```
ON_LBN_SELCHANGE(IDC_LISTBOX, OnSelChange)
```

在 Print Sample 按钮被单击时,OnPushButtonClicked 被激活。由于在 Windows 应用程序中打印是很复杂的工作,因此 OnPushButtonClicked 除了显示一个消息框外什么也不做。OnCheckBoxClicked 处理从复选框来的 BN_CLICKED 通知。因为复选框样式包含了 BS_AUTOCHECKBOX标志,在响应按钮单击时复选标记会自动切换。OnCheckBoxClicked 的工作是每当复选标记切换时都刷新列表框中的内容。它将调用 CMainWindow::FillListBox 来重新初始化列表框然后调用 CMainWindow::OnSelChange 来更新示例文本。

在列表框选项被修改时也调用 OnSelChange。它调用 GetCurSel 来获取当前被选项目的索引号。如果 GetCurSel 返回 LB_ERR,表明什么也没选,OnSelChange 将使按钮无效并清除示例文本。否则使按钮有效,用 CListBox::GetText 来检索被选中的文本并创建用 GetText 所返回字符串指定的字体。然后将字体赋给静态控件并设置控件文本"AaBbCcDdEeFfGg"。

字体枚举和回调函数

用字体名字填写列表框的工作是由 CMainWindow::FillListBox 完成的。当程序启动时,FillListBox 被 OnCreate 调用来初始化列表框。并且在"Show TrueType Fonts Only"复选框被单击时由 OnCheckBoxClicked 调用来对列表框进行重新初始化。FillListBox 首先调用 CListBox::ResetContent 来清除列表框。然后枚举系统中已安装的所有字体并给列表框添加相应的字体名称。

FillListBox 开始枚举处理的过程如下:首先构造名为 dc 的设备描述表对象,使用 CDC 类的 HDC 运算符来提取设备描述表的句柄,并将句柄传递给::EnumFontFamilies 函数:

```
CClientDC dc(this);
::EnumFontFamilies((HDC) dc, NULL, (FONTENUMPROC) EnumFontFamProc,
    (LPARAM) this);
```

第二个参数为 NULL 告诉::EnumFontFamilies 枚举所有安装了的字体。下一个参数是回调函数的地址。回调函数是在应用程序中 Windows 用您请求的信息回调的函数。对于每一个::EnumFontFamilies枚举的字体,Windows 都调用回调函数一次。::EnumFontFamilies 回调函数的原型必须像下面给出的这样:

```
int CALLBACK EnumFontFamProc(ENUMLOGFONT * lpelf,
    NEWTEXTMETRIC * lpntm, int nFontType, LPARAM lParam)
```

lpelf 是指向 ENUMLOGFONT 结构的指针,包含着关于字体的丰富信息,其中包括字体名称。lpntm 是指向类型 NEWTEXTMETRIC 结构的指针,包含着字体度量值:高度、平均字符宽度等等。nFontType 指定字体类型。TrueType 字体由具有 TRUETYPE_FONTTYPE 值的 ANDing nFontType 来标识。如果结果非零,字体就是 TrueType 字体。第四个即最后一个参数 lParam 是传递给::EnumFontFamilies 的任意 32 位 LPARAM 值。FillListBox 传递代表 CmainWindow 的

this 指针，其中原因待会儿解释。

FontView 的回调函数是 CMainWindow 的一个成员。实际上，是回调函数而不是 FillListBox 给列表框添加了字体名字。在每次 CMainWindow::EnumFontFamProc 被调用时，都将从 FillListBox 传递来的 lParam 值强制转换为 CMainWindow 指针：

```
CMainWindow* pWnd = (CMainWindow*) lParam;
```

它然后使用该指针来给列表框添加字体名字，并且只有在“Show TrueType Fonts Only”复选框未被选中或字体是 TrueType 字体时才行：

```
if ((pWnd->m_wndCheckBox.GetCheck() == BST_UNCHECKED) ||
    (nFontType & TRUETYPE_FONTTYPE))
    pWnd->m_wndListBox.AddString(lpelf->elfLogFont.lfFaceName);
return 1;
```

非零返回值告诉 Windows 继续进行枚举操作。（回调函数通过返回 0 值可以随时暂停操作，如果您分配了固定大小的内存来保存字体信息并且内存已满时就可以很方便地用到此功能。）在 Windows 最后一次调用了 EnumFontFamProc 以后，FillListBox 中引发::EnumFontFamilies 的调用也将返回枚举过程完毕。

