第14章 杂 类

收集到本章中的例子的唯一标准是因为没有足够的相关主题的例子使它们独立成章。剪切和粘贴，以及MFC的数据集类几乎都有足够的例子使它们成为一章，但是由于时间的原因，除了书上所列例子外，其余的例子都被省略了。尽管如此，在本章中还是包含了几个有用的例子。

例74 剪切、拷贝和粘贴文本数据 讨论编辑控件窗口内建的剪切和粘贴功能。

例75 剪切、拷贝和粘贴多信息文本数据 讨论多信息编辑控件窗口内建的剪切和粘贴功能。

例76 剪切、拷贝和粘贴二进制数据 使用串行化剪切和粘贴数据类。

例77 数组函数 讨论MFC类维护相同的类或指针的数组。

例78 列表函数 讨论MFC类维护相同的类或指针的链接列表。

例79 映像函数 讨论MFC类维护使用二进制或文本关键字访问相同的类和指针。

例80 系统键盘输入 定向一些系统键，如Delete键，到视图类的Delete ( )函数。

例81 时间 讨论使用MFC类可以访问的时间格式。

14.1 例74 剪切、拷贝和粘贴文本数据

目标

从编辑控件中剪切或拷贝字符串，然后把它粘贴到控件中的另一个地方或其他控件中。用户也可以使用弹出式菜单进行剪切和粘贴，如图 14-1所示。

策略

实际上，编辑控件提供了剪贴功能，在控件上右击鼠标，会弹出如上的菜单。因此，本例的目的是讨论如何自己实现这个弹出菜单，以便以后可以加入自己的菜单命令。还将讨论编辑控件中支持剪贴的成员函数，并把这种功能封装到自己的 CEdit控件的派生类。

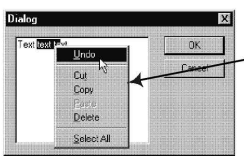

图14-1 添加用户自己的菜单命令到剪贴文本弹出菜单中

步骤

1. 创建一个新的编辑控件类

使用 ClassWizard创建一个新的编辑控件类，该类是从 CEdit派生过来的。再使用ClassWizard添加WM_RBUTTONDOWN消息处理函数到这个类中。

2. 装入和启用弹出式菜单

1) 使用菜单编辑器创建一个如同图 14-1的菜单资源。

2) 编辑WM_RBUTTONDOWN消息处理函数，创建一个基于菜单资源的弹出式菜单。

```
void CWzdEdit::OnRButtonDown(UINT nFlags, CPoint point)
{
CMenu menu;
   // load a menu from the resources
   menu.LoadMenu(IDR_SELECTION_MENU);

   // get a pointer to actual popup menu
   CMenu* pPopup = menu.GetSubMenu(0);
```

一旦装入，弹出式菜单的所有菜单项都被启用。但有时可能需要禁用某些菜单项以表示其当前不可用。例如，若没有任何东西需要取消，我们必须禁用 Undo命令。

3) 使用CEdit的CanUndo ( )启用或禁用Undo命令。

```
UINT nUndo=(CanUndo() ? 0 : MF_GRAYED);
pPopup->EnableMenuItem(ID_EDIT_UNDO,  MF_BYCOMMAND|nUndo);
```

4) 对于依赖于编辑控件中所选的文本的命令，使用 CEdit::GetSel ( )获得当前选中文本的开始和结束位置。若开始和结束位置相等，即没有选中任何文字，此时 Cut、Copy和Delete命令均被禁用。

```
int beg,end;
GetSel(beg,end);
UINT nSel=((beg!=end) ? 0 : MF_GRAYED);
pPopup->EnableMenuItem(ID_EDIT_CUT, MF_BYCOMMAND|nSel);
pPopup->EnableMenuItem(ID_EDIT_COPY, MF_BYCOMMAND|nSel);
pPopup->EnableMenuItem(ID_EDIT_CLEAR, MF_BYCOMMAND|nSel);
```

5) 对于Paste命令，我们需要检查剪贴板中是否有可用的文本。

```
UINT nPaste=(::IsClipboardFormatAvailable(CF_TEXT) ? 0 :
   MF_GRAYED);
pPopup->EnableMenuItem(ID_EDIT_PASTE, MF_BYCOMMAND|nPaste);
```

6) 现在，我们可以显示弹出式菜单，等待用户选择。

```
 // pop up the menu
   CPoint pt;
   GetCursorPos(&pt);
   pPopup->TrackPopupMenu(TPM_RIGHTBUTTON, pt.x, pt.y, this);
   pPopup->DestroyMenu();

   CEdit::OnRButtonDown(nFlags, point);
}
```

3. 处理弹出式菜单命令

1) 若用户选择一个命令，那么该命令将作为一条带命令 ID的WM_COMMAND消息返回到窗口中，使用菜单编辑器可以给命令 ID赋值。要处理这些命令，必须手工添加下面的消息宏到消息映像中。

```
ON_COMMAND(ID_EDIT_UNDO, OnUndo)
ON_COMMAND(ID_EDIT_CUT, OnCut)
ON_COMMAND(ID_EDIT_COPY, OnCopy)
ON_COMMAND(ID_EDIT_PASTE, OnPaste)
ON_COMMAND(ID_EDIT_CLEAR, OnDelete)
ON_COMMAND(ID_EDIT_SELECT_ALL, OnSelectAll)
```

2) 因为响应这些命令的CEdit ( )成员函数一般没有参数，因此，我们可以简单地在编辑类

的.h文件中，作为嵌入函数实现这些消息处理函数。

```
//{{AFX_MSG(CWzdEdit)
afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
//}}AFX_MSG
afx_msg void OnUndo(void){Undo();};
afx_msg void OnCut(void){Cut();};
afx_msg void OnCopy(void){Copy();};
afx_msg void OnPaste(void){Paste();};
afx_msg void OnDelete(void){Clear();};
afx_msg void OnSelectAll(void){SetSel(0,-1);};
DECLARE_MESSAGE_MAP()
```

正如你所看到的，只有Select All命令的需求复杂一些。

3) 关于编辑类的详细清单，请参见本节的“清单——编辑控件类”。

说明

可以添加自己的命令到这个弹出式菜单中。为了使这个类完全被封装，也可以考虑动态地创建弹出式菜单，如例21所示的一样，而不是依赖于外部菜单资源。

CD说明

在CD上运行该工程时，单击Test/Wzd菜单命令打开一个对话框。这个对话框包含一个编辑框，若在该编辑框上右击鼠标将显示一个带有剪贴、拷贝、粘贴等编辑命令的弹出式菜单。

