# 第五部分 附 录

## 附录A 控件窗口风格

使用MFC和Windows API创建控件窗口有多种方法：可以使用对话框编辑器增加一个控件到对话框模板中，对话框模板在对话框被创建时由 Windows API创建；或者可以使用 MFC控件类，如CButton类，自己动手创建控件窗口。若 MFC没有在类中捆绑一个通用控件，仍然可以使用CWnd类和通用控件窗口类名创建控件。

每个控件装入时显示不同的可视特征，只要改变窗口风格便可访问这些特征。要想看到所有可用风格，唯一途径是用对话框编辑器作实验，即使对话框编辑器也不可能支持所有的窗口风格。

本附录描述了可用的每个控件的重要窗口风格。尤其那些影响控件外观的窗口风格将与相应的图一起列出来。

### A.1 Windows 3.1及以上版本提供的控件窗口

#### A.1.1 按钮控件

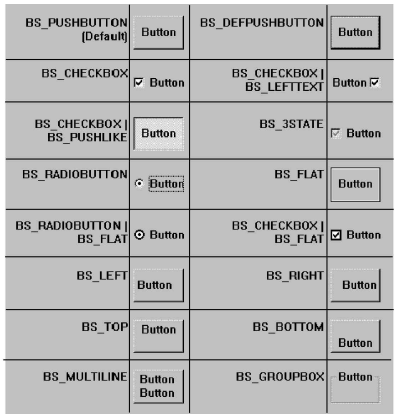
**用Windows API创建按钮控件**

```
HWND CreateWindowEx(dwExStyle, "BUTTON", "Text",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建按钮控件**

```
CButton m_button;       // usually embedded in parent class
m_button.Create(
      "Text", WS_VISIBLE|WS_CHILD|dwStyle, rect,
pParentWnd, id);
```

**可视风格**(见图A-1)




图A-1 按钮控件风格

**其他风格**

| | |
|---|---|
| BS_DEFPUSHBUTTON | 当用户按下回车键，拥有这种风格的键被选中。然而，只有当其父窗口(如对话框)具有输入焦点时才会发生。在父窗口中，一次只能有一个按钮有该风格 |
| BS_AUTOCHECKBOX | 当用户单击它时，自动地在被选中和不被选中之间改变状态 |
| BS_AUTO3STATE | 当用户单击它时，自动地在被选中、不被选中和不确定三种状态之间变化 |
| BS_AUTORADIOBUTTON | 自动取消选中单选按钮组中地其他按钮 |
| BS_OWNERDRAW | 父窗口必须绘制该按钮 |
| BS_ICON | 在按钮表面绘制一个图标，该图标必须在其他步骤中指定 |
| BS_BITMAP | 在按钮表面绘制一个位图，该位图必须在其他步骤中指定 |

**说明**

- 注意组合框实际上是一个按钮控件，它被告知绘制一个框，并把它的名字放在左上角，然后，忽略任何输入。这种在同一控件中非近似特征的堆积，可能是由于当时只有七个通用控件而没有增添的余地。
- 其他按钮效果，通过使用A.4节“普通窗口风格”的边框风格获得。

## A.1.2 静态文本控件

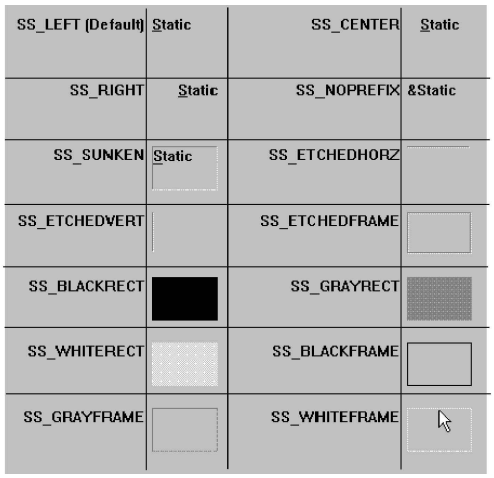
**使用Windows API创建静态文本控件(Static Control)**

```
HWND CreateWindowEx(dwExStyle, "STATIC", "Text",
    WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
    hWndParent, (HMENU) id, hInstance, NULL);
```

**使用MFC创建静态文本控件**

```
CStatic m_static;          // usually embedded in parent class
m_static.Create(
    "&Text", WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-2)




图A-2 静态文本控件风格

**其他风格**

| | |
|---|---|
| SS_OWNERDRAW | 父窗口绘制控件 |
| SS_NOTIFY | 正常情况下，静态文本控件把任何鼠标单击直接传给父窗口<br>该风格强迫控件自己处理这些消息 |
| SS_ICON | 把图标绘制在控件上，图标在另一步中设置 |
| SS_BITMAP | 把位图绘制在控件上，位图在另一步中设置 |
| SS_ENHMETAFILE | 把一个加强的元文件绘制在控件上，该元文件在另一步中设置 |
| SS_CENTERIMAGE | 若静态文本控件显示的是图标、位图或元文件，该风格强迫其位于控件中心 |

**说明**

- 若用户在静态文本中某一字母前指定了&字符，该字符将显示一下划线。当用户按下该字母，Tab次序紧随其后的控件将获得输入焦点。Tab次序是对话资源中的控件实体的次序。
- 拥有SS_ETCHEDVERT或SS_ETCHEDHORZ风格的静态文本控件，为用户绘制一条简单的蚀刻线。怎样使用对话框编辑器实现呢？应该选择哪个控件呢？似乎任何控件都不符合，其实，例69提供了一种方法。文本等同效果(文本字符看起来像蚀刻进窗口中)使用CDC::DrawState ( )和禁用选项可以获得。DrawState ( )用来画禁用菜单项。