为什么 FillListBox 将 this 指针传递给回调函数，又为什么作为 CMainWindow 的成员 EnumFontFamProc 还要将指针强制转换为 CMainWindow 指针？在 FontView.h 中仔细看一下 CMainWindow 的声明，会发现 EnumFontFamProc 是一个静态成员函数。静态类成员函数不接受 this 指针，因此它不能访问自己类中的非静态成员。要调用 m_wndCheckBox 的 GetCheck 函数和 m_wndListBox 的 AddString 函数，EnumFontFamProc 需要指向 m_wndCheckBox 和 m_wndListBox的指针或者指向那些对象所属的 CMainWindow 对象的指针。通过将传递给 FillListBox的 lParam 值强制转换为 CMainWindow 指针，EnumFontFamProc 就可以像非静态成员函数那样访问 CMainWindow 类的非静态成员了。

EnumFontFamProc 为静态的原因是由于回调函数在 C++ 应用程序中要求专门的处理。Windows 严格地定义了回调函数的接口，即通过参数列表传递的参数。当 C++ 类的成员函数被声明时，编译程序会自动附加一个额外的参数来保存 this 指针。不幸的是，附加了参数就使回调函数的参数列表与 Windows 期望的参数列表不匹配了，由此会导致各种问题出现，甚至包括无效内存访问错误，是 Windows 程序员的灾难。有几种解决此问题的方法，而将回调函数声明为静态成员函数是其中最简单最直接的一种。在 C++，不给静态成员函数传递 this 指针，因此它的参数列表不会改变。

在 Windows 中回调函数是很常见的，所以这里讲述的技巧不仅仅是对枚举字体有用。许多依靠回调函数的 Windows API 函数都支持应用程序定义的 lParam 值，这个值正好用来传递 this 指针给静态声明的回调函数。如果需要使用不支持应用程序定义 lParam 的枚举函数，那就不得不求助其他方法来使指针有效了。一种方法是通过将 this 指针复制为全局变

量使它对回调函数是可见的。

7.1.5 CEdit 类

MFC 的 CEdit 类封装了编辑控件的功能。编辑控件用来进行文本输入和编辑,可分为两种类型:单行和多行。单行编辑控件最好用来要求输入单行文本字符串,例如名字、口令以及产品 ID。(参见图 7-5。)想看多行编辑控件的例子,可以打开 Windows 中的"记事本"应用程序。"记事本"窗口的客户区就是一个多行编辑控件。

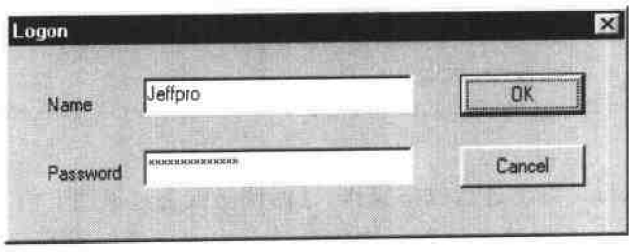

图 7-5 具有两个单行编辑控件的对话框

编辑控件中只能输入 60KB 的文本。对于单行编辑控件,这种限制还不算严格,而对于多行编辑控件,这就有点儿不够了。如果需要处理大量的文本就要使用丰富编辑控件,它是通用控件库中标准编辑控件的增强版本。虽然它被设计用来处理那些在字处理程序中出现的多格式类型的文本,但丰富编辑控件也能够处理一般的文本。Windows 中的"写字板"应用程序就使用了丰富编辑控件来输入和编辑文本。在第 12 章中会学习到用丰富编辑控件来创建类似"写字板"那样的应用程序。

创建编辑控件

若 m_wndEdit 是 CEdit 对象,语句

```
m_wndEdit.Create (WS_CHILD|WS_VISIBLE|WS_BORDER |
    ES_AUTOHSCROLL, rect, this, IDC_EDIT);
```

将创建一个单行编辑控件,在插入符移动超出控件边界时它可以自动水平滚动。在窗口样式中包括 ES_MULTILINE 会创建多行编辑控件:

```
m_wndEdit.Create (WS_CHILD|WS_VISIBLE|WS_BORDER |
    WS_HSCROLL|WS_VSCROLL|ES_MULTILINE, rect, this, IDC_EDIT);
```

