyle; // control style  int x, y, w, h; // control position in dialog  int id; // control identifier  const char\* caption; // control caption  DWORD dwAddData; // additional data  DWORD dwExStyle; // control extended style } CTRLDATA; typedef CTRLDATA\* PCTRLDATA;--------用来定义控件 typedef struct {  DWORD dwStyle; // dialog box style  DWORD dwExStyle; // dialog box extended style  int x, y, w, h; // dialog box position  const char\* caption; // dialog box caption  HICON hIcon; // dialog box icon  HMENU hMenu; // dialog box menu  int controlnr; // number of controls  PCTRLDATA controls; // poiter to control array DWORD dwAddData; // addtional data, must be zero  } DLGTEMPLATE; typedef DLGTEMPLATE\* PDLGTEMPLATE;----------用来定义对话框本身 |

|  |
| --- |
| CreateWindow(class\_name, caption, style, id, x, y, w, h, parent, add\_data) 函数直接创建所有需要的子窗口  创建模态对话框：  int GUIAPI DialogBoxIndirectParam (PDLGTEMPLATE pDlgTemplate, HWND hOwner, WNDPROC DlgProc, LPARAM lParam);  BOOL GUIAPI EndDialog (HWND hDlg, int endCode);----结束对话框过程  void GUIAPI DestroyAllControls (HWND hDlg);------销毁对话框 |

|  |
| --- |
| 用户自定义消息ID范围 MSG\_USER：0x0800-----MSG\_LASTUSERMSG:0xEFFF |

|  |
| --- |
| typedef struct tagCONTROL  {  /\*These fields are similiar with MAINWIN struct.\*/  unsigned char DataType; // the data type  unsigned char WinType; // the window type  unsigned short Flags; // speical runtime flags, such EraseBkGnd flags  int left, top; // the position of control in main window's  int right, bottom; // client area.  int cl, ct; // the positio of control client in main window's  int cr, cb; // client area.  DWORD dwStyle; // the styles of child window.  DWORD dwExStyle; // the extended styles of child 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.  char\* spCaption; // the caption of control.  int id; // the identifier of child window.  LFSCROLLBARINFO vscroll; // the vertical scroll bar information.  LFSCROLLBARINFO hscroll; // the horizital scroll bar information.  /\*\* the window renderer \*/  WINDOW\_ELEMENT\_RENDERER\* we\_rdr;  HDC privCDC; // the private client DC.  INVRGN InvRgn; // the invalid region of this control.  PGCRINFO pGCRInfo; // pointer to global clip region info struct.  // the Z order node,  // only for control with WS\_EX\_CTRLASMAINWIN.  int idx\_znode;  PCARETINFO pCaretInfo; // pointer to system caret info struct.  DWORD dwAddData; // the additional data.  DWORD dwAddData2; // the second addtional data.  int (\*ControlProc) (HWND, int, WPARAM, LPARAM);  PMAINWIN pMainWin; // the main window that contains this control.  struct tagCONTROL\* pParent;// the parent of this control.  /\* Child windows.\*/  struct tagCONTROL\* children;// the first child control.  