## A.1.3 编辑控件

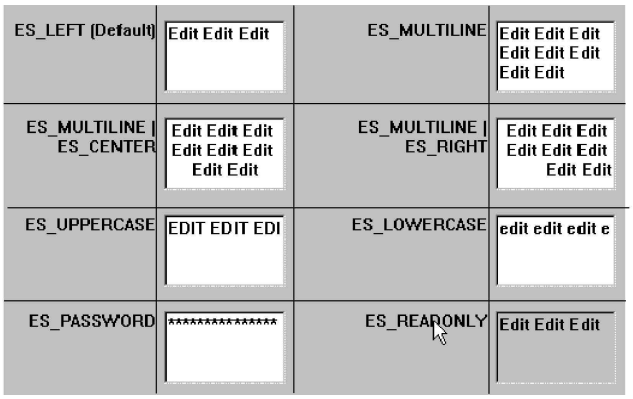
**用Windows API创建编辑控件(Edit Control)**

```
HWND CreateWindowEx(dwExStyle, "EDIT", "Text",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建编辑控件**

```
CEdit m_edit;              // usually embedded in parent class
m_edit.CreateEx(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-3)




图A-3 编辑控件风格

**其他风格**

| | |
|---|---|
| ES_AUTOVSCROLL | 当新文本输入时，引起文本自动向左滚动 |
| ES_AUTOHSCROLL | 当新文本输入时，引起文本向上滚动。只有当控件拥有 ES_MULTILINE风格时才有效 |
| ES_NOHIDESEL | 即使控件失去输入焦点，使被选中的文本仍然处于被选中状态 |
| ES_WANTRETURN | 对于一个多行控件，允许回车键经过控件，而不是像单击父窗口中的缺省按钮一样 |

**说明**

- 这里显示的编辑控件有一个附加的边框，在默认情况下，编辑框窗口没有边框。
- 编辑框中的所有文本具有相同的字体 (可以用SetFont ( )设置)。若想使某些文本加下划线或使某些文本显示黑体，应该用 Rich Edit Control。

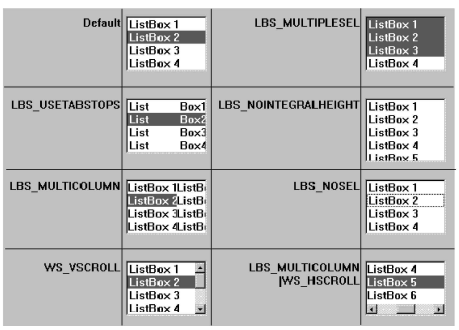
### A.1.4 列表框控件

**用Windows API创建一个列表框控件** (List Box Control)

```
HWND CreateWindowEx(dwExStyle, "LISTBOX", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建列表框控件**

```
CListBox m_listbox;        // usually embedded in parent class
m_listbox.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```




图A-4 列表框控件风格

**可视风格(见图A-4)**

**初始化控件**

■ 重置列表框内容，用

```
m_listbox.ResetContent ( ) ;
```

■ 添加一个字符串，用

```
m_listbox.AddString ("ListBox1") ;
```

■ 当设置LBS_MULTICOLUMN风格时，要设每列的宽度，用

```
m_listbox.SetColumnWidth (60) ;
```

■ 当设置LBS_USETABSTOPS时，要设置制表位，用

```
int tabs[NUM]={10,20,30};
m_listbox.SetTabStops(NUM,tabs);
```

**其他风格**

| | |
|---|---|
| LBS_NOTIFY | 使列表框控件处理鼠标单击事件，而不把它们传递到父窗口 |
| LBS_SORT | 字符串按字母顺序排序 |
| LBS_OWNERDRAWFIXED | 父窗口必须绘制该控件，并且每项的垂直大小相同 |
| LBS_OWNERDRAWVARIABLE | 父窗口必须绘制该控件，并且每项的垂直大小可变 |
| LBS_HASSTRINGS | 对于自绘制的控件，该控件在内部维护字符串，以便还能设置或获得控件字符串 |
| LBS_EXTENDEDSEL | 同LBS_MULTIPLESEL一样，一次可以选择多项，但是用LBS_EXTENDEDSEL，用户必须按下Shift和Control键进行多项选择；用LBS_MULTIPLESEL，用户只需单击一个项 |

**说明**

■ WS_HSCROLL风格只有在设置 LBS_MULTICOLUMN风格时才起作用，但这时WS_VSCROLL不起作用。

■ 通常，一个列表框重载你为它设置的大小，以便能避免控件底部部分地遮蔽列表的最后一行；要关掉该功能，使用LBS_NOINTERGRALHEIGHT风格。

■ 这里所显示的列表框有一个附加的边框；在默认设置下，列表框没有边框。

## A.1.5 滚动条

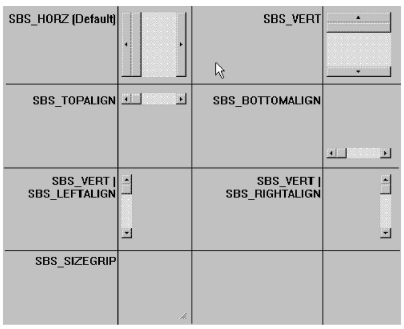
**用Windows API创建滚动条**

```
HWND CreateWindowEx(dwExStyle, "SCROLLBAR", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建滚动条**

```
CScrollBar m_scrollbar;    // usually embedded in parent class
m_scrollbar.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格(见图A-5)**




图A-5 滚动条控件风格

**说明**

■ 与Windows在任何一个窗口的非客户区绘制滚动条不同，事实上，这些滚动条自身是

一些完整的子窗口。一个父窗口可以有它自己的滚动条和任意数量的滚动条控件窗口；但是，非客户区滚动条和控件窗口滚动条都用 WM_HSCROLL和WM_VSCROLL窗口消息回传到父窗口；要在父窗口中区别它们，需要在消息的 lParam中查看滚动条句柄，当为该消息添加一个处理函数时， MFC把该滚动条句柄封装到一个CScrollBar类中。

- 当单独设置SBS_HORZ和SBS_VERT时，滚动条的宽度和高度可以是任意值；当单独设置SBS_TOPALIGN，SBS_BOTTOMALIGN等，滚动条设置一个标准宽度，这可以在图A-5中看见。
- 创建具有SBS_SIZEGRIP风格的滚动条控件，允许用户拖动控件所在的窗口改变其大小，而不管控件位于窗口的什么位置。