清单——编辑控件类

```
#if !defined(AFX_WZDEDIT_H__19B437E6_E7F5_11D1_A18D_DCB3C85EBD34__INCLUDED_)
#define AFX_WZDEDIT_H__19B437E6_E7F5_11D1_A18D_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// WzdEdit.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdEdit window

class CWzdEdit : public CEdit
{
// Construction
public:
    CWzdEdit();

// Attributes
public:

// Operations
public:
```

```
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdEdit)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdEdit();

    // Generated message map functions
protected:
    //{{AFX_MSG(CWzdEdit)
    afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        afx_msg void OnUndo(void){Undo();};
        afx_msg void OnCut(void){Cut();};
        afx_msg void OnCopy(void){Copy();};
        afx_msg void OnPaste(void){Paste();};
        afx_msg void OnDelete(void){Clear();};
        afx_msg void OnSelectAll(void){SetSel(0,-1);};
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(
    AFX_WZDEDIT_H__19B437E6_E7F5_11D1_A18D_DCB3C85EBD34__INCLUDED_)

// WzdEdit.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdEdit.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdEdit

CWzdEdit::CWzdEdit()
{
```

```
}

CWzdEdit::~CWzdEdit()
{
}

BEGIN_MESSAGE_MAP(CWzdEdit, CEdit)
    //{{AFX_MSG_MAP(CWzdEdit)
    ON_WM_RBUTTONDOWN()
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_EDIT_UNDO, OnUndo)
    ON_COMMAND(ID_EDIT_CUT, OnCut)
    ON_COMMAND(ID_EDIT_COPY, OnCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnPaste)
    ON_COMMAND(ID_EDIT_CLEAR, OnDelete)
    ON_COMMAND(ID_EDIT_SELECT_ALL, OnSelectAll)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdEdit message handlers


void CWzdEdit::OnRButtonDown(UINT nFlags, CPoint point)
{
    CMenu menu;
    // load a menu from the resources
    menu.LoadMenu(IDR_SELECTION_MENU);

    // get a pointer to actual popup menu
    CMenu* pPopup = menu.GetSubMenu(0);

    // enable/disable Undo command
    UINT nUndo=(CanUndo() ? 0 : MF_GRAYED);
    pPopup->EnableMenuItem(ID_EDIT_UNDO,  MF_BYCOMMAND|nUndo);

    // enable/disable selection commands
    int beg,end;
    GetSel(beg,end);
    UINT nSel=((beg!=end) ? 0 : MF_GRAYED);
    pPopup->EnableMenuItem(ID_EDIT_CUT, MF_BYCOMMAND|nSel);
    pPopup->EnableMenuItem(ID_EDIT_COPY, MF_BYCOMMAND|nSel);
    pPopup->EnableMenuItem(ID_EDIT_CLEAR, MF_BYCOMMAND|nSel);

    // enable/disable Paste command
    UINT nPaste=(::IsClipboardFormatAvailable(CF_TEXT) ? 0 : MF_GRAYED);
    pPopup->EnableMenuItem(ID_EDIT_PASTE, MF_BYCOMMAND|nPaste);

    // pop up the menu
    CPoint pt;
    GetCursorPos(&pt);
```

```
    pPopup->TrackPopupMenu(TPM_RIGHTBUTTON, pt.x, pt.y, this);
    pPopup->DestroyMenu();

    CEdit::OnRButtonDown(nFlags, point);
}
```

14.2 例75 剪切、拷贝、粘贴多信息文本数据

目标

从多信息文本控件中剪切或拷贝文件，然后把它粘贴到控件的其他位置或者别的控件中。也可使用弹出式菜单进行剪切和粘贴，如图14-2所示。

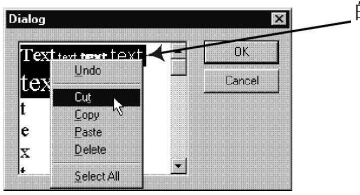

图14-2 创建一个剪贴多信息文本的弹出式菜单

策略

与CEdit不同的是，多信息文本控件并未提供这种功能。我们将使用 CRichEditCtrl类和CMenu类的成员函数创建弹出式菜单，并把所有这些功能封装到 CRichEditCtrl的派生类中。

步骤

1. 设置应用程序

1) 支持多信息文本编辑功能的初始化工作必须与 MFC的其他控件类分离开来。在应用程序类的InitInstance ( )函数中，添加下面所示的一行代码。

```
BOOL CWzdApp::InitInstance()
{
    AfxInitRichEdit();

    :  :  :
```

2) 使用菜单编辑器，创建一个如同图 14-2所示的弹出式菜单的菜单资源。

3) 使用ClassWizard创建一个派生于 CRichEditCtrl的新多信息文本编辑控件，然后用ClassWizard添加一个WM-RBUTTONDOWN消息处理函数。

2. 装入并启用弹出式菜单

1) 在WM_RBUTTONDOWN消息处理函数中，首先生成一个基于上步创建的菜单资源的弹出式菜单。

```
void CWzdRichEditCtrl::OnRButtonDown(UINT nFlags, CPoint point)
{
    CMenu menu;
    // load a menu from the resources
    menu.LoadMenu(IDR_SELECTION_MENU);

    // get a pointer to actual popup menu
```

```
CMenu* pPopup = menu.GetSubMenu(0);
```

一旦装入，弹出式菜单中所有的菜单项都被启用。但有时可能需要禁用某些菜单项以标识当前不可用。例如，若没有任何东西需要取消的话，我们必须禁用 Undo命令。

2) 使用CRichEditCtrl的CanUndo ( )，启用或禁用Undo命令。

```
// enable/disable Undo command
UINT nUndo=(CanUndo() ? 0 : MF_GRAYED);
pPopup->EnableMenuItem(ID_EDIT_UNDO, MF_BYCOMMAND|nUndo);
```

3) 对于依赖于控件中所选的文本的命令，使用 CRichEditCtrl::GetSelectionType ( )，确定是否选择了文本。若选择类型为 SEL_EMPTY，则Cut、Copy和Delete命令为无效。

```
UINT nSel=((GetSelectionType()!=SEL_EMPTY) ? 0 : MF_GRAYED);
pPopup->EnableMenuItem(ID_EDIT_CUT, MF_BYCOMMAND|nSel);
pPopup->EnableMenuItem(ID_EDIT_COPY, MF_BYCOMMAND|nSel);
pPopup->EnableMenuItem(ID_EDIT_CLEAR, MF_BYCOMMAND|nSel);
```