struct tagCONTROL\* active; // the active child control.  struct tagCONTROL\* old\_under\_pointer; // the old under pointer child control.  struct tagCONTROL\* primitive; // the primitive child of mouse event.  NOTIFPROC notif\_proc; // the notification callback procedure.  /\*window element data.\*/  struct \_wnd\_element\_data\* wed;    /\*The following members are only implemented for control.\*/  struct tagCONTROL\* next; // the next sibling control.  struct tagCONTROL\* prev; // the prev sibling control.  PCTRLCLASSINFO pcci; // pointer to Control Class Info struct.  MASKRECT \* mask\_rects;  //if a control is has WS\_EX\_CTRLASMAINWIN, this proc is the next control as main window  /\*  \* MainWindow->hFirstChildAsMainWin --->  \* control1->next\_ctrl\_as\_main ---->  \* control2->next\_ctrl\_as\_main ---->  \* .....  \*  \* Control have to list, one for browsers, the other for as main cotrol list:  \*  \* hFirstChild :  \* control1->next-->  \* control2->next-->  \* control3->next-->  \* ....  \*  \* hFirstChildAsMainWin:  \* control1->next\_ctrl\_as\_main -->  \* control2-> .....  \*  \* THE TWO LIST USE A SAME CONTROL OBJECT,  \* So, When Destroy a Control: YOU MUST REMOVE  \* ITSELF FROM PARENT'S hFirstChildAsMainWin'S LIST  \*  \*/  struct tagCONTROL\* next\_ctrl\_as\_main;  }CONTROL;  typedef CONTROL\* PCONTROL; |

|  |
| --- |
| 窗口Z序  Z序实际定义了窗口之间的层叠关系。Z序，实际是相对于屏幕坐标而言。一般而言，屏幕上的所有窗口均匀一个坐标系，即原点在左上角，X轴水平向右，Y轴垂直向下的坐标系。  Z序就是相对于一个假象的Z轴而言，这个Z轴从屏幕外指向屏幕内。窗口在这个Z轴上的值，就确定了其Z序。Z序值大的窗口，覆盖了Z序值小的窗口。  在程序中，Z序一般表示为一个链表。越接近于链表头结点，其Z序值越大。  在minigui中，维护了两个Z序。其中一个Z序永远在另外一个Z序之上，这样，就可以创建始终位于其他窗口之上的窗口，比如输入法窗口。如果在建立窗口时，指定了 WS\_EX\_TOPMOST扩展属性，就可以创建这样的窗口。  Z序操作实际就是链表操作。 |

|  |
| --- |
| 窗口剪切算法  有了窗口Z序，我们可以计算每个窗口的剪切域。  我么把因为窗口Z序产生的剪切域成为“全局剪切域”。这是相对于窗口自身定义的剪切域而言的，我们把后者称为“局部剪切域”。  窗口的所有输出，首先受到全局剪切域的影响，其次受到局部剪切域的影响。  全局剪切域的生成和维护  在minigui中，剪切域表示为若干互不相交的矩形的并集。这些矩形称为剪切矩形。  最初，屏幕上没有任何窗口时，桌面的剪切域由一个矩形组成，即屏幕矩形。  当屏幕上有一个窗口时，该窗口的剪切域由一个矩形组成，该矩形即为窗口在屏幕上的矩形，而桌面的剪切域却可能是由多个矩形组成的。下图为由四个矩形组成的桌面剪切域。  http://www.minigui.org/docs/techdoc/minigui-5/image01.jpg  如果窗口在桌面的位置变化，则桌面的剪切域就由两个矩形组成  http://www.minigui.org/docs/techdoc/minigui-5/image02.jpg  在只要一个窗口的情况下， 形成的桌面剪切域的矩形最多只能是四个。  如果新增一个窗口，则新的窗口将同时剪切旧的窗口和桌面，  http://www.minigui.org/docs/techdoc/minigui-5/image03.jpg  而这时，桌面和旧窗口的剪切域将多出一些矩形，这些矩形应该是原有剪切域中的每个矩形受到新窗口矩形影响之后生成的剪切矩形。同样，原有的剪切域中的每个矩形最多只能派生出4个新的剪切域，而某些矩形根本不受新的窗口矩形的影响。  这样，我们可以将某个窗口剪切域归纳为原有剪切域中排出某个矩形而生成的。   1. 窗口的全局剪切域初始化为窗口矩形。 2. 当窗口之上有其他窗口覆盖时，则该窗口的全局剪切域为排出新窗口矩形之后的剪切域。 3. 沿z序迭代第2步，直到最顶层窗口。 |

|  |
| --- |