WS_HSCROLL 和 WS_VSCROLL 给控件添加水平和垂直滚动条。可以使用 CEdit::SetRect 或 CEdit::SetRectNP 来定义控件的可编辑范围,它独立于控件的边界。这些函数的用途之一就是定义页尺寸,即使控件被缩放,其大小也保持不变。还可以使用 CEdit::

SetMargins 以像素为单位指定左右边宽度。默认边宽度为 0。下页表中列出了编辑控件专用的窗口样式。

在第一次创建之后，编辑控件可以接受大约 30 000 个字符。可以用 CEdit::LimitText 或 Win32 专用的 CEdit::SetLimitText 来增加和减少限制的数量。下列语句将编辑控件接受的最多字符数设置为 32：

```
m_wndEdit.SetLimitText (32);
```

在多行编辑控件中使用时，SetLimitText 限制的是输入控件中文本的总量而不是每行的长度。在多行编辑控件中没有自动的方式来限制每行的字符数，但可以手工编程来实现。一种方法是用 SetFont 将编辑控件的字体设置为固定调距字体，并用 CEdit::SetRect 指定一个格式化的矩形，它的宽度稍微比每行中期望的字符总宽度大一些。如表 7-9 所示。

表 7-9　编 辑 控 件

样式	说　明
ES_LEFT	在控件中左对齐文本
ES_CENTER	在控件中文本居中
ES_RIGHT	在控件中右对齐文本
ES_AUTOHSCROLL	允许编辑控件水平滚动但没有水平滚动条。想添加水平滚动条，就要包含 WS_HSCROLL 样式
ES_AUTOVSCROLL	允许编辑控件水平滚动但没有垂直滚动条。想添加垂直滚动条，就要包含 WS_VSCROLL 样式
ES_MULTILINE	创建多行编辑控件
ES_LOWERCASE	用小写字母显示所有字符
ES_UPPERCASE	用大写字母显示所有字符
ES_PASSWORD	显示星号通配符代替输入的字符
ES_READONLY	创建文本不能被编辑的编辑控件
ES_NOHIDESEL	在控件失去输入焦点时防止编辑控件隐藏所选内容
ES_OEMCONVERT	对输入控件的所有字符进行 ANSI 到 OEM 再到 ANSI 的转换，以便应用程序在进行自己的 ANSI 到 OEM 的转换时不会得到不希望的结果。已过时
ES_WANTRETURN	对于用在对话框中的多行编辑控件要通过回车键而不是调用默认的按钮来插入换行符

另一个有时用来初始化编辑控件的函数是 CEdit::SetTabStops，它可以设置制表位之间的距离。默认制表位间具有 8 个字符宽。可以任意设定制表位间的距离还可以使它们彼此间的距离不等。如同 CListBox::SetTabStops，CEdit::SetTabStops 也用对话框单位来测量距离。

插入和检索文本

用 SetWindowText 在编辑控件中插入文本,用 GetWindowText 来检索文本。CEdit 从基类 CWnd 中继承了两个函数。语句

```
m_wndEdit.SetWindowText (_T ("Hello, MFC"));
```

将文本字符串"Hello, MFC"插入编辑控件 m_wndEdit,而

```
m_wndEdit.GetWindowText (string);
```

检索名为 string 的 CString 对象中的文本。对于单行和多行编辑控件,GetWindowText 和 SetWindowText 都有效。用 SetWindowText 插入的文本会取代已存在的文本。即使文本在编辑控件中占据了几行,GetWindowText 也可以返回所有的文本。用空字符串调用 SetWindowText 可以删除编辑控件中所有的文本:

```
m_wndEdit.SetWindowText ( _T (""));
```

用 CEdit::ReplaceSel 可以在编辑控件中插入文本而不删除已有的内容。如果在调用 ReplaceSel 时一个或多个字符被选中,插入的文本将取代被选中的文本,否则文本在当前插入符位置处被插入。

多行编辑控件会自动插入换行符。如果想知道在多行编辑控件中检索得到的文本中换行符的位置,就要在调用 GetWindowText 之前使用 CEdit::FmtLines 使软换行符生效:

```
m_wndEdit.FmtLines (TRUE);
```

使用了软换行符,每行就被两个回车符(13)和跟随的换行符(10)定界了。要使软换行符无效,可以用 FALSE 来调用 FmtLines:

```
m_wndEdit.FmtLines (FALSE);
```