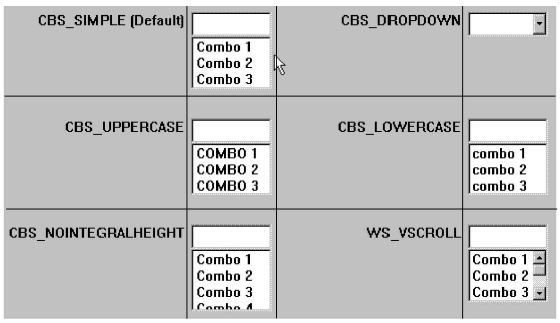
### A.1.6 组合框

**用Windows API创建组合框**

```
HWND CreateWindowEx(dwExStyle, "COMBOBOX", "",
    WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
    hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建组合框**

```
CComboBox m_combobox;    // usually embedded in parent class
m_combobox.Create(
  WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```




图A-6 组合框控件风格

**可视风格**(见图A-6)

**初始化控件**

添加一个字符串到一个组合框，所用语句如下：

```
m_combobox.AddString("Combo 1");
```

**其他风格：**

| | |
|---|---|
| CBS_OWNERDRAWFIXED | 父窗口绘制该控件，并且每项具有相同高度 |
| CBS_OWNERDRAWVARIABLE | 父窗口绘制该控件，但每项具有不同的高度 |
| CBS_HASSTRINGS | 即使该控件是由父窗口绘制的，但它仍然为每个项保持一个字符串 |
| CBS_AUTOHSCROLL | 如果编辑框有效，当输入文本超出窗口边界时，输入文本自动向左滚动 |
| CBS_SORT | 列表按字母顺序排列 |

**说明**

组合框实际上控制两个完全属于它自己的控件窗口：一个编辑框的编辑控件和一个列表框的ComboLBox控件。因为一些窗口消息，如鼠标单击，只传递到这两个控件之一，而不进入组合框，因此，有时需要直接子分类它们，这只要通过搜索该组合框之下的子窗口即可。然而，如果使用下拉风格，列表框控件只有在下拉的情况下才可用。

## A.2 Windows 95/NT及以上版本提供的控件窗口

### A.2.1 多信息编辑框

**用Windows API创建多信息编辑控件**

```
HWND CreateWindowEx(dwExStyle, "RICHEDIT", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建多信息编辑框**

```
CRichEditCtrl m_richeditctrl;          // usually embedded in
                                       // parent class
m_richeditctrl.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**说明**

多信息编辑框使用大部分与编辑框相同的窗口风格；它们之间的主要不同在于多信息编辑框可以处理多信息文本格式文件，允许文本一行中有多种颜色、风格和字体大小。然而，要使用该功能，不能用SetWindowText ( )或GetWindowText ( )输入文本到窗口中，而需要用成员函数 StreamIn ( )和StreamOut ( )。事实上，还是可以使用 SetWindowText ( )和GetWindowText ( )——但是，RTF格式用来改变文本格式的任何文本顺序将会变得可见。

### A.2.2 列表控件

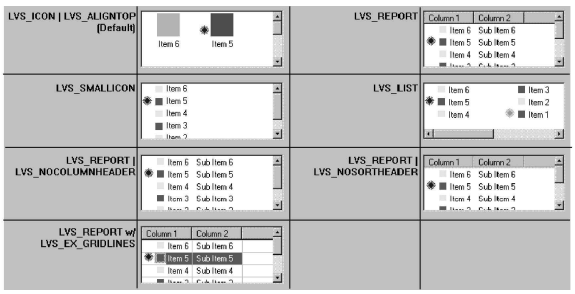
**用Windows API创建列表控件**

HWND CreateWindowEx(dwExStyle, "SysListView32", "",
WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
hWndParent, (HMENU) id, hInstance, NULL);

**用MFC创建列表控件**

CListCtrl m_listctrl; // usually embedded in parent class

```
m_listctrl.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```




图A-7 列表控件风格

**可视风格**(见图A-7)

**初始化控件**

■ 添加列到一个列表控件，用：

```
m_listctrl.InsertColumn(0,"Column 1",LVCFMT_LEFT,width,0);
m_listctrl.InsertColumn(1,"Column 1",LVCFMT_RIGHT,width,1);
```

■ 添加列表控件可用的图像，用：

```
m_listctrl.SetImageList(&m_imageLarge, LVSIL_NORMAL);
m_listctrl.SetImageList(&m_imageSmall, LVSIL_SMALL);
m_listctrl.SetImageList(&m_imageState, LVSIL_STATE);
```

■ 添加新的一行，用：

```
LV_ITEM lvi;
lvi.mask = LVIF_TEXT | LVIF_IMAGE;
lvi.iItem = 0;
lvi.iSubItem = 0;
lvi.pszText = "Item 1";
lvi.iImage = 1;
int inx=m_listctrl.InsertItem(&lvi);
```

■ 添加一项到一列中，用：

```
m_listctrl.SetItemText(inx,1，"Sub Item 1");
```

■ 改变一行的状态图像，用：

```
m_listctrl.SetItemState(inx, INDEXTOSTATEIMAGEMASK(1),
   LVIS_STATEIMAGEMASK);
```

■ 要设置扩展风格，如LVS_EX_GRIDLINES，用：

```
m_listctrl.SendMessage(LVM_SETEXTENDEDLISTVIEWSTYLE,
   0,LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT);
```

**其他风格**

| | |
|---|---|
| LVS_SINGLESEL | 一次只允许选中一项 |
| LVS_SHOWSELALWAYS | 即使该窗口失去焦点，也使选项保持选中状态 |
| LVS_SORTASCENDING | 按升序排列列表项 |
| LVS_SORTDESCENDING | 按降序排列列表项 |
| LVS_EDITLABELS | 允许列表项在适当位置被编辑；当用户做完编辑时，父窗口将会收到一个LVN_ENDLABELEDIT消息 |
| LVS_NOSCROLL | 禁用滚动 |
| LVS_OWNERDRAWFIXED | 父窗口负责绘制一项 |

**说明**

■ 列宽以像素为单位，一种将字符宽转换为像素的快速方法是：