| 剪切矩形结构体(gdi.h中)  typedef struct \_CLIPRECT  {  /\*\*  \* The clipping rectangle itself.  \*/  RECT rc;  /\*\*  \* The next clipping rectangle.  \*/  struct \_CLIPRECT\* next;  /\*\*  \* The previous clipping rectangle.  \*/  struct \_CLIPRECT\* prev;  } CLIPRECT;  剪切域结构体(gdi.h中)  typedef struct \_CLIPRGN  {  /\*\*  \* Type of the region, can be one of the following:  \* - NULLREGION\n  \* A null region.  \* - SIMPLEREGION\n  \* A simple region.  \* - COMPLEXREGION\n  \* A complex region.  \*/  BYTE type;  /\* \* Reserved for alignment.\*/  BYTE reserved[3];  /\*\*  \* The bounding rect of the region.  \*/  RECT rcBound;  /\*\*  \* Head of the clipping rectangle list.  \*/  PCLIPRECT head;//剪切矩形链表头  /\*\*  \* Tail of the clipping rectangle list.  \*/  PCLIPRECT tail;  /\*\*  \* The private block data heap used to allocate clipping rectangles.  \* \sa BLOCKHEAP  \*/  PBLOCKHEAP heap;  } CLIPRGN;  对剪切域的操作集中在region.c文件中。  与排除矩形相反的操作是包含某个矩形到剪切域中。这个操作作用于隐藏或者销毁某个窗口。  当一个窗口被隐藏或销毁时，该窗口之下的所有窗口将受到影响，此时，要将被隐藏或销毁的矩形包含到受影响的窗口的全局剪切域中。为此，Minigui的剪切域维护接口中有一个函数专用于该类操作(IncludeClipRect)。为确保剪切域中矩形互不相交，该函数首先计算与每个剪切矩形相交的矩形，然后将自己添加到该剪切域中。  但是，某些情况下，我们必须重新计算所有窗口的全局剪切域，比如移动某个窗口。  剪切矩形私有堆  在剪切域非常复杂，或者窗口非常多时，需要大量的矩形来表示每个窗口的全局剪切域。而在C程序中，如果频繁使用malloc和free申请和释放每个剪切矩形，将带来许多问题。   1. malloc和free是非常耗时的操作。 2. 频繁malloc和free将导致C程序堆的碎片化，从而可能导致将来的内存分配失败。为了避免频繁使用malloc和free。Minigui在初始化时，建立了一个私有堆。我们可以直接从这个堆中分配剪切矩形。而不需要从进程的全局堆中分配剪切矩形。这个私有堆实际是由一些空闲待用的剪切矩形组成的。每次分配时返回该链表的头结点，而释放时放进该链表的尾结点。如果链表为空，则利用malloc从进程的全局堆中分配剪切矩形。 |

|  |  |  |
| --- | --- | --- |
| The window element renderer structure.  关于渲染器，minigui定义了三种渲染器：classic风格，flat风格和skin风格，默认classic风格。  编译corelib 时，三种渲染器都会编译进去。  三种风格具体显示如下，以listview为例。 | | |
|  |  |  |
| const char name[LEN\_RENDERER\_NAME+1] | 渲染器名称 | #define LEN\_RENDERER\_NAME 15 |
| int (\*init) (PWERENDERER renderer) | 初始化渲染器的基本信息(metrics, 颜色，字体，图标等)和一些私有数据。 |  |
| int (\*deinit) (PWERENDERER renderer); | 释放渲染器资源 |  |
| DWORD (\*calc\_3dbox\_color) (DWORD color, int flag); | calculate lighter, lightest, darker and darkest color according to main color | Flag can be one of the values：  LFRDR\_3DBOX\_COLOR\_DARKER🡪darker color  LFRDR\_3DBOX\_COLOR\_DARKEST🡪darkest color  LFRDR\_3DBOX\_COLOR\_LIGHTER🡪lighter color  LFRDR\_3DBOX\_COLOR\_LIGHTEST🡪lightest color |
| void (\*draw\_3dbox) (HDC hdc, const RECT\* pRect, DWORD color, DWORD flag); | draw 3dbox | Flag can be OR'ed by the following values：  LFRDR\_BTN\_STATUS\_NORMAL🡪Normal status  LFRDR\_BTN\_STATUS\_HILITE🡪Highlight status  LFRDR\_BTN\_STATUS\_PRESSED🡪Pushed status  LFRDR\_BTN\_STATUS\_DISABLED🡪Disabled status  LFRDR\_BTN\_STATUS\_SELECTED🡪Selected status  LFRDR\_3DBOX\_THICKFRAME🡪Use thick/thin frame  LFRDR\_3DBOX\_FILLED🡪Filled rectangle |