现在在任何方式下都不会表示出换行符了。无论 FmtLines 设置如何,硬回车(用户按下回车键时手动输入的换行符)以单个回车符/换行符对表示。FmtLines 不影响多行编辑控件中文本的外观。它只影响控件在内部保存文本的方式以及用 GetWindowText 检索得到的文本的格式。

要从多行编辑控件中仅读一行文本,可以使用 CEdit::GetLine。GetLine 将一行的内容复制到您提供的缓冲区中。行用从 0 开始的索引号标识。语句

```
m_wndEdit.GetLine (0, pBuffer, nBufferSize);
```

将多行编辑控件中第一行文本复制到由 pBuffer 所指的缓冲区中,第三个参数是缓冲区大小,以字节为单位(不是以字符)。GetLine 返回复制到缓冲区中的字节数。在用 CEdit::LineLength 检索一行之前可以确定所需要的缓冲区空间。通过调用 CEdit::GetLineCount 还

可以确定多行编辑控件中所含文本的行数。注意 GetLineCount 从不返回 0 值，即使在没有文本输入的情况下返回的是 1。

清除、剪切、复制、粘贴以及撤消

CEdit 提供了一些便于使用的成员函数，它们的功能与“编辑”菜单中的“清除”、“剪切”、“复制”、“粘贴”以及“撤消”命令相同。语句

```
m_wndEdit.Clear();
```

在不影响剪贴板中内容的情况下将所选文本清除。语句

```
m_wndEdit.Cut();
```

清除所选文本并将其复制到剪贴板。语句

```
m_wndEdit.Copy();
```

将所选文本复制到剪贴板并不改动编辑控件中的内容。

可以调用 CEdit::GetSel 来查询编辑控件中当前所选的内容，该函数返回具有两个压缩 16 位整数的 DWORD 值，分别指定所选内容开头和结尾字符的索引号。GetSel 的另一种形式是将索引号复制给一对整数，而整数的地址通过引用传递。如果索引号相等，说明当前没有文本被选中。下列 IsTextSelected 函数，您可能会在从 Cedit 派生来的编辑控件类中用到，如果有选中内容则返回非零值，否则返回零值：

```
BOOL CMyEdit::IsTextSelected()
{
    int nStart, nEnd
    GetSel(nStart, nEnd);
    return (nStart != nEnd);
}
```

如果没有选中文本 CEdit::Cut 和 CEdit::Copy 什么也不做。

文本选择可以用 CEdit::SetSel 编程实现。语句

```
m_wndEdit.SetSel(100, 150);
```

选中从第 101 个（从 0 开始索引号为 100 的字符）开始的 50 个字符，如果不可见就将所选内容滚动到视图中。要防止滚动，可以包含第三个参数并将其设置为 TRUE。

在多行编辑控件中编程选择文本时，通常需要将行号和行中可能的偏移转换为可以传递给 SetSel 的索引号。CEdit::LineIndex 接受基于 0 的行号并返回该行中第一个字符的索引号。在下一个例子中使用了 LineIndex 来确定多行编辑控件中第八行第一个字符的索引号，LineLength 检索行的长度，SetSel 则选中整行：

```
int nStart = m_wndEdit.LineIndex(7);
int nLength = m_wndEdit.LineLength(nStart);
m_wndEdit.SetSel(nStart, nStart + nLength);
```

CEdit 还提供了一个名为 LineFromChar 的函数用来根据字符索引号计算行号。

CEdit::Paste 将文本粘贴到编辑控件中。下列语句将当前保存在剪贴板中的文本粘贴到名为 m_wndEdit 的编辑控件中:

```
m_wndEdit.Paste();
```

如果剪贴板中没有文本,那么 CEdit::Paste 什么也不做。在调用 Paste 时,如果没有选中文本,剪贴板中的文本将在当前插入记号所在位置插入。如果有选中内容,从剪贴板中得到的文本将替换控件中选中的文本。通过调用::IsClipboardFormatAvailable 可以预先确定剪贴板中是否包含文本(由此也预先确定了 Paste 函数实际工作与否)。语句

```
BOOL bCanPaste = ::IsClipboardFormatAvailable(CF_TEXT);
```