```
CDC* dc = GetDC();

TEXTMETRIC tm;
dc->GetTextMetrics(&tm);
ReleaseDC(dc);
```

这时，当指定列宽时，可以用 *tm.tmAveCharWidth字符数。

■ 指定为控件使用的图像列表必须在位，否则将不显示任何图像，但控件还是能像没发生任何错误一样工作。保持图像列表可用的最佳方法是把图像列表插入到该控件的父窗口类中。

```
class CDialog
{
   :  :  :
CImageList m_imagelist;
};
```

### A.2.3 扩展组合框

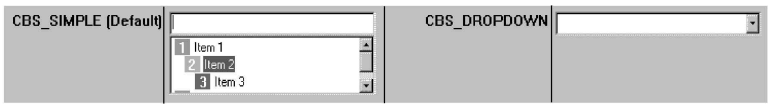
**用Windows API创建一个扩展组合框**

```
HWND CreateWindowEx(dwExStyle, "ComboBoxEx32", "Text",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建扩展组合框**

```
CComboBoxEx m_comboex;    // usually embedded in parent class
m_comboex.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-8)




图A-8 扩展组合框控件风格

**初始化控件**

■ 添加控件可用的图像，用：

```
m_comboex.SetImageList(&m_imageList);
```

■ 添加项到控件，用：

```
COMBOBOXEXITEM cbei;
cbei.mask = CBEIF_INDENT | CBEIF_TEXT |CBEIF_IMAGE;
cbei.iItem              = 0;
cbei.pszText            = pszItem1;
cbei.cchTextMax         = sizeof(pszItem1);
cbei.iImage             = 0;            //IMAGE TO DISPLAY
cbei.iSelectedImage = 1;            //IMAGE TO DISPLAY WHEN SELECTED
cbei.iIndent            = 0;            //# OF PIXELS TO INDENT
m_comboex.InsertItem(&cbei);
```

**说明**

■ 扩展组合框使用大部分与一般组合框相同的风格，但可以在它的项的边上指定一个图像，并且项可以缩进一个像素。

■ 扩展组合框相对较新，一些 MFC版还不能支持；然而，还是可以用 Windows API调用访问它，并用窗口消息与它通信。

■ 应负责维护图像列表对象，即使把图像列表对象放入该控件。如果不这样做，图像将不显示。但不会从MFC或Windows发送错误消息。

## A.2.4 动画控件

**用Windows API创建动画控件**

```
HWND CreateWindowEx(dwExStyle, "SysAnimate32", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建动画控件**

```
CAnimateCtrl m_animate;    // usually embedded in parent class
m_animate.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**

| | |
|---|---|
| ACS_CENTER | 把.avi图像置于控件中心 |
| ACS_TRANSPARENT | 绘制没有背景色的控件，使被绘制的控件直接显示在父窗口中 |
| ACS_AUTOPLAY | 打开.avi文件时，自动开始播放 |
| ACS_TIMER | 通常，.avi文件被另一个线程回放；该风格使回放不需要线程 |

**初始化控件**

■ 打开并播放一个.avi文件，用：

```
m_animate.Open("filename");
m_animate.Play(0,        // from frame (0=start)
   -1,                   // to frame (-1=end)
   1);                   // number of replays (-1=forever)
```

■ 停止并关闭一个.avi文件，用：

```
m_animate.Stop();
m_animate.Close();
```

**说明**

■ 参见例9和例57。

## A.2.5 滑块控件

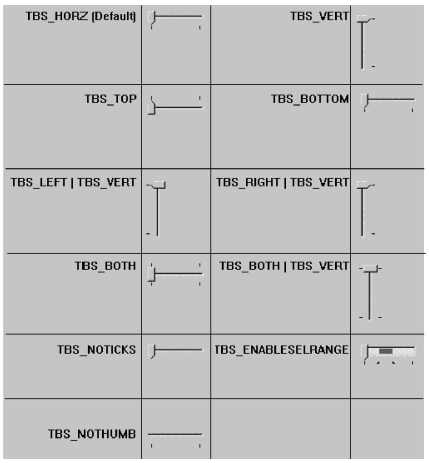
**用Windows API创建滑块控件**

```
HWND CreateWindowEx(dwExStyle, "msctls_ trackbar32", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建滑块控件**

```
CSliderCtrl m_slider;      // usually embedded in parent class
m_slider.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-9)




图A-9 滑块控件风格

**其他风格**

| | |
|---|---|
| TBS_AUTOTICKS | 为控件范围内的每一个增量都创建刻度线 |
| TBS_TOOLTIPS | 当鼠标压住该控件时，弹出一个弹出窗口，显示滑块的当前位置 |

**说明**

滑块(跟踪条)控件使用的WM_VSCROLL和WM_HSCROLL消息回传到父窗口，这与滚动条控件窗口相同。OnHScroll ( )和OnVScroll ( )消息处理函数把该控件封装到CScrollBar类中，但它真正封装的是一个滑块控件，还需使该类嵌入CSliderCtrl类中。

## A.2.6 树式视图控件

**用Windows API创建树式视图控件**

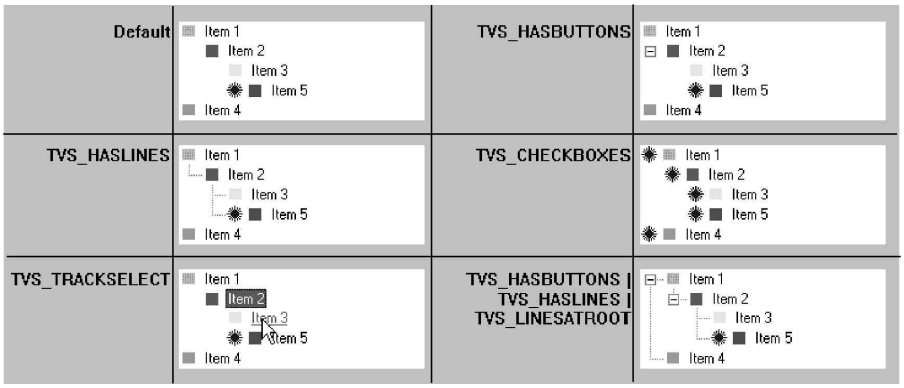
```
HWND CreateWindowEx(dwExStyle, "SysTreeView32", "Text",
```