4) 对于Paste命令，我们使用CRichEditCtrl::CanPaste ( )。

```
UINT nPaste=(CanPaste() ? 0 : MF_GRAYED);
pPopup->EnableMenuItem(ID_EDIT_PASTE, MF_BYCOMMAND|nPaste);
```

5) 现在，可以显示弹出式菜单，等待用户选择。

```
    CPoint pt;
    GetCursorPos(&pt);
    pPopup->TrackPopupMenu(TPM_RIGHTBUTTON, pt.x, pt.y, this);
    pPopup->DestroyMenu();

    CRichEditCtrl::OnRButtonDown(nFlags, point);
}
```

3. 处理弹出式菜单命令

1) 如果用户单击了一个命令，那么该命令将作为一条带命令 ID的WM_COMMAND消息返回到窗口中。可以使用菜单编辑器给该命令 ID赋值。为了处理这些命令，必须手工添加下列消息宏到消息映像中。

```
ON_COMMAND(ID_EDIT_UNDO, OnUndo)
ON_COMMAND(ID_EDIT_CUT, OnCut)
ON_COMMAND(ID_EDIT_COPY, OnCopy)
ON_COMMAND(ID_EDIT_PASTE, OnPaste)
ON_COMMAND(ID_EDIT_CLEAR, OnDelete)
ON_COMMAND(ID_EDIT_SELECT_ALL, OnSelectAll)
```

2) 因为响应这些消息的CRichEditCtrl成员函数一般不带参数，因此我们可以简单地在类的.h文件中，以嵌入函数实现这些消息处理。

```
//{{AFX_MSG(CWzdEdit)
afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
//}}AFX_MSG
afx_msg void OnUndo(void){Undo();};
afx_msg void OnCut(void){Cut();};
afx_msg void OnCopy(void){Copy();};
afx_msg void OnPaste(void){Paste();};
afx_msg void OnDelete(void){Clear();};
afx_msg void OnSelectAll(void){SetSel(0,-1);};
```

DECLARE_MESSAGE_MAP()

正如所看到的，只有Select All命令的需求复杂一些。

3) 关于编辑类的详细清单，参见本节的“清单——多信息文本编辑控件类”。

说明

- 在MFC的某些版本中，可能会发现多信息文本编辑控件的 Cut、Copy和Paste函数导致应用程序出错以至退出。然而，这个发行版本没有这个问题。
- 在这个弹出式菜单中理想的情况是应包含 Font(字体)和Paragraph(段落)菜单命令。应用多信息文本编辑控件类的成员函数，也很容易实现这两个命令处理函数。

CD说明

在CD上运行该工程时，单击 Test/Wzd菜单命令以打开一个对话框。这个对话框包含一个多信息文本编辑框，用右键单击它，将显示一个带有剪切、拷贝、粘贴等编辑命令的弹出式菜单。

清单——多信息文本编辑控件类

```
#if !defined(
   AFX_WZDRICHEDITCTRL_H__19B437E7_E7F5_11D1_A18D_DCB3C85EBD34__INCLUDED_)
#define AFX_WZDRICHEDITCTRL_H__19B437E7_E7F5_11D1_A18D_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// WzdRichEditCtrl.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CWzdRichEditCtrl window

class CWzdRichEditCtrl : public CRichEditCtrl
{
// Construction
public:
   CWzdRichEditCtrl();

// Attributes
public:

// Operations
public:

// Overrides
   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CWzdRichEditCtrl)
   //}}AFX_VIRTUAL
```

```
// Implementation
public:
    virtual ~CWzdRichEditCtrl();

    // Generated message map functions
protected:
    //{{AFX_MSG(CWzdRichEditCtrl)
    afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
    //}}AFX_MSG
    afx_msg void OnUndo(void){Undo();};
    afx_msg void OnCut(void){Cut();};
    afx_msg void OnCopy(void){Copy();};
    afx_msg void OnPaste(void){Paste();};
    afx_msg void OnDelete(void){Clear();};
    afx_msg void OnSelectAll(void){SetSel(0,-1);};
    DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(
    AFX_WZDRICHEDITCTRL_H__19B437E7_E7F5_11D1_A18D_DCB3C85EBD34__INCLUDED_)
// WzdRichEditCtrl.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdRichEditCtrl.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdRichEditCtrl

CWzdRichEditCtrl::CWzdRichEditCtrl()
{
}

CWzdRichEditCtrl::~CWzdRichEditCtrl()
{
}


BEGIN_MESSAGE_MAP(CWzdRichEditCtrl, CRichEditCtrl)
```

```
    //{{AFX_MSG_MAP(CWzdRichEditCtrl)
    ON_WM_RBUTTONDOWN()
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_EDIT_UNDO, OnUndo)
    ON_COMMAND(ID_EDIT_CUT, OnCut)
    ON_COMMAND(ID_EDIT_COPY, OnCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnPaste)
    ON_COMMAND(ID_EDIT_CLEAR, OnDelete)
    ON_COMMAND(ID_EDIT_SELECT_ALL, OnSelectAll)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdRichEditCtrl message handlers

void CWzdRichEditCtrl::OnRButtonDown(UINT nFlags, CPoint point)
{
    CMenu menu;
    // load a menu from the resources
    menu.LoadMenu(IDR_SELECTION_MENU);

    // get a pointer to actual popup menu
    CMenu* pPopup = menu.GetSubMenu(0);

    // enable/disable Undo command
    UINT nUndo=(CanUndo() ? 0 : MF_GRAYED);
    pPopup->EnableMenuItem(ID_EDIT_UNDO,  MF_BYCOMMAND|nUndo);

    // enable/disable selection commands
    UINT nSel=((GetSelectionType()!=SEL_EMPTY) ? 0 : MF_GRAYED);
    pPopup->EnableMenuItem(ID_EDIT_CUT, MF_BYCOMMAND|nSel);
    pPopup->EnableMenuItem(ID_EDIT_COPY, MF_BYCOMMAND|nSel);
    pPopup->EnableMenuItem(ID_EDIT_CLEAR, MF_BYCOMMAND|nSel);

    // enable/disable Paste command
    UINT nPaste=(CanPaste() ? 0 : MF_GRAYED);
    pPopup->EnableMenuItem(ID_EDIT_PASTE, MF_BYCOMMAND|nPaste);

    // pop up the menu
    CPoint pt;
    GetCursorPos(&pt);
    pPopup->TrackPopupMenu(TPM_RIGHTBUTTON, pt.x, pt.y, this);
    pPopup->DestroyMenu();

    CRichEditCtrl::OnRButtonDown(nFlags, point);
}
```