在剪贴板中文本有效时将 bCanPaste 设置为非零值,否则设置为零。

编辑控件还具有内置的撤消功能,可以返回到上次编辑操作之前的状态。语句

```
m_wndEdit.Undo();
```

撤消上次操作,倘若该操作可以被撤消的话。通过 CEdit::CanUndo 可以预先决定调用 Undo 完成的工作。使用相关函数 CEdit::EmptyUndoBuffer 可以手动复位撤消标志,使得接下来对撤消的调用直到另一个编辑操作执行以前不进行任何工作(对 CanUndo 的调用将返回 FALSE)。

编辑控件通知

编辑控件给它们的父窗口发送消息来报告多种输入事件。在 MFC 应用程序中使用 ON_EN 消息映射宏将这些通知传递给了处理函数。表 7-10 中总结了编辑控件通知和相应的消息映射宏。

EN_CHANGE 通知一般用来当文本输入编辑控件时动态地更新其他控件。下列程序代码在有文本输入编辑控件(m_wndEdit,ID = IDC_EDIT)时更新按钮(m_wndPushButton),如果编辑控件至少包含一个字符,则使按钮有效,否则无效:

```
// In CMainWindow's message map
ON_EN_CHANGE(IDC_EDIT, OnUpdatePushButton)
    .
    .
    .
void CMainWindow::OnUpdatePushButton()
{
```

```
    m_wndPushButton.EnableWindow (m_wndEdit.LineLength ());
}
```

表 7-10　编辑控件通知

通知	发送条件	消息映射宏
EN_UPDATE	控件的文本将被修改	ON EN_UPDATE
EN_CHANGE	控件文本已修改	ON EN_CHANGE
EN_KILLFOCUS	编辑控件失去输入焦点	ON_EN_KILLFOCUS
EN_SETFOCUS	编辑控件接收到输入焦点	ON_EN_SETFOCUS
EN_HSCROLL	用滚动条将编辑控件水平滚动	ON EN_HSCROLL
EN_VSCROLL	用滚动条将编辑控件垂直滚动	ON EN_VSCROLL
EN_MAXTEXT	由于编辑控件已经包含了使用 CEdit::LimitText 或 CEdit::SetLimitText 指定的字符数因此不可以再输入字符了。如果插入符已处于控件格式化矩形的最右或最底端而控件又不支持滚动致使字符无法再输入时也发送此通知	ON EN_MAXTEXT
EN_ERRSPACE	由于内存不足而操作失败	ON EN_ERRSPACE

提供具有这种特点的交互反馈通常被认为是非常好的用户界面设计。与单击按钮接收到错误信息相比，恐怕大多数用户都愿意等待所有要求的信息被输入之后再使用按钮。

7.1.6　赶快！即时记事本

MyPad 应用程序，它的部分源代码如图 7-6 所示，使用多行编辑控件几乎创建了 Windows "记事本"应用程序的副本。从源代码中可以看到编辑控件完成了许多工作。CEdit 函数如 Undo 和 Cut 仅用了一行代码就实现了"编辑"菜单中的命令功能。

MyPad 是一个基于视图的应用程序，开始我使用了 MFC AppWizard，但在步骤 1 中没有选中"Document/View Architecture Support"栏。为避免不必要的代码，在 AppWizard 步骤 3 的对话框中我也没有选"ActiveX Controls"栏。在运行了 AppWizard 之后，用 Visual C++ 资源编辑器给"文件"菜单添加了"新建"命令并给"编辑"菜单添加了"删除"命令。还用资源编辑器给"新建"命令提供了加速键(Ctrl-N)。然后使用 ClassWizard 来添加命令处理程序、更新处理程序以及消息处理程序。

通过在名为 m_wndEdit 的 CEdit 数据成员上调用 Create，视图的 WM_CREATE 消息处理程序创建了编辑控件。OnCreate 将控件的宽度和高度设置为 0，而每当视图接收到 WM_SIZE 消息时 OnSize 都会将控件的尺寸重新设置使其填充视图的客户区域。第一个 WM_SIZE 消息在视图在屏幕上可见之前到达，以后的 WM_SIZE 消息则在每次 MyPad 窗口(以及

视图)被缩放时到达。视图类中的一行 WM_SETFOCUS 处理程序在视图收到输入焦点时就将焦点转移给编辑控件。

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__0FA1D288_8471_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_MAINFRM_H__0FA1D288_8471_11D2_8E53_006008A82731__INCLUDED_

