第8章 视　　图

在SDI或MDI应用程序中，视图是用户与应用程序，尤其是应用程序正在编辑的文档进行交互的主要机制。本章中的所有例子都与视图有关，包括在对话框外创建视图，到把一个视图分割成多个视图。

例32　滚动视图　本例添加一个滚动视图到应用程序中，滚动视图是图形设计应用程序的理想选择。

例33　改变鼠标光标形状　本例讨论怎样有条件地改变光标的形状，这通常也是一个图形设计应用程序的要求。

例34　沙漏光标　本例将讨论怎样把鼠标变成沙漏形状，以指示一个漫长的操作。

例35　窗体视图　本例讨论在对话框外创建一个视图。对话框定义一组控件窗口的大小和位置，这将在后面两章中讨论。

例36　列表视图　本例讨论创建一个包含文本信息的列表的视图，该视图具有以图形突出显示单行文本信息的能力。

例37　动态分割一个视图　本例讨论怎样给应用程序添加视图分割能力。在初始创建应用程序时，AppWizard提供了一个自动添加该特征的机会。

8.1　例32　滚动视图

目标

使视图能够自动地滚动一个比视图大的图像 (见图8-1)。

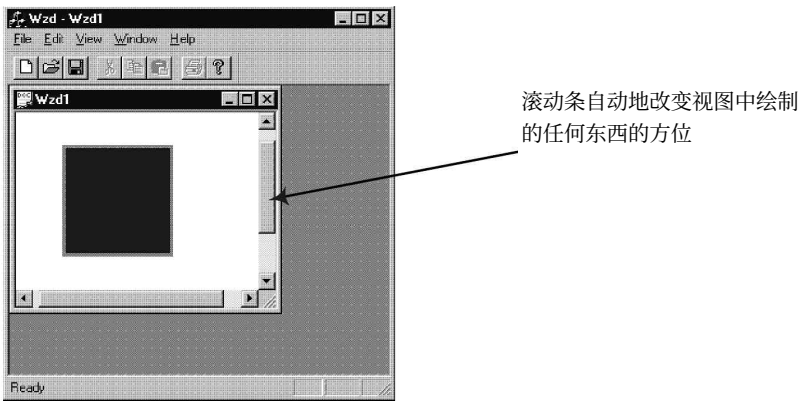


图8-1　添加一个滚动视图使滚动条有效

策略

用AppWizard创建一个带滚动视图的应用程序，并讨论怎样添加一个滚动视图到一个已经

用AppWizard创建的应用程序中。滚动视图是从 CScrollView派生的，而CScrollView本身是从标准MFC CView类派生的。我们将用 CScrollView::SetScrollSizes( )，以像素为单位设置视图的大小。如果用户重新调整视图的大小，使它比创建时的视图小，这时将出现水平和垂直滚动条，因此用户可以继续看到整个视图。

步骤

1. 用AppWizard创建一个滚动视图

在用AppWizard创建应用程序的最后一步，可以看到一个类的目录，这些类是为应用程序创建的。选择CXxxView类(这里的Xxx是工程的名称)，然后在基类组合框中选择CScrollView，并单击Finsh。

2. 用ClassWizard创建一个滚动视图

用ClassWizard创建一个从CScrollView派生的新视图类，然后在应用程序类的InitInstance( )中，用新类替换用来定义应用程序文档模板的类。

```
// add new view class to document template
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_WZDTYPE,
    RUNTIME_CLASS(CWzdDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CWzdScrollView));       <<<<<<<
AddDocTemplate(pDocTemplate);
```

这时，可以从工程中删除旧的视图类。否则，如果这将成为新文档模板的视图类，则只要把它添加到那个模板即可。

3. 设置滚动视图

使用ClassWizard重载CScrollView的OnInitialUpdate( )，在那里可以用 CScrollView::SetScrollSizes( )设置视图的最小像素大小。换句话说，如果用户把它们的视图减小到比该值还小时，滚动条出现，以便他们能够滚动屏幕重新看到整个视图区。

```
void CWzdScrollView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    CSize sizeTotal;
    sizeTotal.cx = 250; // size required to display image
    sizeTotal.cy = 250; // size required to display image
    SetScrollSizes(MM_TEXT, sizeTotal);
}
```

说明

- 要确定滚动区域的大小，先确定图像大小。例如：如果计划观看一张 11″×8 1/2″的纸张，并要求每英寸显示100像素，那么垂直大小应该是11×100=1100像素高。
- CScrollView类提供两种功能：第一种是自动创建滚动条，第二种是通过调整视图的视口(Viewport)处理这些滚动条。视口和窗口在第 4章中已讨论，它们允许在绘制图像时