14.3 例76 剪切、拷贝和粘贴二进制数据

目标

从用户文档中剪切或拷贝二进制数据，然后把它粘贴到文档的其他位置或其他文档中。

策略

本例基于如下假设：用户文档是由若干专有的数据类组成。这些数据类可能包括图形、图像、图标及控件窗口等，这些数据类被绘制在应用程序的视图中。本例同时假定已经实现了用鼠标选取一个或多个这些绘制对象的某种方法。本例所提供的功能包括剪切或拷贝一组选择的数据类到剪贴板上，因此这些对象可以被粘贴在文档的其他位置或其他文档中。

我们将使用串行化功能来剪切或拷贝所选数据到剪贴板中。第 13章已经讨论过了串行化的问题，那里主要用于从磁盘上调出或保存文档到磁盘。然而，它同样可以很完美地用于调出或保存文档的数据到剪贴板上。我们将在新的选择类中封装剪切和粘贴功能。

步骤

1. 创建一个CSelection类

1) 使用Developer Studio创建一个派生于CObject的新类CSelection，单击Insert/New Class…菜单命令打开New Class…对话框。然后选择Class Type为Generic Class。

2) 在新类的构造函数中，为文档创建一个新的剪贴板类型。拥有这种类型剪贴板的文档可以粘贴该剪贴板类型，但其他类型的文档不行。

```
CWzdSelect::CWzdSelect()
{
   m_clipboardFormat = ::RegisterClipboardFormat("CWzdInfo1");
}
```

3) 按如下方式，添加一个Select ( )成员函数到新类中。Select ( )维护一系列的选项。若有新的选项，则视图类调用该函数。这个函数还跟踪最后一次被选中的数据项。

```
void CWzdSelect::Select(CWzdInfo1 *pInfo)
{
   m_pActiveSelection=pInfo;
   if (!m_WzdSelectionList.Find(pInfo))
   {
      m_WzdSelectionList.AddTail(pInfo);
   }
}
```

4) 在类中加入Serialize ( )函数。它的任务是串行化类的选项列表到剪贴板中。

```
void CWzdSelect::Serialize(CArchive& ar)
{
   int nCount;
   CObject::Serialize(ar);
   if(ar.IsStoring())
   {
      nCount = m_WzdSelectionList.GetCount();
      ar << nCount;
      for (POSITION pos =
         m_WzdSelectionList.GetHeadPosition(); pos;)
      {
         m_WzdSelectionList.GetNext(pos)->Serialize(ar);
      }
```

```
    }
    else
    {
        ar >> nCount;
        while (nCount-- > 0)
        {
            CWzdInfo1* pInfo = new CWzdInfo1;
            pInfo->Serialize(ar);
            m_WzdSelectionList.AddTail(pInfo);
        }
    }
}
```

5) 添加CutSelections ( )函数。既然剪切和拷贝非常类似，因此这个函数也只是简单地调用拷贝函数。我们将使用调用参数pList作为标志——若为NULL，则CopySelections ( )只是拷贝到剪贴板上。

```
void CWzdSelect::CutSelections(CList<CWzdInfo1*,
    CWzdInfo1*> *pList)
{
    CopySelections(pList);
}
```

CopySelections ( )函数既处理剪切又处理拷贝数据对象到剪贴板上。

2. 实现CopySelections ( )辅助函数

1) 添加CopySelections ( )函数。它的首要任务是当没有任何东西被选中时返回。

```
void CWzdSelect::CopySelections(CList<CWzdInfo1*,
    CWzdInfo1*> *pList /*=NULL*/)
{
    if (m_WzdSelectionList.GetCount() <= 0) return;
```

2) 其次，串行化选项列表到一个共享内存文件中。因为 CSharedFile被全局分配，而这正是我们所要求的，我们将把选项放到剪贴板中。

```
CSharedFile file;
CArchive ar(&file, CArchive::store);
Serialize(ar);
ar.Close();
```

3) 现在，把这个文件放到剪贴板中。我们使用已注册过的剪贴板类型，并把从内存文件中获得的句柄传给它，该句柄实际上是一个在全局堆中分配的内存句柄。

```
COleDataSource *pDS = new COleDataSource();
pDS->CacheGlobalData(m_clipboardFormat, file.Detach());
pDS->SetClipboard();
```

4) 请记住我们是让pList来标识是剪切还是拷贝。为了从文档中剪切所选内容，视图类会把一个指向文档数据对象列表的指针赋给这个参数。现在，我们把这些项目从文档中删除。

```
// if cutting, delete items in selection list from document
if (pList)
{
    for (POSITION pos=m_WzdSelectionList.GetHeadPosition();pos;)
    {
        CWzdInfo1 *pInfo=m_WzdSelectionList.GetNext(pos);
```

```
        POSITION posx;
        if (posx=pList->Find(pInfo))
        {
            pList->RemoveAt(posx);
        }
        delete pInfo;
    }
}
```

5) 最后，清空选项列表。

```
m_WzdSelectionList.RemoveAll();
m_pActiveSelection=NULL;
```

3. 粘贴二进制数据

1) 创建一个PasteClipboard ( )函数，它以指向文档数据对象列表的指针为参数。这个函数的首要任务是删除当前所选中的所有内容。

```
void CWzdSelect::PasteClipboard(CList<CWzdInfo1*,
    CWzdInfo1*> *pList)
{
    // reset selections
    m_WzdSelectionList.RemoveAll();
    m_pActiveSelection=NULL;
```

2) 其次，捆绑一个数据对象到剪贴板，并把数据串行化返回到选项列表。

```
COleDataObject object;
object.AttachClipboard();
CFile* pFile = object.GetFileData(m_clipboardFormat);
if (!pFile) return;
CArchive ar(pFile, CArchive::load);
Serialize(ar);
ar.Close();
delete pFile; //deletes file
```

3) 最后，我们把这些选择内容加入到应用程序文档中。

```
for (POSITION pos=m_WzdSelectionList.GetHeadPosition();pos;)
{
    m_pActiveSelection=m_WzdSelectionList.GetNext(pos);
    pList->AddTail(m_pActiveSelection);
}
```