| void (\*draw\_radio) (HDC hdc, const RECT\* pRect, DWORD color, int status); | draw radio | Status can be OR'ed by the following values:  LFRDR\_MARK\_ALL\_SELECTED🡪All selected status  LFRDR\_MARK\_HALF\_SELECTED🡪Half selected status  LFRDR\_MARK\_HAVESHELL🡪Have shell for mark |
| void (\*draw\_checkbox) (HDC hdc, const RECT\* pRect, DWORD color, int status); |  | Status sa draw\_radio |
| void (\*draw\_checkmark) (HDC hdc, const RECT\* pRect, DWORD color, int status); |  | Status sa draw\_radio |
| void (\*draw\_arrow) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color, int status); |  | Status can be OR'ed by the following values:  LFRDR\_ARROW\_LEFT🡪 Left arrow  LFRDR\_ARROW\_RIGHT🡪 Right arrow  LFRDR\_ARROW\_UP🡪 Up arrow  LFRDR\_ARROW\_DOWN-🡪 Down arrow  LFRDR\_ARROW\_NOFILL🡪 No filled arrow area  LFRDR\_ARROW\_HAVESHELL🡪 Have shell for arrow |
| void (\*draw\_fold) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color, int status, int next); |  | Status can be OR'ed by the following values:  LFRDR\_TREE\_FOLD🡪 Draw opening/closing folder  LFRDR\_TREE\_CHILD🡪 Draw connecting status between parent and child node  LFRDR\_TREE\_NEXT🡪 Draw connecting status between current and next node  Next The number of child node |
| void (\*draw\_focus\_frame) (HDC hdc, const RECT \*pRect, DWORD color) | draw focus frame |  |
| void (\*draw\_normal\_item) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color) | draw normal item function, this function used by listbox |  |
| void (\*draw\_hilite\_item) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color) | draw highlight item function, this function used by listbox |  |
| void (\*draw\_disabled\_item) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color); | draw disabled item function, this function used by listbox |  |
| void (\*draw\_significant\_item) (HWND hWnd, HDC hdc, const RECT\* pRect,  DWORD color); | aw significant item function, this function used by listbox |  |
| void (\*draw\_push\_button) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color1, DWORD color2, int status); |  | status, can be OR'ed by the following values:  BST\_NORMAL  BST\_HILITE  BST\_PUSHED  BST\_DISABLE  BST\_UNCHECKED  BST\_CHECKED  BST\_INDETERMINATE  BST\_FOCUS |
| void (\*draw\_radio\_button) (HWND hWnd, HDC hdc, const RECT\* pRect, int status); |  | status, can be OR'ed by the following values:  LFRDR\_BTN\_STATUS\_NORMAL🡪Normal status  LFRDR\_BTN\_STATUS\_HILITE🡪Highlight status  LFRDR\_BTN\_STATUS\_PRESSED🡪Pushed status  LFRDR\_BTN\_STATUS\_DISABLED🡪 Disabled status  LFRDR\_BTN\_STATUS\_SELECTED🡪 Selected status |