不必考虑图像的哪一部分将在视图中出现，或者它们将在视图中哪里出现，因为滚动视图会自动调整该视口。所有必须考虑的是，用一个设备环境绘制图像；可惜，这只适用于用设备环境绘制的图像。如果视图包含其他子窗口，那么无论在何时，该视图被滚动时都必须手工移动这些子窗口。用 ClassWizard添加WM_VSCROLL和WM_HSCROLL消息处理函数，然后用CWnd::MoveWindow( )移动子窗口。

- 滚动视图大多应用在图形应用程序或 CAD应用程序中。然而窗体视图也是从滚动视图派生来的，它允许滚动一个大的窗体。但是，当窗体不大时，这有点不美观，因此我们将在下面例子中讨论怎样关闭一个窗体视图中的滚动条。

CD说明

在CD上执行该工程时，一个绘制好的方框将在没有滚动条的视图中出现；然而，如果用鼠标抓住视图的边，并缩小它以隐藏一半方框，这时，滚动条将出现，它允许通过滚动来观看整个方框。

8.2 例33 改变鼠标光标形状

目标

根据用户选择的绘图工具，改变鼠标光标的形状；或者希望在窗口中默认显示的光标不是箭头，而是其他光标，如图8-2中看到的十字光标。

策略

用两种方法改变鼠标形状：第一，通过定义使用新光标的窗口类，改变窗口默认的光标；第二，使用 CWnd的SetCursor( )函数。

图8-2 自定义窗口类以改变鼠标光标形状

步骤

1. 定义一个视图类光标

改变窗口的默认光标，以便鼠标移经那个窗口的客户区的任一时刻，都能显示默认的形状，必须为那个窗口定义一个窗口类。首先，使用 ClassWizard重载那个窗口类的PreCreateWindow( )函数，本例中使用视图类窗口；然后，用 AfxRegisterWndClass定义窗口类。

```
BOOL CWzdView::PreCreateWindow(CREATESTRUCT& cs)
{
   cs.lpszClass = AfxRegisterWndClass(
      CS_DBLCLKS,                                    // double clicks are
                                                     // passed through
      AfxGetApp()->LoadStandardCursor(IDC_CROSS),    // stock cursor, but you
                                                     // can also load your own
      (HBRUSH)(COLOR_WINDOW+1),                      // normal background color
      AfxGetApp()->LoadIcon(IDR_MAINFRAME));         // normal icon
   return CView::PreCreateWindow(cs);
}
```

2. 用CWnd::SetCursor( )改变光标

1) 根据应用程序的模式改变光标，可以用 ClassWizard为WM_SETCURSOR消息添加一个消息处理函数。在该函数中确定是否在一个特定的模式中；如果是，则用 CWnd的SetCursor( )函数改变光标形状。

```
BOOL CWzdView::OnSetCursor(CWnd* pWnd, UINT nHitTest,
   UINT message)
{
   if (m_bDrawMode)
   {
      SetCursor(AfxGetApp()->LoadCursor(IDC_DRAW_CURSOR));
      return TRUE;
   }

   return CView::OnSetCursor(pWnd, nHitTest, message);
}
```

2) 也可以在其他时刻用 SetCursor( )暂时地改变光标形状。

```
void CWzdView::OnLButtonDown(UINT nFlags, CPoint point)
{
   ::SetCursor(AfxGetApp()->LoadCursor(IDC_DRAW_CURSOR1));

   CView::OnLButtonDown(nFlags, point);
}
```

然而，如同本例一样，下一个鼠标移动消息传来，将使光标恢复到使用 SetCursor( )以前的形状，这使得该方法不太可靠。

说明

- 可以只在窗口的客户区改变鼠标光标的形状。
- 有关使用ClassWizard添加一个消息处理函数的例子，参见例 13。

CD说明

在CD上执行该工程时，可以看到视图中默认的鼠标光标是一个十字。选择工具栏上的新Pencil按钮，将使得默认的光标成为一支铅笔；在视图中按下鼠标左键，将使鼠标光标成为一支折断的铅笔。

8.3 例34 沙漏光标

目标

把鼠标光标暂时变成沙漏形状，以指示一个漫长的操作，要求用户应该等待。

策略

用上个例子中描述的方法只能创建一个沙漏光标，然而 MFC提供了一个叫做CWaitCursor的辅助类(Helper Class)，它能自动操作该功能。

步骤

用CWaitCursor( )创建一个沙漏光标

1) 要显示沙漏光标，只要在堆栈中发生漫长处理的位置之前，创建一个 CWaitCursor类的实例。

```
void CClass::Foo()
{
   CWaitCursor wc;
   :  :  :
```