该选择类的详细清单参见本节的“清单——选择类”。

4. 实现新的选择类

1) 在用户视图类中嵌入CSelection类。

```
//////////////////////////
// WzdView.h
xprivate:
    CWzdSelect m_select;
```

2) 调用CSelection类的不同成员函数以处理选择、剪切、拷贝和粘贴等菜单命令。

```
void CWzdView::OnSelect()
{
```

```
    m_select.Select(GetDocument()->GetInfo1List()->GetHead());
}
void CWzdView::OnEditCut()
{
    m_select.CutSelections(GetDocument()->GetInfo1List());
}
void CWzdView::OnUpdateEditCut(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_select.SelectionCount());
}
void CWzdView::OnEditCopy()
{
    m_select.CopySelections();
}
void CWzdView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_select.SelectionCount());
}
void CWzdView::OnEditPaste()
{
    m_select.PasteClipboard(GetDocument()->GetInfo1List());
}
void CWzdView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_select.CanPasteClipboard());
}
```

说明

- 若想粘贴数据到文档的某一特定位置处，则需要深入设计 PasteClipboard ( )。对于图形图像，这样做可能是不必要的，因为一个图形的 X、Y位置比其在文档数据列表中位置重要得多。然而，当粘贴数据到数据库时，数据库项列表可能需一些特殊的考虑。
- 对于选择图形项目，可以使用 MFC的CTracker类在视图中突出显示所选项目。

CD说明

在CD上运行该工程时，首先在 WzdSelect.cpp.中的每个函数上设置断点。然后单击菜单上的Test/Wzd命令填充文档。最后，单击不同的 EditZ菜单命令，观察文档数据的剪切、拷贝和粘贴。

清单 —— 选择类

```
#ifndef WZDSELECT_H
#define WZDSELECT_H

#include "afxtempl.h"
#include "WzdInfo1.h"

class CWzdSelect : public CObject
{
```

```
public:
    CWzdSelect();
    ~CWzdSelect();

    void Select(CWzdInfo1 *pInfo);
    int SelectionCount();
    void CutSelections(CList<CWzdInfo1*,CWzdInfo1*> *pList);
    void CopySelections(CList<CWzdInfo1*,CWzdInfo1*> *pList=NULL);
    BOOL CanPasteClipboard();
    void PasteClipboard(CList<CWzdInfo1*,CWzdInfo1*> *pList);
    void Serialize(CArchive& archive);

private:
    int m_clipboardFormat;
    CWzdInfo1 *m_pActiveSelection;
    CList<CWzdInfo1*,CWzdInfo1*> m_WzdSelectionList;

};
#endif

// WzdSelect.cpp : implementation of the CWzdSelect class
//

#include    "stdafx.h"
#include <afxadv.h>
#include <afxole.h>
#include "WzdSelect.h"

////////////////////////////////////////////////////////////////////////////
// CWzdSelect

CWzdSelect::CWzdSelect()
{
    m_pActiveSelection=NULL;
    m_clipboardFormat = ::RegisterClipboardFormat("CWzdInfo1");
}

CWzdSelect::~CWzdSelect()
{
}
////////////////////////////////////////////////////////////////////
//CWzdSelect Methods

void CWzdSelect::Serialize(CArchive& ar)
{
    int nCount;
    CObject::Serialize(ar);
    if(ar.IsStoring())
    {
        nCount = m_WzdSelectionList.GetCount();
```

```
        ar << nCount;
        for (POSITION pos = m_WzdSelectionList.GetHeadPosition(); pos;)
        {
            m_WzdSelectionList.GetNext(pos)->Serialize(ar);
        }
    }


    else
    {
        ar >> nCount;
        while (nCount-- > 0)
        {
            CWzdInfo1* pInfo = new CWzdInfo1;
            pInfo->Serialize(ar);
            m_WzdSelectionList.AddTail(pInfo);
        }
    }
}

void CWzdSelect::Select(CWzdInfo1 *pInfo)
{
    m_pActiveSelection=pInfo;
    if (!m_WzdSelectionList.Find(pInfo))
    {
        m_WzdSelectionList.AddTail(pInfo);
    }
}

int CWzdSelect::SelectionCount()
{
    return m_WzdSelectionList.GetCount();
}

void CWzdSelect::CutSelections(CList<CWzdInfo1*,CWzdInfo1*> *pList)
{
    CopySelections(pList);
}

void CWzdSelect::CopySelections(CList<CWzdInfo1*,CWzdInfo1*> *pList /*=NULL*/)
{
    if (m_WzdSelectionList.GetCount() <= 0) return;

    // create an archive to a memory file and selection list to it
    CSharedFile file;
    CArchive ar(&file, CArchive::store);
    Serialize(ar);

    // close archive and put it in clipboard
    ar.Close();
    COleDataSource *pDS = new COleDataSource();
    pDS->CacheGlobalData(m_clipboardFormat, file.Detach());
    pDS->SetClipboard();
```

```
    // if cutting, delete items in selection list from document
    if (pList)
    {
        for (POSITION pos=m_WzdSelectionList.GetHeadPosition();pos;)
        {
            CWzdInfo1 *pInfo=m_WzdSelectionList.GetNext(pos);

            POSITION posx;
            if (posx=pList->Find(pInfo))
            {
                pList->RemoveAt(posx);
            }
            delete pInfo;
        }
    }

    // kill selections
    m_WzdSelectionList.RemoveAll();
    m_pActiveSelection=NULL;
}

BOOL CWzdSelect::CanPasteClipboard()
{
    COleDataObject object;
    return (object.AttachClipboard() &&
        object.IsDataAvailable(m_clipboardFormat));
}

void CWzdSelect::PasteClipboard(CList<CWzdInfo1*,CWzdInfo1*> *pList)
{
    // reset selections
    m_WzdSelectionList.RemoveAll();
    m_pActiveSelection=NULL;

    // open archive to clipboard and serialize into selection list
    COleDataObject object;
    object.AttachClipboard();
    CFile* pFile = object.GetFileData(m_clipboardFormat);
    if (!pFile) return;
    CArchive ar(pFile, CArchive::load);
    Serialize(ar);
    ar.Close();
    delete pFile; //deletes file

    // add selection back into document
    for (POSITION pos=m_WzdSelectionList.GetHeadPosition();pos;)
    {
        m_pActiveSelection=m_WzdSelectionList.GetNext(pos);
        pList->AddTail(m_pActiveSelection);
    }
}
```