| void (\*draw\_check\_button) (HWND hWnd, HDC hdc, const RECT\* pRect, int status); |  | status, can be OR'ed by the following values:  LFRDR\_BTN\_STATUS\_NORMAL🡪 Normal status  LFRDR\_BTN\_STATUS\_HILITE🡪 Highlight status  LFRDR\_BTN\_STATUS\_PRESSED🡪 Pushed status  LFRDR\_BTN\_STATUS\_DISABLED🡪 Disabled status  LFRDR\_BTN\_STATUS\_SELECTED🡪 Selected status |
| void (\*draw\_border) (HWND hWnd, HDC hdc, BOOL is\_active); | draw window border |  |
| void (\*draw\_caption) (HWND hWnd, HDC hdc, BOOL is\_active); | draw window text and icon function |  |
| void (\*draw\_caption\_button) (HWND hwnd, HDC hdc, int ht\_code, int state); |  | ht\_code : the number for close, max, or min button, 0 for all.  state The button state, can be one of the values:  LFRDR\_BTN\_STATUS\_NORMAL🡪 Normal status  LFRDR\_BTN\_STATUS\_HILITE🡪 Highlight status  LFRDR\_BTN\_STATUS\_PRESSED🡪 Pushed status  LFRDR\_BTN\_STATUS\_DISABLED🡪 Disabled status |
| void (\*draw\_scrollbar) (HWND hWnd, HDC hdc, int sb\_pos); |  | sb\_pos : the pos need to draw.  status, which is one of the value:  LFRDR\_BTN\_STATUS\_NORMAL🡪 Normal status  LFRDR\_BTN\_STATUS\_HILITE🡪 Highlight status  LFRDR\_BTN\_STATUS\_PRESSED🡪 Pushed status  LFRDR\_BTN\_STATUS\_DISABLED🡪 Disabled status |
| void (\*calc\_trackbar\_rect) (HWND hWnd, LFRDR\_TRACKBARINFO \*info, DWORD dwStyle, const RECT\* rcClient, RECT\* rcRuler, RECT\* rcBar, RECT\* rcBorder); |  |  |
| void (\*draw\_trackbar) (HWND hWnd, HDC hdc, LFRDR\_TRACKBARINFO \*info) |  |  |
| int (\*calc\_we\_area) (HWND hWnd, int which, RECT\* we\_area) | calculate window element area | which The hit test code of the part to be calculated.  we\_area The reactangle for calculating result |
| int (\*calc\_we\_metrics) (HWND hWnd, LFRDR\_WINSTYLEINFO\* style\_info, int which); | calculate window element metrics | which, can be one of the following values:  LFRDR\_METRICS\_BORDER🡪 The metrics of window border  LFRDR\_METRICS\_CAPTION\_H🡪 The height of window caption  LFRDR\_METRICS\_MENU\_H🡪 The height of window menu  LFRDR\_METRICS\_VSCROLL\_W🡪 The width of vertical scrollbar  LFRDR\_METRICS\_HSCROLL\_H🡪 The height of horizontal scrollbar  LFRDR\_METRICS\_ICON\_H🡪 The height of window icon.  LFRDR\_METRICS\_ICON\_W🡪 The width of window icon  LFRDR\_METRICS\_MINWIN\_WIDTH🡪 The minimize width of window  LFRDR\_METRICS\_MINWIN\_HEIGHT🡪 The minimize height of window |
| int (\*hit\_test) (HWND hWnd, int x, int y) |  | x, y The position of the mouse in screen coordinates  return HT\_XXX or HT\_USER\_XXX, hittest value |
| int (\*on\_click\_hotspot) (HWND hWnd, int which) |  | user-defined hotspot function |
| void (\*draw\_custom\_hotspot) (HWND hWnd, HDC hdc, int ht\_code, int state) |  | ht\_code The hittest code. If 0, it should draw all user-defined hotspot.  state The button state, can be one of the values:  LFRDR\_BTN\_STATUS\_NORMAL  LFRDR\_BTN\_STATUS\_HILITE  LFRDR\_BTN\_STATUS\_PRESSED |