该类的构造函数提供了沙漏光标，而析构函数则恢复原来的光标。

2) 要返回正常的光标而不销毁该类的实例，可以用：

```
SetCursor (AfxGetApp( )→ LoadStandardCursor (IDC_ARROW));
```

3) 此时，如果想恢复沙漏光标，可以用：

```
wc.Restore( )；
```

说明

- 这种方法存在的问题是，建立的函数偶尔会调用其他一些函数，而该函数在不恰当的时刻恢复了光标。要解决该问题，只要在侵入的函数后面调用 CWaitCursor::Restore( )；如果需要从应用程序很少到达的地方调用 Restore( )，可以调用：

```
AfxGetApp( )→ RestoreWaitCursor( )
```

- 沙漏光标更适合于短暂的等待。对于长时间的等待，可以考虑使用一个无模式对话框，并在上面显示简短的消息，描述正进行什么处理；这可以解决一个低级函数关闭沙漏光标的问题，还可以在上面显示一个 Abort按钮，允许用户在不想等待的情况下结束该函数。怎样创建一个无模式对话框参见例 41。
- 使用无模式对话框的另一个好处是，可以用一个进度指示控件指示应用程序的进度；如果进度难以度量或者根本不可能度量，可以用一个动画控件显示一段不断反复的动画。怎样在对话框中放置一段动画参见例 43。

CD说明

在CD上执行该工程时，单击 Test/Wzd菜单命令，将使鼠标光标变成沙漏形状并保持 2秒钟；接着变成箭头光标1秒钟；再变成沙漏光标1秒钟；最后恢复到默认光标。

8.4 例35 窗体视图

目标

把一个对话框模板插入到视图中 (见图8-3)。

策略

在AppWizard的最后一步，创建一个窗体视图，它把对话框模板变成一个视图；还讨论用ClassWizard和Dialog Editor(对话编辑器)给一个已有的应用程序添加一个窗体视图；窗体视图是从CFormView派生的，而CFormView自身是从CScrollView派生的；同时还要做一些新增的

工作，使用户不能重调窗体视图的大小；还要讨论怎样关闭 CScrollView坚持绘制的滚动条；最后讨论如果允许用户重调视图的大小，则怎样调整窗体视图中的控件尺寸。

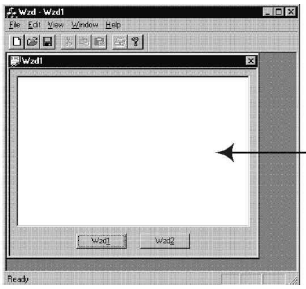


图8-3 创建一个窗体视图，把一个对话框模板变成一个视图

步骤

1. 用AppWizard创建一个窗体视图

1) 在用AppWizard创建应用程序的最后一步中，可以看到一个包含多个类的目录，这些类是为应用程序创建的。选择 CXxxView类(这里的Xxx是工程的名称)，然后在基类的组合框中选择CFormView，并单击Finish。

2) 当创建一个窗体或应用程序时， AppWizard在应用程序资源中创建了一个新的对话框模板。用对话框编辑器添加控件到该模板，并用 ClassWizard为这些控件添加处理函数。

2. 用ClassWizard创建一个窗体视图

1) 创建一个新的对话框模板，并用对话框编辑器添加控件到该模板。开始时模板可以不美观 —— 在以后可以修改它，但是必须确信它的风格是这样的：子窗口、没有边框、不可见和没有标题。

2) 在该窗体上右击鼠标，并从弹出式菜单中选择 ClassWizard，输入一个新的类名。使它的基类为CFormView而不是CDialog，用ClassWizard创建一个从CFormView派生的新类。

3) 在应用程序类的InitInstance( )中，用该新类替换当前使用的用来定义应用程序文档模板的类。

```
// add new view class to document template
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_WZDTYPE,
    RUNTIME_CLASS(CWzdDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CWzdFormView));          <<<<<<<
AddDocTemplate(pDocTemplate);
```

然后可以从工程中删除旧的视图类；否则，如果新类将成为一个新文档模板的视图类，只要添加它到那个模板即可。

3. 更新窗体视图

1) 如果用ClassWizard为该窗体视图创建成员变量，那么需要直接调用 UpdateData( )，以便在窗体和这些成员变量间交换数值，从窗体检索信息可以用下面的代码根据要求完成。

```
UpdateData (TRUE)；
```