14.4 例77 数组函数

目标

在应用程序中，使用MFC的数组类组织数据。

策略

使用MFC的CArray类创建和维护数组。我们将使用 CArray的模板版，因为它是类型安全的，并且比其他几个特定类型的数组版本容易记。本例将包括一组 CWzdInfo类对象，但CArray可包括任何数据类型(如int, float等)。

步骤

1. 声明数组类

1) 要使用模板类，需要在代码中加入以下包含文件。

```
# include <afxtempl.h>
```

2) 要声明CArray类，需要提供两个参数：第一个参数是数组包括的对象类型；第二个参数是类型说明，这个说明当引用数组中的对象时使用。如果第一个参数指定为一个类，第二个参数应该为该类的参考。如果第一个参数是类的指针，则第二个参数应该为同样的指针。

```
CArray <CWzdInfo, CWzdInfo&> m_WzdClassArray;
```

3) 最后，告诉CArray使数组多大。

```
m_WzdClassArray.SetSize(10); //sets size of array
```

2. 使用数组类

1) 若想保存数组中一个完整的数据对象，而不是指向数据对象实例的指针，则需要在以下两步中，在数据类中加入一个拷贝构造函数 (若数据类没有拷贝函数时)。

2) 在数据类的声明文件中，加入如下语句。

```
CWzdInfo & operator = (CWzdInfo & src);
```

3) 在实现数据类的文件中，加入如下语句：

```
CWzdInfo& CWzdInfo::operator=(CWzdInfo& src)
{
   if(this != &src)
   {
      m_sName = src.m_sName;
   // repeat for every member variable in the class
      :  :  :
   }
   return *this;
}
```

4) 然后，存放一个完整的数据类到数组中只是一件简单的事情而已。

```
m_WzdClassArray[2]=info1;
m_WzdClassArray[3]=info2;
```

这里，info1和info2是CwzdInfo类的对象。

5) 查看数组。

```
for (int i=0;i<m_WzdClassArray.GetUpperBound();i++)
{
   info=m_WzdClassArray[i];
   /////

}
```

6) 从数组中删去项目。

```
info=m_WzdClassArray[3]; //remove class object at index 3
m_WzdClassArray.RemoveAt(3);
```

7) 销毁数组。

```
m_WzdClassArray.RemoveAll ( ) ;
```

或者，若用户在堆栈上声明数组，只要从一个函数中返回，将引发 CArray的析构函数释放数组中的所有数据对象。

8) 若用户声明的是一个对象指针数组而不是对象数组，则可使用如下语句销毁数组及其对象。

```
while (m_WzdPtrArray.GetUpperBound()>-1)
{
   delete m_WzdPtrArray[0];
   m_WzdPtrArray.RemoveAt(0);
}
```

说明

- 为什么要用CArray，而不是简单的array [ ]声明呢？CArray类提供了丰富的成员函数，这些成员函数可以用来增大或减小数组的大小、删除所选项，并在任意位置上插入一个新项。换句话说，若不使用CArray，这些函数你必须自己编写。
- 当以顺序索引的方式维护对象时，应该使用 CArray类而不是CList和CMap。若索引不是顺序的，或者甚至不是数字，相应地要使用 CMap。若不想使用索引来引用对象，要相应地使用CList来维护对象。
- 参阅以下两个关于CList和CMap的例子。

CD说明

在CD上运行该工程时，在 WzdView.cpp中的OnTestWzd ( )函数上设置断点。然后单击Test/Wzd菜单命令，观察数组类的使用。

14.5 例78 列表函数

目标

在应用程序中，使用MFC的链表类组织数据。

策略

使用MFC的CList类创建和维护链表。我们将使用模板版的CList类，因为它是类型安全的，

并且比其他几个特定数据类型的列表版本更容易记住。本例将收集一组 CWzdInfo类对象，但是CList也可收集任何数据类型(如int、float等)。

步骤

1. 声明自己的列表类

1) 要使用模板类，需要在代码中包括如下包含文件。

```
# include <afxtemp1.h>
```

2) 要声明CList类，需要提供两个参数：第一个参数是列表所收集的对象类型；第二个参数是类型说明，这个说明当引用列表中的对象时使用。若第一参数为一个类，第二个参数应该为该类的参考。若第一个参数为类的指针，则第二个参数为同样的指针。

```
CList <CWzdInfo, CWzdInfo &> m_WzdClassList；
```

2. 使用列表类

1) 若想在列表中保存一个完整的数据对象，而不是指向数据对象实例的指针，需要在以下两步中，在数据类中加入一个拷贝的构造函数 (若数据类没有拷贝函数时)。

2) 在数据类的声明文件中，加入如下语句。

```
CWzdInfo & operator=(CWzdInfo & src);
```

3) 在数据类的实现文件中，加入如下语句。

```
CWzdInfo& CWzdInfo::operator=(CWzdInfo& src)
{
   if(this != &src)
   {
      m_sName = src.m_sName;
      // repeat for every member variable in the class
      :   :   :
   }
   return *this;
}
```

4) 加入类对象到列表中。

```
m_WzdList.AddTail(info1); // adds to the tail of the list
m_WzdList.AddHead(info2); // adds to the head of the list
```

其中，info1和info2为CWzdInfo类对象。

5) 查看数组。

```
for (POSITION pos=m_WzdList.GetHeadPosition();pos;)
{
   info=m_WzdList.GetNext(pos);

   /////
}
```

6) 从列表中删除项目。

```
pos=m_WzdList.FindIndex(1); // find 2nd element in list (zero based)
m_WzdList.RemoveAt(pos);  // remove from class list
```

7) 销毁列表。

```
m_WzdClassList.RemoveAll ( );
```

或者，若在栈上声明列表，从函数中返回时将销毁该类，并将自动释放列表中的所有数

据对象。

8) 若声明的是一个对象指针列表而不是对象列表，则可使用如下代码销毁列表及其对象。

```
while (!m_WzdPtrList.IsEmpty())
{
   delete m_WzdPtrList.RemoveHead();
}
```

说明

使用CList维护一个不需要直接引用的对象集。否则，使用 CArray或CMap类。

CD说明

在CD上运行该工程时，在 WzdView.cpp中在OnTestWzd ( )函数上设置断点。然后单击Test/Wzd菜单命令，观察列表类的使用。

14.6 例79 映像函数

目标

在应用程序中使用数据映像功能来维护数据。一个数据映像是一个可被数字或字符串键参考的同类对象集。

策略

使用MFC的CMap类创建和维护一个数据映像，将使用 CMap的模板版本，因为它是类型安全的，并且比其他几个特定类型的 CMap版本更容易记住。本例子将收集 CWzdInfo类对象的两个映像，第一个具有数字键，第二个具有字符串键。然而， CMap可以包括具有任何类型键的任何数据类型(如int、float等)。