```
WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height, hWndParent,
(HMENU) id, hInstance, NULL);
```

**用MFC创建树式视图控件**

```
CTreeCtrl m_treectrl;    // usually embedded in parent class
m_treectrl.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-10)




图A-10 树式控件风格

**初始化控件**

■ 要添加一个可以在该控件中显示的图像列表 (图A-10中看到的方块是LVSIL_NORMAL图像，星形是LVSIL_STATE图像)，用：

```
m_treectrl.SetImageList(&m_imageSmall, LVSIL_NORMAL);
m_treectrl.SetImageList(&m_imageState, LVSIL_STATE);
```

■ 添加一项到列表中，用：

```
TV_INSERTSTRUCT    tvi;
tvi.item.mask = TVIF_IMAGE | TVIF_TEXT |TVIF_SELECTEDIMAGE;
tvi.hParent = TVI_ROOT;
tvi.hInsertAfter = TVI_LAST;
tvi.item.iImage = tvi.item.iSelectedImage =0;
tvi.item.pszText = pszItem1;
tvi.item.stateMask = TVIS_STATEIMAGEMASK;
tvi.item.state = INDEXTOSTATEIMAGEMASK(1);
HTREEITEM hTreeRoot = m_treectrl.InsertItem(&tvi);
```

**其他风格**

| | |
|---|---|
| TVS_EDITLABELS | 允许用户直接编辑控件中的项 |
| TVS_DISABLEDRAGDROP | 禁止项从该控件拖走 |
| TVS_SHOWSELALWAYS | 使选项保持被选中状态，即使该控件失去焦点时也保持 |
| TVS_TRACKSELECT | 当鼠标移到控件上时，使控件作出可视反应 |

**说明**

在前面的实例中可以看到，默认情况下树式控件没有边框；用户可以用边框窗口风格为它指定一个边框。参见本附录的下个例子有关边框窗口风格的内容。

## A.2.7 微调按钮控件

### 用Windows API创建微调按钮控件

```
HWND CreateWindowEx(dwExStyle, "msctls_updown32", "Text",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height, hWndParent,
   (HMENU) id, hInstance, NULL);
```

### 用MFC创建微调按钮控件

```
CSpinButtonCtrl m_spin;    // usually embedded in parent class
m_spin.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```


图A-11 微调控件风格

### 可视风格(见图A-11)

### 其他风格

| | |
|---|---|
| UDS_WRAP | 在调整值大于控件范围的尾值或小于初始值时，使该“伙伴”窗口值设为初始值或尾值 |
| DS_NOTHOUSANDS | 在初始化该值时，不在千位之间插入逗号或句号 |
| UDS_AUTOBUDDY | 自动选择父窗口中的前一个窗口作为它的“伙伴”窗口；否则，必须使用 CspinButtonCtrl的SetBuddy ( )函数。微调控件在该伙伴窗口中增加或减少该值 |
| UDS_SETBUDDYINT | 使该控件用它当前数值的一个格式化文本串版本更新它的“伙伴”窗口 |
| UDS_ALIGNRIGHT | 使该控件与“伙伴”窗口的右边对齐 |
| UDS_ALIGNLEFT | |
| UDS_ARROWKEYS | 使箭头键可以增加或减少该控件 |
| UDS_HOTTRACK | 当鼠标移到该控件时，使该控件作出可视反应 |

### 说明

- 微调(Spin)或上/下(Up/Down)按钮控件主要用来增加一个编辑控件。在这种情况下，该按钮叫做一个“伙伴”按钮。
- 在默认的窗口风格下，按钮可以是任意高度，但只有一个标准宽度。然而，用 UDS-HORZ风格，可以设置如图A-11所示的任意大小。

## A.2.8 进度指示控件风格

### 用Windows API创建进度指示控件风格

```
HWND CreateWindowEx(dwExStyle, "msctls_progress32", "Text",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height, hWndParent,
   (HMENU) id, hInstance, NULL);
```

### 用MFC创建进度指示控件风格

```
CProgressCtrl m_progress;    // usually embedded in parent class
m_progress.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```


图A-12 进度指示控件风格

### 可视风格(见图A-12)

### 初始化控件

- 设置控件表示的范围，用：

```
m_progress.SetRange(0,100);
```

- 设置控件的真实进度，用：

```
m_progress.SetPos(33);
```

**说明**

你也许还曾见过这样的进度指示控件，即在控件的中央显示数字指示进度。这里没有自动实现该功能的窗口风格，但只要创建一个普通窗口，然后重载 WM_PAINT消息，用Rectangle ( )绘制条，以及用TextOut ( )输出百分数。

### A.2.9 标题控件风格

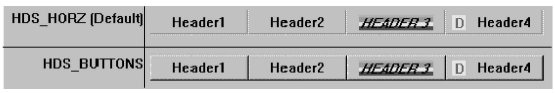
**用Windows API创建标题控件风格**

```
HWND CreateWindowEx(dwExStyle, "SysHeader32", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建标题控件风格**

```
CHeaderCtrl m_header;        // usually embedded in parent class
m_header.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-13)




图A-13 标题控件风格

**初始化控件**

■ 添加一个文本列标题(如图A-13所示的第1、2列)，用：

```
HD_ITEM hdi;
hdi.mask=HDI_TEXT|HDI_WIDTH|HDI_FORMAT;
hdi.fmt=HDF_CENTER|HDF_STRING;
hdi.cxy=100;                    // column width in pixels
hdi.pszText=pszHeader1;              // "text"
hdi.cchTextMax=sizeof(pszHeader1);
m_header.InsertItem(0,&hdi);          // 0=column 1
```