2) 通常，视图类的 OnUpdate( )成员函数用文档类的数据更新视图；而这里，应该用UpdateData( )，UpdateData( )用文档的值刷新窗体。

```
void CWzdView::OnUpdate(CView* pSender, LPARAM lHint,
    CObject* pHint)
{
    // store any data from document into member variables of
    // Form View class here

    UpdateData(FALSE);
}
```

4. 固定的窗体视图

1) 如果不让用户改变窗体视图的大小，添加下面的代码行到 SDI应用程序的CChildFrame类的PreCreateWindow( )中，或者MDI应用程序的CMainFrame类的PreCreateWindow( )中。

```
// removes min/max boxes
cs.style &= ~(WS_MAXIMIZEBOX|WS_MINIMIZEBOX);

// makes dialog box unsizable
cs.style &= ~WS_THICKFRAME;
```

2) 在默认情况下，窗体视图创建一个比创建它们的对话框模板大得多的视图，因此，它们的外观相当笨拙；可以把所有控件集中到视图的左边，而右边空出大量的空间。要把视图缩小到模板的大小，用 ClassWizard重载窗体视图类的 OnInitialUpdate( )函数，然后用CScrollView的ResizeParentToFit( )以缩小窗体。

```
void CWzdView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();

    // make frame the size of the original dialog box
    ResizeParentToFit();

    // get rid of those pesky scroll bars by making the point
    // at which they appear very small
    SetScrollSizes(MM_TEXT, CSize(20,20));
}
```

最后调用SetScrollSizes( )的目的是防止CScrollView打开滚动条；CscrollView认为视图小得不足以容纳窗体时，它便打开滚动条，因为在一些平台上，当视图被缩小到正好适合窗体的大小时， CScrollView认为应该打开滚动条。为了防止这种情况的发生，可以通知CScrollView视图的大小正好是20×20像素，因为窗体比那个视图大，因此滚动条不会出现。

在某些应用程序中，需要允许用户调整窗体视图的大小；尤其，如果视图的绝大部分是一个列表控件或一个编辑框，并且它们在底部有一些按钮时。为了允许用户重新调整窗体视图的大小，需要在窗体上经常调整控件的大小和移动控件，这可以在下面看到。

5. 可调整大小的窗体视图

1) 不要使用前面看到的创建一个固定窗体视图的变化；尤其不要添加下面的代码行到框架窗口的PreCreateWindow( )函数中。

```
cs.style &=~WS_THICKFRAME;
```

2) 使用ClassWizard为对话框模板中的每个控件创建控件成员变量，这使得在窗体视中更易于调用每个控件窗口的MoveWindow( )函数。

3) 使用ClassWizard添加一个WM_SIZE消息处理函数到窗体视图；然后，在相同的位置，用每个控件的MoveWindow( )函数调整它们的大小，并指定它们的位置；同时，在视窗改变大小时，必须重新调整控件大小和/或移动控件。

```
void CWzdView::OnSize(UINT nType, int cx, int cy)
{
   CFormView::OnSize(nType, cx, cy);

   if (m_ctrlWzdList.m_hWnd)
   {

      CRect rect;
      m_ctrlWzdButton1.GetClientRect(&rect);

      // list control is always 10 pixels from corners and
      // above buttons (last two arguments are width & height)
      m_ctrlWzdList.MoveWindow(
         10,10,cx-20,cy-rect.Height()-30);

      // buttons are always the size they started 10 pixels
      // from bottom, 20 from each other and centered
      int strt = (cx - (rect.Width()*2+20))/2;

      rect.OffsetRect(strt,cy-rect.Height()-10);
      m_ctrlWzdButton1.MoveWindow(rect);

      rect.OffsetRect(rect.Width()+20,0);
      m_ctrlWzdButton2.MoveWindow(rect);
   }
```

在试图移动控件窗口之前，确信控件窗口类有一个窗口句柄，因为一个 WM_SIZE消息可能在控件窗口创建之前发送到窗体视图。

说明

有关用ClassWizard添加消息处理函数的例子，参见例13。

CD说明

在CD上执行该工程时，可以看到视图包括一个对话框模板控件，用鼠标重调视图的大小，将使得这些控件也改变大小。

8.5 例36 列表视图

目标

创建一个具有多列数据的列表视图(见图8-4)。

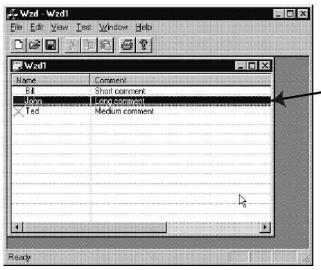


图8-4 创建一个具有多列数据列表的列表视图

策略

使用AppWizard创建一个具有列表视图的应用程序；讨论怎样用 ClassWizard添加列表视图到一个已有的应用程序中；列表视图使用 CListView类，它有一个内嵌的列表控件窗口，在视图中可以访问该控件窗口，可以对它直接操纵，以改变它的风格并给它添加数据。