步骤

1. 声明映像类

1) 要使用模板类，需要在代码中加入如下包含文件。

```
# include <afxtempl.h>
```

2) 要声明CMap类，需要提供四个参数：头两个参数定义访问映像中对象的键的变量类型；后两个参数定义存放在映像中的数据对象的类型。在这两个参数对中，第一个参数是键或者对象的对象类型，第二个参数是当引用键或对象时模板类要使用的类型说明。若第一个参数为类，则第二个参数为类的参考。若第一个参数为类的指针，则第二个参数为同样的指针。如下声明了一个以整型键为参考的 CWzdInfo对象集的映像。

```
CMap <int, int, CWzdInfo, CWzdInfo&> m_WzdIntToClassMap;
```

3) 其次，我们再声明一个以CString作为键，引用CWzdInfo对象集的映像。

```
CMap <CString, LPCSTR, CWzdInfo, CWzdInfo&> m_WzdStringToClassMap;
```

说明部分解释了为什么在这里用 LPCSTR。

2. 使用映像类

1) 若想在映像中存放完整的数据对象，而不是指向数据类实例的指针，则需要在以下两步中，在数据类中加入拷贝构造函数 (若数据类没有拷贝函数)。

2) 在数据类的声明文件中，加入如下语句：

```
CWzdInfo& operator=(CWzdInfo& src);
```

3) 在数据类的实现文件中，加入如下语句：

```
CWzdInfo& CWzdInfo::operator=(CWzdInfo& src)
{
   if(this != &src)
   {
      m_sName = src.m_sName;
      // repeat for every member variable in the class
      :   :   :
   }
   return *this;
}
```

4) 加入类对象到前面声明的任何一个映像中。

```
m_WzdIntToClassMap[3]=info2;
m_WzdStringToPtrMap["these"]=info1;
```

其中，info1和info2是CWzdInfo类对象。

5) 查看以CString声明的映像。

```
CString str;
for (POSITION pos = m_WzdStringToClassMap.GetStartPosition();
   pos;)
{
   m_WzdStringToPtrMap.GetNextAssoc(pos,str,pInfo);

   // str contains key
   // pInfo contains pointer to data

}
```

用户可以用一个整型键来查看以整型为键的映像。

6) 从映像中删除项目。

```
m_WzdIntToClassMap.RemoveKey(3);
m_WzdStringToClassMap.RemoveKey("them");
```

7) 销毁映像。

```
m_WzdIntToClassMap.RemoveAll();
m_WzdStringToClassMap.RemoveAll();
```

然而，若在栈上声明映像，则不需要加入任何代码——和它们所维护其他对象一样，CMap类会自动地为用户销毁。

8) 若声明的是对象指针映像而不是对象映像，用户可以使用如下代码销毁映像及对象。

```
CWzdInfo *pInfo;

for (pos = m_WzdStringToPtrMap.GetStartPosition(); pos;)
{
   m_WzdStringToPtrMap.GetNextAssoc(pos,str,pInfo);
   m_WzdStringToPtrMap.RemoveKey(str);
   delete pInfo;
}
```

说明

- CMap实际上并非使用字符串键来实施查找，事实上，它把字符串转换为一个整型标识，正如字符串的Checksum，实施这种转换的例子要求在CMap声明中使用LPCSTR作参数，即使键的类型为CString。

```
CMap <CString, LPCSTR, CWzdInfo, CwzdInfo &> m_WzdStringToClassMap;
```

- 映像被自动地分成17个对象列表。当要查找对象时，使用你提供的键来实施二分法搜索，以确定17个列表中哪一个包含所找的对象。然后，在这个列表中进行简单的扫描查找对象。可以通过CMap的成员函数InitHashTable ( )把映像分成更多的列表。在更大的映像集中加快搜索速度。

```
m_WzdStringToPtrMap.InitHashTable (50);          // create 50 lists
```

然而，在数据类被加入到映像之前只能设置一次。

- MFC使用CMap集记录哪个CWnd类拥有哪个窗口对象。当你调用CWnd的FromHandle ( )成员函数时，MFC在映像中使用提供的hWnd键来查找CWnd对象。如果没有，MFC生成一个临时CWnd对象，它封装你所给的 hWnd句柄。当应用程序为空等待时，这个CWnd对象被销毁。

CD说明

在CD上运行该工程时，在 WzdView.cpp中的OnTestWzd上设置断点。然后单击 Test/Wzd菜单命令观察映像类的使用。

14.7 例80 系统键盘输入

目标

设置组合键来触发某一菜单命令。例如，用 Alt+D组合键删除视图中所选项目。

策略

首先，使用Developer Studio提供的加速键编辑器(Accelerator Editor)创建一项资源，当视图被创建、使用、拥有输入焦点时该资源被装入。在视图没有输入焦点时或当我们想定制键的处理时，我们也可以手工处理WM_KEYDOWN窗口命令。

步骤

1. 自动处理加速键

1) 在Workspace View中选择Resource View，打开Accelerator文件夹，双击当前的加速键资源打开加速键编辑器。

2) 要修改一个已存在的加速键，选取编辑列表的项目；要增加一个新键，选取列表底部的空框。指定想触发它的命令ID和组合键。单击关闭按钮完成改变。

3) 当应用程序被创建时，应用框架就会装入加速键表。

在某一给定时刻，只能有一个加速键表被调出。然而，也可以手工通过重载窗口的PreTranslateMessage ( )函数为某一窗口重载加速键表。

在PreTranslateMessage ( )中，在加速键表得到响应之前，有机会转换组合键为一个菜单命令(正如PreTranslateMessage( )名字所暗示的一样)。在这个函数中，可以找到WM_KEYDOWN或WM_SYSKEYDOWN窗口消息。若所转换的组合键包含 Alt键或Shift-Alt键的组合，则应该查找 WM_SYSKEYDOWN消息。所有其他组合键 (包括Ctrl-Alt)将出现在WM_KEYDOWN消息中。