■ 添加一个位图标题(如图A-13所示的第3列)，用：

```
hdi.mask = HDI_FORMAT | HDI_WIDTH | HDI_BITMAP;
hdi.fmt = HDF_CENTER|HDF_BITMAP;
hdi.cxy = 100;                    //column width in pixels
hdi.hbm = HBITMAP(m_bitmap);
m_header.InsertItem(2,&hdi);
```

■ 添加一个具有图标的文本列(如图A-13第4列所示)，用：

```
m_header.SetImageList(&m_imageList);
hdi.mask=HDI_IMAGE| HDI_FORMAT| HDI_TEXT;
hdi.fmt=HDF_LEFT |HDF_IMAGE | HDF_STRING;
hdi.pszText=pszHeader4;          // "text"
hdi.cchTextMax=sizeof(pszHeader4);
hdi.iImage= 1;                // image number in image list
hdi.cxy=100;                  // column width in pixels
m_header.InsertItem(3,&hdi);
```

**其他风格**

| | |
|---|---|
| HDS_DRAGDROP | 允许用户改变列标题 |
| HDS_FULLDRAG | 使控件显示整个标题，即使用户重新调其大小时也这样 |
| HDS_HOTTRACK | 当鼠标移到它的上面时，使控件作出可视反应 |

**说明**

- 单独一个标题控件并无多大意义，它的重要性在于报表模式下作为列表控件的表题。列表控件一旦被创建，可以用下面的代码捆绑一个 CHeaderCtrl类到被那个列表控件使用的标题控件。

```
CListCtrl m_listctrl;
m_listctrl.Create(...);
CHeaderCtrl *pHeader=m_listctrl.GetHeaderCtrl();
```

然后用ModifyStyle ( )和前面所列的风格改变列表控件标题的风格。

- GetHeaderCtrl ( )是CListCtrl的新成员函数，早期的MFC版本可能没有；如果这样的话，还是可以找到标题控件的句柄，通过搜索列表控件的子窗口找到 SysHeader32窗口类，然后把它添加到CHeaderCtrl类。
- 即使已经在标题控件中设置了一个图像，还需把它保存在内存中；通常，可以通过使该图像列表成为父窗口的一个成员变量完成。

## A.2.10 选项卡控件

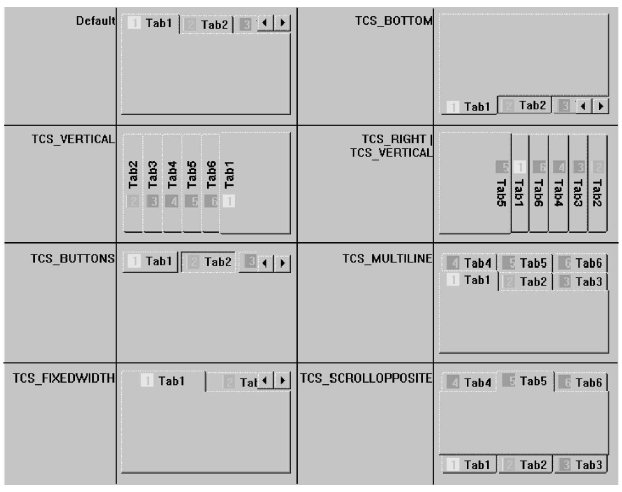
**用Windows API创建选项卡控件**

```
HWND CreateWindowEx(dwExStyle, "SysTabControl32", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建选项卡控件**

```
CTabCtrl m_tab;          // usually embedded in parent class
m_tab.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

**可视风格**(见图A-14)




图A-14 选项卡控件风格

**初始化控件**

■ 添加一个选项卡控件可用的图像列表，用：

```
m_tab.SetImageList(&m_imageList);
```

■ 添加一个具有图像的文本项到选项卡控件，用：

```
TC_ITEM tc;
tc.mask=TCIF_TEXT|TCIF_IMAGE;
tc.pszText=pszTab1;
tc.iImage=0;
m_tab.InsertItem(0,&tc);
```

**其他风格**

| | |
|---|---|
| TCS_MULTISELECT | 用Ctrl键一次可以选取多个选项卡；然而这只适用于 TCS_BUTTONS风格 |
| TCS_FORCEICONLEFT | 使图标在选项卡的左边，但只适用于 TCS_FIXEDWIDTH风格 |
| TCS_FORCELABELLEFT | 使标签在选项卡左边，但只适用于 TCS_FIXEDWIDTH风格 |
| TCS_OWNERDRAWFIXED | 允许父窗口绘制控件 |
| TCS_HOTTRACK | 在鼠标移到该控件上面时，使控件作出可视反应 |

**说明**

■ 单独一个选项卡控件毫无意义 —— 就像一个标题控件没有列表控件一样。如果想要选项卡样式，最好用CPropertySheet创建一个属性表，然后用 CPropertyPage类为它添加属性页；这样，如果想要改变属性表使用的选项卡控件风格，可以用 GetTabControl ( )访问该控件。

```
CPropertySheet ps;
ps.Create(...);
CTabCtrl *pTab=ps.GetTabControl();
```

然后，用ModifyStyle ( )改变风格。

### A.2.11 月历控件

**用Windows API创建月历控件**

```
HWND CreateWindowEx(dwExStyle, "SysMonthCal32", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

**用MFC创建月历控件**

```
CMonthCalCtrl m_month;    // usually embedded in parent class
m_month.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

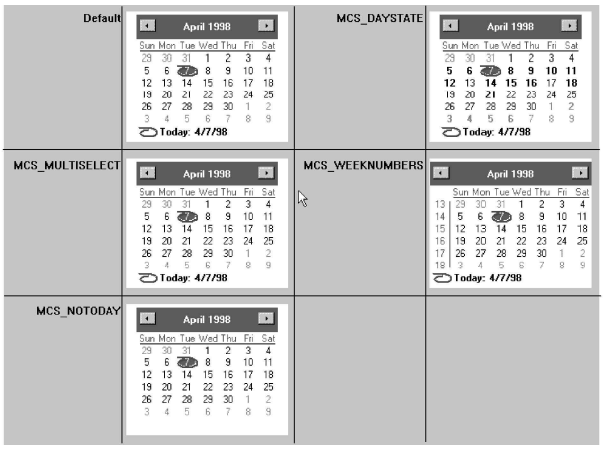
**可视风格**(见图A-15)