步骤

1. 使用AppWizard创建一个列表视图

使用AppWizard创建应用程序的最后一步，可以看到一个包含多个类的目录，它们为应用程序创建。选择CXxxView类(Xxx是工程的名称)，然后在基类组合框中，选择 CListView，并单击Finish。

2. 用ClassWizard创建一个列表视图

用ClassWizard创建一个新的从CListView派生的视图类，在应用程序类的 InitInstance( )中用该新类替换当前用来定义应用程序文档模板的类。

```
// add new view class to document template
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_WZDTYPE,
    RUNTIME_CLASS(CWzdDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CWzdListView));          <<<<<<<
AddDocTemplate(pDocTemplate);
```

接着，可以删除工程中的旧视图类；否则，如果这将成为新文档模板的视图类，只要把它添加到那个模板中。

3. 在OnInitialUpdate( )中设置列表视图

1) 用ClassWizard重载列表视图的OnInitialUpdate( )函数。

```
Void CWzdView::OnInitialUpdate( )
{
```

2) 设置列表控件的风格。本例中，设置列表控件为“报表”风格，第一列按字母顺序排列，并让控件总是显示选定内容，甚至在应用程序没有焦点时也要显示。

```
GetListCtrl().ModifyStyle(0,LVS_REPORT| LVS_SHOWSELALWAYS|
    LVS_SORTASCENDING);
```

注意 GetListCtrl( )用来访问支持该视图的真正的列表控件；参考MFC文档有关其他列表控件风格。

3) 接着，设置两种列表控件风格：第一种扩展风格，在列表上画线，分隔列表的行和列；第二种风格允许用户选中整行。(默认情况下，列表控件只允许第一列被选中。)

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

4) 因为本例中使用的报表风格有列标头，因此还需要用 CListCtrl::InsertColumn( )定义并命名这些列。InsertColumn( )的列宽变量以像素为单位，因此，可以用字符的平均像素宽度来帮助设置该变量。

```
CDC* pDC = GetDC();
TEXTMETRIC tm;
pDC->GetTextMetrics(&tm);
GetListCtrl().InsertColumn(0,"Name",LVCFMT_LEFT,
   30 * tm.tmAveCharWidth, 0);
GetListCtrl().InsertColumn(1,"Comment",LVCFMT_LEFT,
   70 * tm.tmAveCharWidth, 1);
```

5) 在列表控件每行的开始处，可以有选择地放置一幅位图图像；然而，必须用GetListCtrl( ).SetImageList( )定义那个控件可用的图像列表。

```
m_ImageList.Create(IDB_STATUS_BITMAP, 15, 1, RGB(0,0,0));
GetListCtrl().SetImageList(&m_ImageList, LVSIL_STATE);
```

4. 更新列表视图

1) 使用ClassWizard重载CListView的OnUpdate( )函数，在该重载函数中，把文档数据拷贝到列表中。本例中，文档包括一个数据类对象列表，这可以在本节的“清单 —— 数据类”中找到。

```
void CWzdView::OnUpdate(CView* pSender, LPARAM lHint,
   CObject* pHint)
{
   CList<CWzdInfo*,CWzdInfo*> *pList=GetDocument()->
      GetInfoList();
   GetListCtrl().DeleteAllItems();
   for (POSITION pos = pList->GetHeadPosition(); pos;)
   {
      CWzdInfo *pInfo = pList->GetNext(pos);
      AddItem(-1,pInfo);
   }
}
```

2) 在OnUpdate中，调用另外一个函数，AddItem，它将真正地把数据填入列表控件。

```
void CWzdView::AddItem(int i,CWzdInfo *pInfo)
{
   if (i==-1)
   {
      i=GetListCtrl().InsertItem(0, pInfo->m_sName);
   }
   else
   {
```

```
        GetListCtrl().SetItemText(i, 0,  pInfo->m_sName);
    }

    GetListCtrl().SetItemText(i, 1,  pInfo->m_sComment);
    GetListCtrl().SetItemData(i,(DWORD)pInfo);
    // tells list control which bitmap to display at start of line
    GetListCtrl().SetItemState(i, INDEXTOSTATEIMAGEMASK(
        pInfo->m_nState), LVIS_STATEIMAGEMASK);
}
```

5. 操作一个列表视图

1) 确定是否有什么内容被用户选取，可以用：

```
if (GetListCtrl().GetSelectedCount())
{
    // yes
}
```