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

#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
    CMainFrame();
protected:
    DECLARE_DYNAMIC(CMainFrame)

// Attributes
public:

// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
        AFX_CMDHANDLERINFO* pHandlerInfo);
    //}}AFX_VIRTUAL

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

// Generated message map functions
```

```
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg void OnSetFocus(CWnd * pOldWnd);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif
// !defined(AFX_MAINFRM_H__0FA1D288_8471_11D2_8E53_006008A82731__INCLUDED_)
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "MyPad.h"

#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_SETFOCUS()
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
```

```
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    cs.lpszClass = AfxRegisterWndClass(0);
    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}
void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnSetFocus(CWnd* pOldWnd)
{
    // forward focus to the view window
    m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
    AFX_CMDHANDLERINFO* pHandlerInfo)

{

    // let the view have first crack at the command
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    // otherwise, do default handling
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
        CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL))
    {
        TRACE0("Failed to create view window\n");
        return -1;
    }
    return 0;
}
```

ChildView.h

```
// ChildView.h : interface of the CChildView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDVIEW_H__0FA1D28A_8471_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_CHILDVIEW_H__0FA1D28A_8471_11D2_8E53_006008A82731__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CChildView window

class CChildView : public CWnd
{
// Construction
public:
    CChildView();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildView)
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
```

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

    // Generated message map functions
protected:
    BOOL IsTextSelected();
    CEdit m_wndEdit;
    //{{AFX_MSG(CChildView)
    afx_msg void OnPaint();
    afx_msg void OnEditCut();
    afx_msg void OnEditCopy();
    afx_msg void OnEditPaste();
    afx_msg void OnEditDelete();
    afx_msg void OnEditUndo();
    afx_msg void OnUpdateEditCut(CCmdUI * pCmdUI);
    afx_msg void OnUpdateEditCopy(CCmdUI * pCmdUI);
    afx_msg void OnUpdateEditPaste(CCmdUI * pCmdUI);
    afx_msg void OnUpdateEditDelete(CCmdUI * pCmdUI);
    afx_msg void OnUpdateEditUndo(CCmdUI * pCmdUI);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnFileNew();
    afx_msg void OnSetFocus(CWnd * pOldWnd);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif
// !defined(AFX_CHILDVIEW_H__0FA1D28A_8471_11D2_8E53_006008A82731__INCLUDED_)
```

ChildView.cpp

```
// ChildView.cpp : implementation of the CChildView class
//

#include "stdafx.h"
#include "MyPad.h"
#include "ChildView.h"
```

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

/////////////////////////////////////////////////////////////////////////////
// CChildView

CChildView::CChildView()
{
}

CChildView::~CChildView()
{
}

BEGIN_MESSAGE_MAP(CChildView,CWnd )
    //{{AFX_MSG_MAP(CChildView)
    ON_WM_PAINT()
    ON_WM_CREATE()
    ON_WM_SIZE()
    ON_WM_SETFOCUS()
    ON_COMMAND(ID_EDIT_CUT, OnEditCut)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_COMMAND(ID_EDIT_DELETE, OnEditDelete)
    ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
    ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateEditCut)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    ON_UPDATE_COMMAND_UI(ID_EDIT_DELETE, OnUpdateEditDelete)
    ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
    ON_COMMAND(ID_FILE_NEW, OnFileNew)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildView message handlers

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
```

```
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle |= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
        ::LoadCursor(NULL, IDC_ARROW), HBRUSH(COLOR_WINDOW + 1), NULL);

    return TRUE;
}
void CChildView::OnPaint()
{
    CPaintDC dc(this);
}

int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd ::OnCreate(lpCreateStruct) == -1)
        return -1;

    m_wndEdit.Create (WS_CHILD|WS_VISIBLE|WS_VSCROLL|ES_MULTILINE |
        ES_AUTOVSCROLL, CRect (0, 0, 0, 0), this, IDC_EDIT);
    return 0;
}

void CChildView::OnSize(UINT nType, int cx, int cy)
{
    CWnd ::OnSize(nType, cx, cy);
    m_wndEdit.MoveWindow (0, 0, cx, cy);
}