| void (\*calc\_thumb\_area) (HWND hWnd, BOOL vertical, LFSCROLLBARINFO\* sb\_info); | calculate scrollbar's thumb area function | vertical Vertical scrollbar or Horizontal scrollbar.  sb\_info The scrollbar information for calculating result. |
| void (\*disabled\_text\_out) (HWND hWnd, HDC hdc, const char\* spText, PRECT rc, DWORD dt\_fmt) | output disabled text |  |
| void (\*draw\_tab) (HWND hWnd, HDC hdc, RECT \*rect, char \*title, DWORD color, int flag, HICON icon); | draw propsheet's tab | flag, can be OR'ed by the following values:  LFRDR\_TAB\_BOTTOM🡪 Use bottom tab  LFRDR\_TAB\_ACTIVE🡪 Active tab  LFRDR\_TAB\_ICON🡪 With icon |
| void (\*draw\_progress) (HWND hWnd, HDC hdc, int nMax, int nMin, int nPos, BOOL fVertical) | draw progressbar's chunk/bar |  |
| void (\*draw\_header) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color) | draw table header | this function used by listview and gridview. |
| DWORD (\*on\_get\_rdr\_attr) (struct \_WINDOW\_ELEMENT\_RENDERER\*, int we\_attr\_id); | get renderer private data | we\_attr\_id The id pointed renderer private data |
| DWORD (\*on\_get\_rdr\_attr) (struct \_WINDOW\_ELEMENT\_RENDERER\*, int we\_attr\_id) | set renderer private data | we\_attr\_id The id pointed renderer private data.  we\_attr The new renderer private data |
| void (\*erase\_background) (HWND hWnd, HDC hdc, const RECT \*rect) |  |  |
| void (\*draw\_normal\_menu\_item) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color) | draw menu normal item |  |
| void (\*draw\_hilite\_menu\_item) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color); | draw menu highlight item |  |
| void (\*draw\_disabled\_menu\_item) (HWND hWnd, HDC hdc, const RECT\* pRect, DWORD color) | draw menu disabled item |  |
| int we\_metrics [WE\_METRICS\_NUMBER] | window element metrics information. | #define WE\_METRICS\_NUMBER 6 |
| DWORD we\_colors [WE\_COLORS\_NUMBER][3] | window element color information | #define WE\_COLORS\_NUMBER 14 |
| PLOGFONT we\_fonts [WE\_FONTS\_NUMBER] | window element font information | #define WE\_FONTS\_NUMBER 4 |
| HICON we\_icon [2][SYSICO\_ITEM\_NUMBER] | renderer icon information | #define SYSICO\_ITEM\_NUMBER 5 |
| unsigned int refcount; | reference count |  |
| const void\* private\_info | private information |  |

|  |
| --- |
| 窗口信息结构体 |