2) 浏览列表控件中只被选中的项，可以用：

```
int i=-1;
 while ((i = GetListCtrl().GetNextItem(i, LVIS_SELECTED)) != -1)
 {
    CWzdInfo *pInfo=(CWzdInfo *)GetListCtrl().GetItemData(i);

}
```

3) 添加一个新项到文档和列表控件中，可以用：

```
CList<CWzdInfo*,CWzdInfo*> *pList=GetDocument()->GetInfoList();
CWzdInfo *pInfo=new CWzdInfo("new","comment",CWzdInfo::NEW);
pList->AddHead(pInfo);
AddItem(-1,pInfo);
```

4) 要查明特定行在列表控件中是可见的，可以用 CListCtrl::EnsureVisible( )，它只是滚动列表，直到显示出选中的项为止。

```
GetListCtrl().EnsureVisible(inx,FALSE);
```

inx变量是要显示的行号。

5) 在用户没有真正选中列表视图中某一项的情况下，要使它显示被选中，可以用下面的代码：

```
LV_ITEM lvi;
lvi.mask = LVIF_STATE;
lvi.iItem = inx;
lvi.stateMask = 0x000f;
lvi.state = LVIS_SELECTED|LVIS_FOCUSED;
GetListCtrl().SetItemState(inx, &lvi);
```

inx变量是要选取的行号。

6) 从列表控件中删除一行，可以用：

```
GetListCtrl().DeleteItem(inx);
```

7) 使控件重画自己，可以用：

```
GetListCtrl().Invalidate();
```

8) 有关列表控件的其他操作，参考 MFC文档中的CListCtrl；要看该列表视图类的完整内

容，参看见本节的“清单 —— 列表视图类”。本节还包括本例中使用的数据类清单“清单 —— 数据类”。

说明

任何控件窗口可以变成一个视图，只要用 CView类创建一个应用程序，然后把需要的MFC通用控件类插入到那个CView类中 —— 作为一个例子，如用CButton控件类。在CView 的OnCreate( )消息处理函数中，调用CButton的Create( )；在CView的OnSize( )消息处理函数中，调用CButton的MoveWindow( )，使它填充屏幕。

CD说明

在CD上执行该工程时，可以看到视图被一个列表控件填充，在 WzdView.cpp的OnTestWzd( )处设置一个断点，然后单击Test/Wzd菜单命令，单步调试以观察列表中的一个项被修改。