2. 手工处理Alt和Shift-Alt组合键

1) 使用ClassWizard，在控制输入窗口(通常为视图)的类中重载PreTranslateMessage ( )函数。

2) 为了处理Alt和Shift-Alt组合键，在PreTranslateMessage ( )函数中加入如下代码。

```
if (pMsg->message==WM_SYSKEYDOWN)
{
   switch(pMsg->wParam)
   {
      case VK_DELETE:
         SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
         return TRUE; // translated

      case VK_INSERT:
         SendMessage(WM_COMMAND,ID_EDIT_PASTE);
         return TRUE; // translated
   }
}

return CView::PreTranslateMessage(pMsg);
```

3) 若想判断是否同时按下Shift键，使用如下语句。

```
BOOL bShift=::GetKeyState (VK_SHIFT)&0x8000;
```

3. 手工处理所有其他组合键。

1) 为了处理所有其他组合键包括Ctrl、Ctrl-Alt和Shift-Ctrl-Alt，在PreTranslateMessage ( )函数中加入如下代码。

```
 if (pMsg->message==WM_KEYDOWN)
   {
      switch(pMsg->wParam)
      {
         case VK_DELETE:
            SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            return TRUE; // translated

         case VK_INSERT:
            SendMessage(WM_COMMAND,ID_EDIT_PASTE);
            return TRUE; // translated
      }
   }
   return CView::PreTranslateMessage(pMsg);
}
```

2) 若想判断是否同时按下Shift、Ctrl或者Alt键，使用如下语句之一。

```
BOOL bCtrl=::GetKeyState(VK_CONTROL)&0x8000;
BOOL bShift=::GetKeyState(VK_SHIFT)&0x8000;
BOOL bAlt=::GetKeyState(VK_MENU)&0x8000;
```

若要了解重载的 PreTranslateMessage ( )函数的详细情况，请参见本节的“清单 ——

PreTranslateMessage ( )重载函数 ”。

说明

本例的另外一种方法可能就是在窗口中添加 WM_KEYDOWN或WM_SYSKEYDOWN的消息处理函数。然而，这里所指的方法允许重载加速键列表中任何定义。正如其名字所暗示的一样，在系统使用加速键列表转换键盘输入到命令消息之前， PreTranslateMessage ( )函数被调用。任何成功的输入转换被丢弃，因此使得这种输入无法到达 WM_KEYDOWN或者WM_SYSKEYDOWN消息处理函数。

CD说明

在CD上运行该工程时，在WzdView.cpp中的PreTranslateMessage ( )函数上设置断点。然后使用Control、Shift和Alt键，按下不同的组合键观察它们的处理。

清单——PreTranslateMessage ( )重载函数

```
BOOL CWzdView::PreTranslateMessage(MSG* pMsg)
{

   // ALT key is not pressed or is pressed with CTRL key....
   if (pMsg->message==WM_KEYDOWN)
   {
      BOOL bCtrl=::GetKeyState(VK_CONTROL)&0x8000;
      BOOL bShift=::GetKeyState(VK_SHIFT)&0x8000;

      // only gets here if CTRL key is pressed
      BOOL bAlt=::GetKeyState(VK_MENU)&0x8000;
      switch(pMsg->wParam)
      {
         case 'N':
            if (bAlt&&bShift) //&&bCtrl assumed
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            else if (bShift&&bCtrl)
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            else if (bAlt) //&&bCtrl assumed
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            else if (bCtrl)
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            else if (bShift)
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            else
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            return TRUE; // translated

         case VK_ESCAPE:
            if (bShift)
               SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
            return TRUE; // translated
```

```
            case VK_DELETE:
                SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
                return TRUE; // translated

            case VK_INSERT:
                SendMessage(WM_COMMAND,ID_EDIT_PASTE);
                return TRUE; // translated
        }
    }
    // ALT key is pressed but not with CTRL key....
    else if (pMsg->message==WM_SYSKEYDOWN)
    {
        BOOL bShift=::GetKeyState(VK_SHIFT)&0x8000;
        switch(pMsg->wParam)
        {
            case 'N':
                if (bShift)
                    SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
                else
                    SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
                return TRUE; // translated


            case VK_DELETE:
                SendMessage(WM_COMMAND,ID_EDIT_CLEAR);
                return TRUE; // translated

            case VK_INSERT:
                SendMessage(WM_COMMAND,ID_EDIT_PASTE);
                return TRUE; // translated
        }
    }

    return CView::PreTranslateMessage(pMsg);
}
```

14.8 例81 时间

目标

获取、操纵时间，并以各种不同的格式显示。

策略

简单讨论MFC所提供的不同时间类及其功能。

步骤

1) 获取当前时间。

```
CTime time;
time=CTime::GetCurrentTime();
```

2) 获取时间元素。

```
int year = time.GetYear();
int month = time.GetMonth();
int day = time.GetDay();
int hour = time.GetHour();
int minute = time.GetMinute();
int second = time.GetSecond();
int DayOfWeek = time.GetDayOfWeek();
```

3) 获取时间间隔。

```
CTimeSpan timespan(0,0,1,0); // days,hours,minutes,seconds
timespan = CTime::GetCurrentTime() - time;
```

4) 把时间转换为字符串。

```
CString sDate,sTime,sElapsedTime;
sDate = time.Format("%m/%d/%y");                    //ex: 12/10/98
sTime = time.Format("%H:%M:%S");                    //ex: 9:12:02
sElapsedTime = timespan.Format("%D:%H:%M:%S");
   // %D is total elapsed days
```

要想知道更多的时间格式，参见MFC文档中的strftime。

使用COleDateTime类

1) 获得一年中的某一天。

```
COleDateTime datetime;
datetime = COleDateTime::GetCurrentTime();
int DayOfYear = datetime.GetDayOfYear();
```

2) 从文本串中读取时间。

```
COleDateTime datetime;
datetime.ParseDateTime("12:12:23 27 January 93");
```

说明

- CTime和COleDateTime具有几乎同样的功能。然而，COleDateTime允许用户获得一年中的某一天(创建Julian日期的一种好方法)，以及分析一个时间文本串。
- 与CTime相比，COleDateTime的优点在于它支持DWORD变量。COleDateTime使用的位数是双浮点的两倍，既然CTime只是简单地计算从1970年1月1日之后经过的秒数，所以到了2037年它将达到4294967295，从而不能再使用。相反，COleDateTime是一个浮点数，它表示是从1900年12月30号之后的天数(小时是天的小数部分)，几千年之内不会溢出。

CD说明

在CD上运行该工程时，在WzdView.cpp中的OnWzdType ( )函数中设置断点。然后，单击Options/Wzd菜单命令，观察时间操作。