| /\*\*  \* The window information structure.  \*/  typedef struct \_WINDOWINFO  {  /\*\* The position and size of window.\*/  int left, top;  int right, bottom;  /\*\* The position and size of client area.\*/  int cl, ct;  int cr, cb;  /\*\* The styles of window.\*/  DWORD dwStyle;  /\*\* The extended styles of window.\*/  DWORD dwExStyle;  /\*\* The background color.\*/  int iBkColor;  /\*\* The handle of menu.\*/  HMENU hMenu;  /\*\* The handle of accelerator table.\*/  HACCEL hAccel;  /\*\* The handle of cursor.\*/  HCURSOR hCursor;  /\*\* The handle of icon.\*/  HICON hIcon;  /\*\* The handle of system menu.\*/  HMENU hSysMenu;  /\*\* The pointer to logical font.\*/  PLOGFONT pLogFont;  /\*\* The caption of window.\*/  char\* spCaption;  /\*\* The identifier of window.\*/  int id;  /\*\* The vertical scrollbar information.\*/  LFSCROLLBARINFO vscroll;  /\*\* The horizital scrollbar information.\*/  LFSCROLLBARINFO hscroll;  /\*\* The window renderer. \*/  WINDOW\_ELEMENT\_RENDERER\* we\_rdr;  } WINDOWINFO; |

|  |
| --- |
|  |
| #define MSG\_ADDNEWMAINWIN 0x00F0  CreateMainWindow(&CreateInfo)  🡪SendMessage (HWND\_DESKTOP, MSG\_ADDNEWMAINWIN, (WPARAM) pWin, 0)//添加新窗口到z序  🡪DesktopWinProc (HWND hWnd, int message, WPARAM wParam, LPARAM lParam)//桌面消息处理函数  (message >= MSG\_FIRSTWINDOWMSG && message <= MSG\_LASTWINDOWMSG)  🡪static int dskWindowMessageHandler (int message, PMAINWIN pWin, LPARAM lParam)  case MSG\_ADDNEWMAINWIN  🡪static int dskAddNewMainWindow (PMAINWIN pWin)//add the new window to the z-order list  🡪 SendAsyncMessage ((HWND)pWin, MSG\_NCPAINT, 0, 0)//绘制非客户区(即标题，scroll等)  **#define** MSG\_NCPAINT 0x00B2  向下↓走到默认窗口处理函数  🡪int DefaultWindowProc (HWND hWnd, int message, WPARAM wParam, LPARAM lParam)  If (IsMainWindow(hWnd))  🡪int PreDefMainWinProc (HWND hWnd, int message, WPARAM wParam, LPARAM lParam)  message >= MSG\_FIRSTPAINTMSG && message <= MSG\_LASTPAINTMSG  🡪static int DefaultPaintMsgHandler(PMAINWIN pWin, int message,WPARAM wParam, LPARAM lParam)  🡪static void wndDrawNCFrame(MAINWIN\* pWin, HDC hdc, const RECT\* prcInvalid)  🡪 draw\_caption ((HWND)pWin, hdc, is\_active)//标题  🡪 wndDrawNCButtonEx (pWin, hdc, HT\_HSCROLL, 0);//水平scroll  🡪 wndDrawNCButtonEx (pWin, hdc, HT\_VSCROLL, 0); //垂直scroll  🡪 draw\_border ((HWND)pWin, hdc, is\_active)//边框  🡪 DrawMenuBarHelper (pWin, hdc, prcInvalid); |

|  |
| --- |
| typedef struct \_ZORDERNODE------Z序结构(zorder.h)  {  DWORD flags;  char caption[MAX\_CAPTION\_LEN+1];  RECT rc;  RECT dirty\_rc;  unsigned int age;  int cli; /\* which client? \*/  HWND fortestinghwnd; /\* which window of the client? \*/  HWND main\_win; /\* handle to the main window \*/  int next;  int prev;  /\*The first position of mask rect.