清单 —— 列表视图类

```
// WzdView.h : interface of the CWzdView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_)
#define AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#include <afxcview.h>

class CWzdView : public CListView
{
protected:  // create from serialization only
    CWzdView();
    DECLARE_DYNCREATE(CWzdView)

// Attributes
public:
    CWzdDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
```

```
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CWzdView)
    afx_msg void OnTestWzd();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    int         m_AveCharWidth;
    CImageList  m_ImageList;
    int GetTextExtent(int len);
    void AddItem(int ndx,CWzdInfo *pInfo);

};

#ifndef _DEBUG  // debug version in WzdView.cpp
inline CWzdDoc* CWzdView::GetDocument()
    {return (CWzdDoc*)m_pDocument;}
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif // !defined(
    AFX_WZDVIEW_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_)

// WzdView.cpp : implementation of the CWzdView class
//
```

```
#include "stdafx.h"
#include "Wzd.h"

#include "WzdDoc.h"
#include "WzdView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CWzdView

IMPLEMENT_DYNCREATE(CWzdView, CListView)

BEGIN_MESSAGE_MAP(CWzdView, CListView)
    //{{AFX_MSG_MAP(CWzdView)
    ON_COMMAND(ID_TEST_WZD, OnTestWzd)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdView construction/destruction

CWzdView::CWzdView()
{
    m_AveCharWidth=0;
}

CWzdView::~CWzdView()
{
}

BOOL CWzdView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        // the CREATESTRUCT cs

        return CListView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
```

```
// CWzdView drawing

void CWzdView::OnDraw(CDC* pDC)
{
    CWzdDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CWzdView printing

BOOL CWzdView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CWzdView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CWzdView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CWzdView diagnostics

#ifdef _DEBUG
void CWzdView::AssertValid() const
{
    CListView::AssertValid();
}

void CWzdView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}

CWzdDoc* CWzdView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CWzdDoc)));
    return (CWzdDoc*)m_pDocument;
}
#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CWzdView message handlers


void CWzdView::OnInitialUpdate()
{

    m_ImageList.Create(IDB_STATUS_BITMAP, 15, 1, RGB(0,0,0));
    GetListCtrl().SetImageList(&m_ImageList, LVSIL_STATE);
    GetListCtrl().ModifyStyle(
        0,LVS_REPORT|LVS_SHOWSELALWAYS|LVS_SORTASCENDING);
    GetListCtrl().SendMessage(
        LVM_SETEXTENDEDLISTVIEWSTYLE,0,LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT);
    GetListCtrl().InsertColumn(0,"Name",LVCFMT_LEFT,GetTextExtent(30),0);
    GetListCtrl().InsertColumn(1,"Comment",LVCFMT_LEFT,GetTextExtent(70),1);

    CListView::OnInitialUpdate();

}

int CWzdView::GetTextExtent(int len)
{
    CDC* dc = GetDC();

    if (! m_AveCharWidth)
    {
        TEXTMETRIC tm;
        dc->GetTextMetrics(&tm);
        m_AveCharWidth = tm.tmAveCharWidth;
    }
    CSize size(m_AveCharWidth * len, 0);
    dc->LPtoDP(&size);
    ReleaseDC(dc);
    return size.cx;
}

void CWzdView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    CList<CWzdInfo*,CWzdInfo*> *pList=GetDocument()->GetInfoList();
    GetListCtrl().DeleteAllItems();
    for (POSITION pos = pList->GetHeadPosition(); pos;)
    {
        CWzdInfo *pInfo = pList->GetNext(pos);
        AddItem(-1,pInfo);
    }
}

void CWzdView::AddItem(int i,CWzdInfo *pInfo)
```

```
{
    if (i==-1)
    {
        i=GetListCtrl().InsertItem(0, pInfo->m_sName);
    }
    else
    {
        GetListCtrl().SetItemText(i, 0,  pInfo->m_sName);
    }

    GetListCtrl().SetItemText(i, 1,  pInfo->m_sComment);
    GetListCtrl().SetItemData(i,(DWORD)pInfo);
    GetListCtrl().SetItemState(i, INDEXTOSTATEIMAGEMASK(pInfo->m_nState),
        LVIS_STATEIMAGEMASK);
}


void CWzdView::OnTestWzd()
{
    // determine if anything was selected
    if (GetListCtrl().GetSelectedCount())
    {
        // yes
    }

    // loop through selections
    int i=-1;
    while ((i = GetListCtrl().GetNextItem(i, LVIS_SELECTED)) != -1)
    {
        CWzdInfo *pInfo=(CWzdInfo *)GetListCtrl().GetItemData(i);

    }

    // add item to list
    CList<CWzdInfo*,CWzdInfo*> *pList=GetDocument()->GetInfoList();
    CWzdInfo *pInfo=new CWzdInfo("new","comment",CWzdInfo::NEW);
    pList->AddHead(pInfo);
    AddItem(-1,pInfo);

    // to modify
    i = 1;
    pInfo=(CWzdInfo *)GetListCtrl().GetItemData(i);
    //
    AddItem(i,pInfo);

    // to ensure a line is visible
    GetListCtrl().EnsureVisible(i,FALSE);

    // to select a line
```

```
    LV_ITEM lvi;
    lvi.mask = LVIF_STATE;
    lvi.iItem = i;
    lvi.stateMask = 0x000f;
    lvi.state = LVIS_SELECTED|LVIS_FOCUSED;
    GetListCtrl().SetItemState(i, &lvi);

    // to delete a line
    GetListCtrl().DeleteItem(i);

    // to redraw view
    GetListCtrl().Invalidate();

}
```

清单——数据类

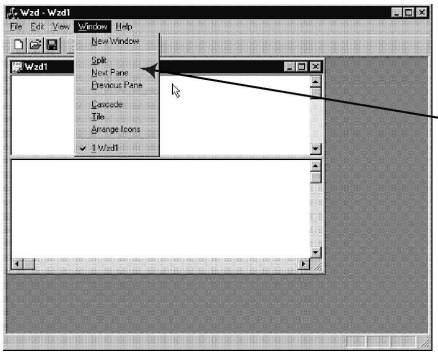
```
#ifndef WZDINFO_H
#define WZDINFO_H


class CWzdInfo : public CObject
{
public:

enum STATES {
    OLD,
    NEW,
    MODIFIED,
    DELETED
};

    DECLARE_SERIAL(CWzdInfo)

    CWzdInfo();
    CWzdInfo(CString sName,CString sComment,int nState);

    void Set(CString sName,CString sComment,int nVersion, int nState);

    //misc info
    CString m_sName;
    CString     m_sComment;
    int m_nVersion;
    int m_nState;

    CWzdInfo& operator=(CWzdInfo& src);

};
#endif
```

```
// WzdInfo.cpp : implementation of the CWzdInfo class
//

#include "stdafx.h"
#include "WzdInfo.h"

/////////////////////////////////////////////////////////////////////////////
// CWzdInfo

IMPLEMENT_SERIAL(CWzdInfo, CObject, 1)


CWzdInfo::CWzdInfo()
{
    m_sName=_T("");
    m_sComment=_T("");
    m_nVersion=1;
    m_nState=CWzdInfo::NEW;
}

CWzdInfo::CWzdInfo(CString sName,CString sComment,int nState) :
    m_sName(sName),m_sComment(sComment),m_nState(nState)
{
}

void CWzdInfo::Set(CString sName,CString sComment,int nVersion, int nState)
{
    m_sName=sName;
    m_sComment=sComment;
    m_nVersion=nVersion;
    m_nState=nState;
}

CWzdInfo& CWzdInfo::operator=(CWzdInfo& src)
{
    if(this != &src)
    {
        m_sName = src.m_sName;
        m_sComment = src.m_sComment;
        m_nVersion = src.m_nVersion;
        m_nState = src.m_nState;
    }
    return *this;
}
```