**说明**

■ CMonthCalCtrl类是MFC的新类，某些版本可能没有；但还是可以使用 Windows API调用指定SysMonthCal32类创建该控件，然后，通过发送窗口消息操纵它。

■ 在实例中可以看到，在默认情况下，月历控件没有边框。虽然月历控件可以有任意大小，但月历本身在控件中央保持大小不变 —— 通过改变大小，只是增大或减小控件周围

的白色边框。




图A-15 日历控件风格

## A.2.12 日期/时间控件

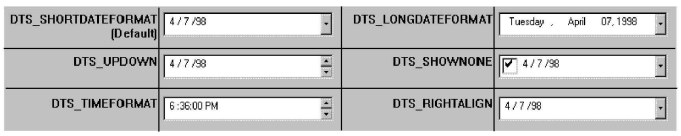
### 用Windows API创建日期/时间控件

```
HWND CreateWindowEx(dwExStyle, "SysDateTimePick32", "",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

### 用MFC创建日期/时间控件

```
CDateTimeCtrl m_datetime;    // usually embedded in parent class
m_datetime.Create(
   WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

### 可视风格(见图A-16)




图A-16 日期/时间控件风格

### 其他风格

| | |
|---|---|
| DTS_SHOWNONE | 除非复选它，否则应用程序请求时不返回一个日期 |
| DTS_APPCANPARSE | 允许用户按F2键时编辑日期；用户编辑完后，控件发送一个 DTN_USERSTRING 消息到父窗口 |
| DTS_RIGHTALIGN | 下拉时，月历控件与日期/时间控件是右对齐的 |

### 说明

- CDateTimeCtrl类是MFC的新类，你的版本可能没有包含它；但还是可以用 Windows API调用指定SysDateTimePick32窗口类使用该控件，然后通过发送窗口消息操纵它。
- 当一个日期/时间控件下拉时，实际上是打开一个月历控件；可以通过调用 CDateTime-Ctrl的GetMonthCalCtrl ( )成员函数访问该控件。

## A.3 对话窗口风格

### 对话窗口

#### 用Windows API创建对话窗口

```
::CreateDialogIndirect(hInst, lpDialogTemplate,
   pParentWnd->GetSafeHwnd(), AfxDlgProc);
```

#### 用MFC创建对话窗口

```
CDialog m_dialog;          // usually embedded in parent class
m_dialog.Create(
   "Text", WS_VISIBLE|WS_CHILD|dwStyle, rect, pParentWnd, id);
```

#### 其他风格

| | |
|---|---|
| DS_ABSALIGN | 用屏幕座标而不是客户窗口座标创建对话框；换句话说，对话框是相对于整个屏幕创建的，而不是相对于应用程序主窗口 |
| DS_SYSMODAL | 使对话框成为最顶端的窗口，直到用户释放它；这对于系统范围导致的错误是有用的 |
| DS_SETFONT | 使CreateDialogIndirect ( )发送一个WM_FONT消息到由它创建的窗口，以定义一个新字体 |
| DS_NOIDLEMSG | 通常，模式对话框在对话框空闲时发送一个 WM_ENTERIDLE消息到物主窗口；该风格使该功能无效 |
| DS_SETFOREGROUND | 使对话框初始显示在前景上 |
| DS_3DLOOK | 使所有由该对话框创建的控件窗口具有3-D外观，也使所有文本不用黑体 |
| DS_FIXEDSYS | 系统默认字体为SYSTEM_FIXED_FONT而不是SYSTEM_FONT |
| DS_CONTROL | 设置该对话框所需的所有风格，使它成为属性表中的一个属性页(WS_CHILD、无边框等) |
| DS_CENTER | 使对话框位于工作空间的中央，这可以是父窗口，如果DS_ABSALIGN被设定，也可以是屏幕的中央 |
| DS_CENTERMOUSE | 创建时，鼠标光标位于对话框的中央位置 |
| DS_CONTEXTHELP | 在对话框的左上角放置一个问号。如果用户单击该问号，然后再单击一个控件，该控件将接收到一个WM_HELP消息 |

### 说明

一个对话框是一个弹出窗口，专门用来创建和操纵大量的窗口控件。事实上， CreateDialogIndirect ( )只调用CreateWindowEx ( )，接着发送少量附加窗口消息到该窗口，然后对话框模板资源提供一个要创建的控件列表。

## A.4 普通窗口风格

到目前为止，我们只是讨论了那些属于一个特定控件的窗口风格，现在，我们将讨论所有窗口通用的窗口风格。

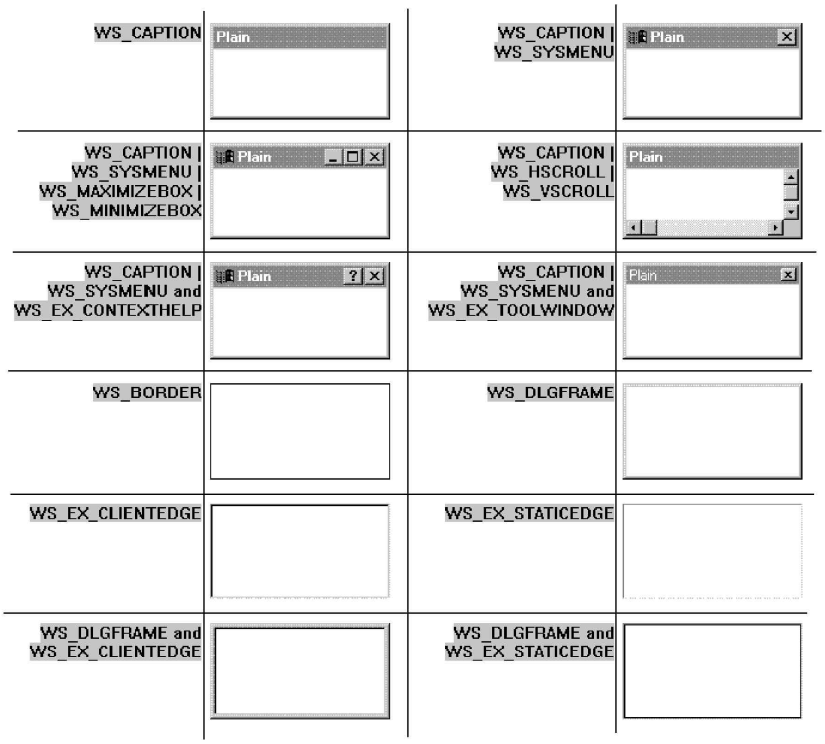
#### 用Windows API创建一个普通窗口

```
HWND CreateWindowEx(dwExStyle, "AfxWnd", "Text",
   WS_CHILD|WS_VISIBLE|dwStyle x, y, width, height,
   hWndParent, (HMENU) id, hInstance, NULL);
```