\*/  int idx\_mask\_rect;  } ZORDERNODE;  全局变量\_\_mg\_zorder\_info 在初始化的时候分配内存(kernel\_alloc\_z\_order\_info)  typedef struct \_ZORDERINFO------Z序信息结构(zorder.h)  {  int size\_usage\_bmp;  int max\_nr\_popupmenus;  int max\_nr\_globals;  int max\_nr\_topmosts;  int max\_nr\_normals;  int nr\_popupmenus;  int nr\_globals;  int nr\_topmosts;  int nr\_normals;  int first\_global;  int first\_topmost;  int first\_normal;  int active\_win;  int cli\_trackmenu;  HWND ptmi\_in\_cli;  int zi\_semid;  int zi\_semnum;  /\* The usage bitmap for mask rect. \*/  int size\_maskrect\_usage\_bmp;  #ifdef \_MGRM\_THREADS  #ifndef \_\_NOUNIX\_\_  pthread\_rwlock\_t rwlock;  #else  pthread\_mutex\_t rwlock;  #endif  #endif  } ZORDERINFO; |

|  |
| --- |
| 鼠标和键盘消息上报流程  InitGUI  🡪SystemThreads()//创建了3个线程  🡪pthread\_create (&\_\_mg\_desktop, NULL, DesktopMain, &wait)//desktop线程  🡪\_\_mg\_timer\_init ()//定时线程  🡪pthread\_create (&\_\_mg\_parsor, NULL, EventLoop, &wait)//事件线程  🡪while (\_\_mg\_quiting\_stage > \_MG\_QUITING\_STAGE\_EVENT) {  event = IAL\_WaitEvent (IAL\_MOUSEEVENT | IAL\_KEYEVENT, 0,  NULL, NULL, NULL, (void\*)&\_\_mg\_event\_timeout);//等待事件上报  🡪kernel\_GetLWEvent (IAL\_MOUSEEVENT, &lwe)//转换鼠标消息  🡪ParseEvent (&lwe)//将消息放入消息队列🡪QueueDeskMessage (&Msg)  向下↓走到desktop的窗口处理函数  DesktopWinProc (HWND\_DESKTOP, Msg.message, Msg.wParam, Msg.lParam)  🡪 message >= MSG\_FIRSTMOUSEMSG && message <= MSG\_LASTMOUSEMSG//鼠标消息  🡪 MouseMessageHandler (message, flags, x, y)//鼠标消息处理函数   1. 转发给捕获屏幕的窗口   PostMessage (\_\_mg\_capture\_wnd, message, flags | KS\_CAPTURED, MAKELONG (x, y))   1. 查找鼠标点所在的窗口，按照Z序找到最上层的窗口。   gui\_GetMainWindowPtrUnderPoint (x, y)   1. 该窗口注册HOOK单独处理   dskHandleMouseHooks ((HWND)pCtrlPtrIn, message, flags, MAKELONG (x, y))   1. 该窗口是控件类型就获取控件所在的主窗口，此处就是应用层我们自己创建的窗口 2. 该窗口被设置为topmost顶层窗口，设置为MOUSEACTIVE   dskMoveToTopMost (pUnderPointer, RCTM\_CLICK, MAKELONG (x, y))  SendNotifyMessage ((HWND) pUnderPointer, MSG\_MOUSEACTIVE, 0, 0)   1. 转换消息：MSG\_DT\_LBUTTONDOWN 🡪 (MSG\_LBUTTONDOWN + MSG\_DT\_MOUSEOFF) 2. PostMessage(MSG\_DT\_LBUTTONDOWN)到应用窗口。 3. 应用窗口无法处理内部消息(MSG\_DT\_LBUTTONDOWN)，漏到默认窗口处理函数中继续处理 4. DefaultMainWinProc() -->  PreDefMainWinProc()   #define DefaultMainWinProc \_\_mg\_def\_proc[0]  #define DefaultDialogProc \_\_mg\_def\_proc[1]  #define DefaultControlProc \_\_mg\_def\_proc[2]   1. PreDefMainWinProc() 🡪 DefaultDTMouseMsgHandler() 2. DefaultDTMouseMsgHandler函数转化消息为   (MSG\_LBUTTONDOWN + MSG\_DT\_MOUSEOFF) 🡪MSG\_DT\_LBUTTONDOWN   1. 再次PostMessage(MSG\_LBUTTONDOWN)消息到应用窗口，这时窗口可以收到MSG\_LBUTTONDOWN消息。 2. 对于控件的鼠标消息，应用窗口的过程处理函数走到窗口默认处理函数   int PreDefMainWinProc (HWND hWnd, int message, WPARAM wParam, LPARAM lParam)  🡪 message >= MSG\_FIRSTMOUSEMSG && message <= MSG\_NCMOUSEOFF  🡪 DefaultMouseMsgHandler(pWin, message, wParam, LOSWORD (lParam), HISWORD  (lParam))  🡪查找鼠标所在控件  pUnderPointer = wndMouseInWhichControl (pWin, x, y, &UndHitCode)  🡪清之前的焦点，PostMessage鼠标消息到对应控件 |