8.6 例37 动态分割一个视图

目标

允许用户把一个视图分割成两部分，以便能看到同一个文档的不同部分 (见图8-5)。



添加Split View（分割视图）命令到Windows菜单

图8-5 分隔一个视图以使用户可以看到同一个文档的不同部分

策略

可以要求AppWizard添加一个分割视图命令到应用程序。然而，如果没有那么有远见，或者要修改 AppWizard创建的默认设置，可以在 Mainframe或ChildFrame类中插入一个CSplitterWnd类变量，然后重载那个类的 OnCreateClient( )，以初始化该变量。添加一个 Split菜单命令到视图中。

步骤

1. 使用AppWizard添加一个分割命令

创建应用程序时， AppWizard的第四步包括一个 Advanced按钮，单击它，选取 Window Styles选项卡，并单击Use Split Window。一个新的Split命令将被添加到视图菜单命令中。

2. 手工添加一个分割命令

1) 把一个CSplitterWnd成员变量插入一个MDI应用程序的CChildFrame类中，或者SDI应用程序的CMainFrame类中。

```
CSplitterWnd m_wndSplitter;
```

2) 用ClassWizard重载主框架或子框架类的OnCreateClient( )，在那里初始化CSplitterWnd变量，以创建分别多达两个的垂直窗格和水平窗格。

```
BOOL CChildFrame::OnCreateClient(LPCREATESTRUCT lpcs,
    CCreateContext* pContext)
{
    if (!m_wndSplitter.Create(this,
        2,          // maximum rows (maximum == 2)
        1,          // maximum columns (maximum == 2)
        CSize(5,5),    // minimum pane size in pixels
        pContext))
    {
        TRACE0( " Failed to Create Splitter Window\n " );
        return FALSE;
    }

    return TRUE; //CMDIChildWnd::OnCreateClient(lpcs,
        pContext);
```

}

在这里返回TRUE，则不需调用基类的OnCreateClient( )，否则基类将只重载分割类。

3) 在开始创建应用程序时，不管是否要求分割窗口，分割一个视图的基础都已经存在于应用程序中。要启用该功能，只需添加菜单项到 Window菜单。对于要添加的菜单项和要使用的菜单ID，用AppWizard创建一个具有分割窗口的新的应用程序，然后剪切和粘贴那些菜单项到原来的应用程序中。

说明

- 既可以动态地创建分割窗口，也可以静态地创建。动态创建时，只能分割成四个窗格；而静态创建的分割窗口可以分割的窗格数，只受限于你可以分割最小窗格大小的能力。参见MFC文档中有关CSplitterWnd的成员函数调用，以创建静态分割窗口的内容。
- 实际上，前面设置的最小窗格大小是分割窗口消失大小。换句话说，如果用户把分割条从视图的一边移动到该距离以内，则分割条消失，并且视图重新变成整个窗口的大小。
- AppWizard给予用户一个2×2分割，最多可以达到4个视图。然而，大多数应用程序通常只允许一个视图被分割成两个视图，要把选项限制到两个视图，则把CSplitterWnd::Create( )函数中水平和垂直数设为1。
- 分割视图时，一个完全相同的视图类的实例被创建，它指向完全相同的文档类，因此在该应用程序中访问其他类时，只能从视图中访问一个文档；然而，可以从文档中访问一组视图。如果需要应用程序支持分割视图，那么永远不要在视图中存取文档信息，事实上，应该明白视图类只能观看和编辑文档类，而不能做其他任何事情。

CD说明

在CD上执行该工程时，可以用Window菜单中的命令分割视图。