### 用MFC创建一个普通窗口

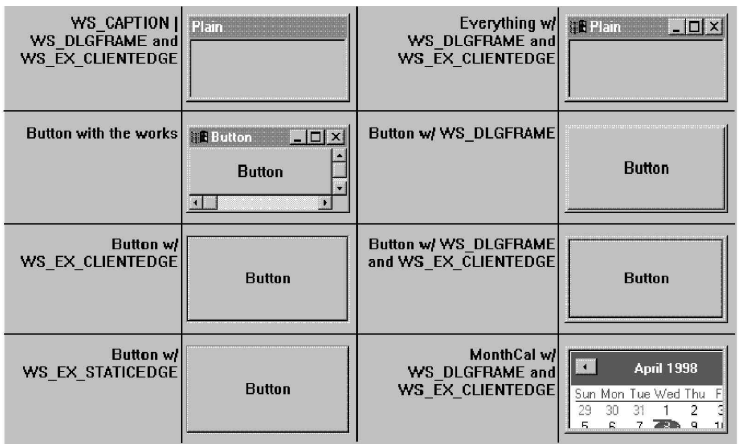
```
CWnd m_wnd;                // usually embedded in parent class
m_wnd.CreateEx(
   dwExStyle,"AfxWnd","Title",dwStyle, x, y, width,
   height, hwndParent, nIDorHMenu, lpParam /*= NULL*/);
```

**可视风格(**见图A-17)




图A-17 普通窗口风格

**可视风格2——有趣的组合**(见图A-18)




图A-18 混合控件风格

## 其他风格

| | |
|---|---|
| WS_CHILD | 通常，完全由父窗口绘制，并且在屏幕上完全包含在它的父窗口中 (如控件窗口和视图) |
| WS_OVERLAPPED | 通常是应用程序的主窗口 (如框架窗口) |
| WS_OVERLAPPEDWINDOW | 包含所有其他通常与一个框架窗口关联的风格 (如边框) |
| WS_POPUP | 通常是一个信息或数据集合窗口 (如对话框和消息框) |
| WS_POPUPWINDOW | 包含所有其他通常与一个弹出式窗口关联的风格 (如边框) |
| WS_CLIPCHILDREN | 禁止父窗口绘制子窗口 |
| WS_CLIPSIBLINGS | 禁止子窗口绘制兄弟窗口 (其他子窗口) |
| WS_GROUP | 与单选按钮一起使用，用来描绘一组控件：在一个单选按钮被单击时，窗口取消单击组中其他按钮；这只是标记组中第一个输入项—丢失的输入项被定义为在下一个WS_GROUP风格发生前的输入项 |
| WS_TABSTOP | 当用户在父窗口中用Tab键切换焦点时，Windows寻找的标志，用以知道一个控件窗口是否有资格接收输入焦点 |
| WS_MAXIMIZE | 不管为该窗口设置的宽和高为多少，创建该窗口时使它最大化为屏幕尺寸 |
| WS_MINIMIZE | 使窗口创建时最小化 |
| WS_VISIBLE | 初始创建一个窗口时，使它可见；默认情况下，该窗口是不可见的，直到调用ShowWindow (SW_SHOW)为止 |
| WS_DISABLED | 灰化一个窗口，并把所有的鼠标单击传输给它的父窗口 |
| WS_EX_ACCEPTFILES | 在拖放模式下，用户释放鼠标时，具有该风格的窗口将接收一个WM_DROPFILES窗口消息 |
| WS_EX_CONTEXTHELP | 在窗口的左上角放置一个问号；如果用户单击它，再单击一个控件，该控件将接收一个WM_HELP消息 |
| WS_EX_CONTROLPARENT | 允许用户用Tab键在该窗口的子窗口中移动焦点；但每个子窗口必须有WS_TABSTOP风格 |
| WS_EX_LEFTSCROLLBAR | 垂直滚动条将显示在窗口的左边，而不是右边，这是指非客户区滚动条，而滚动条控件窗口不受影响 |
| WS_EX_MDICHILD | 创建一个MDI子窗口。在一个MDI应用程序中，主窗口用MDICLIENT窗口类创建的窗口填充它的客户区；然后，该窗口维护创建在它里面的窗口；该风格是那个窗口的一个标志，用以指出它必须维护它的哪个子窗口 |
| WS_EX_NOPARENTNOTIFY | 当一个子窗口被创建、被销毁或被单击时，通常发送一个WM_PARENT-NOTIFY消息到它的父窗口，该风格关闭这一特征 |
| WS_EX_TOPMOST | 该窗口成为一个最顶端的窗口，窗口管理器维护两个列表：一个是常规窗口的列表，另一个是最顶端的窗口列表。管理器总是使最顶端的窗口最后绘制，这可以保持它们在桌面的上端。但是在最顶端的窗口之间，它们还必须竞争最上端的位置 |
| WS_EX_TRANSPARENT | 具有该风格的窗口不擦除它的背景和在它下面的任何窗口，因此允许透视任何一个在它下面的窗口。这也可以通过在创建它的窗口类中不指定一种背景色来实现，然而移动该窗口将使背景与它一起移动。该风格为模式操作设计，通常在另一个窗口中绘制一个突出显示部位时使用。如：当对话框编辑器突出显示一个控件时 |