void CChildView::OnSetFocus(CWnd * pOldWnd)
{
    m_wndEdit.SetFocus ();
}

void CChildView::OnEditCut()
{
    m_wndEdit.Cut ();
}

void CChildView::OnEditCopy()
{
    m_wndEdit.Copy ();
```

```
}

void CChildView::OnEditPaste()
{
    m_wndEdit.Paste();
}

void CChildView::OnEditDelete()
{
    m_wndEdit.Clear();
}
void CChildView::OnEditUndo()
{
    m_wndEdit.Undo();
}

void CChildView::OnUpdateEditCut(CCmdUI * pCmdUI)
{
    pCmdUI -> Enable(IsTextSelected());
}

void CChildView::OnUpdateEditCopy(CCmdUI * pCmdUI)
{
    pCmdUI -> Enable(IsTextSelected());
}

void CChildView::OnUpdateEditPaste(CCmdUI * pCmdUI)
{
    pCmdUI -> Enable(::IsClipboardFormatAvailable(CF_TEXT));
}

void CChildView::OnUpdateEditDelete(CCmdUI * pCmdUI)
{
    pCmdUI -> Enable(IsTextSelected());
}

void CChildView::OnUpdateEditUndo(CCmdUI * pCmdUI)
{
    pCmdUI -> Enable(m_wndEdit.CanUndo());
}

void CChildView::OnFileNew()
{
    m_wndEdit.SetWindowText(_T(""));
}
```

```
BOOL CChildView::IsTextSelected()
{
    int nStart, nEnd;
    m_wndEdit.GetSel (nStart, nEnd);
    return (nStart != nEnd);
}
```

图 7-6 MyPad 应用程序

7.1.7 CComboBox 类

组合框将单行编辑控件和列表框组合为便于使用的一体结构。组合框有三种类型：简单型、下拉型和下拉列表型。图 7-7 给出了带有列表的下拉列表型组合框。

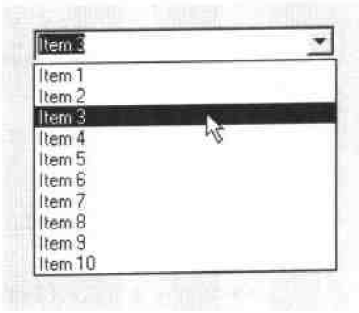

图 7-7 显示列表的下拉列表型

在三种组合框中，简单型组合框是最少用到的。简单型组合框的列表始终被显示。当用户从列表中选中一个项目时，该项目会自动复制到编辑控件中。用户也可以在编辑控件中直接输入文本。如果用户输入的文本与列表中某项目匹配，该项目就被自动加亮显示并被滚动到可见的地方。

下拉型组合框与简单型组合框不同之处就在于它的列表只有被要求时才显示。下拉型组合框的工作方式与简单型相同，但它不允许在编辑控件中输入文本。这样就有效地限制了用户只能选择列表框中的项目。

传递给 Create 或 CreateEx 的样式标志决定了所创建的组合框的类型。CBS_SIMPLE 创建一个简单型组合框，CBS_DROPDOWN 创建一个下拉型组合框，而 CBS_DROPDOWNLIST 创建一个下拉列表型组合框。其他样式控制组合框外观和功能的别的方面，如表7-11所示。其中许多样式看上去很熟悉，这是由于它们模仿了列表框和编辑控件的样式。例如：CBS_AUTOHSCROLL 对于组合框控件的编辑控件部分所执行的操作与 ES_AUTOHSCROLL 对于单独的编辑控件所执行的操作完全相同。在创建组合框控件时，想使列表框中具有垂直滚动条就别忘了包含 WS_VSCROLL 样式，同样想使控件的边框可见就要包括 WS_BORDER 样式。如果 m_wndComboBox 是 CComboBox 对象，语句

```
m_wndComboBox.Create (WS_CHILD|WS_VISIBLE|WS_BORDER |
    WS_VSCROLL|CBS_DROPDOWNLIST|CBS_SORT, rect, this,
    IDC_COMBOBOX);
```

创建一个下拉列表型组合框，它的列表框当其中的项目超出可显示的项目数时会包含垂直滚动条，而且还可以给加入的项目自动排序。调用 CComboBox::Create 时所指定的控件矩形框应该足够地大用来容纳控件的列表框部分以及控件的编辑框。

表 7-11　组合框样式

样式	说　明
CBS_AUTOHSCROLL	在组合框的编辑控件部分使用水平滚动条
CBS_DISABLENOSCROLL	在不需要时取消组合框的滚动条。没有此样式，不需要的滚动条会被隐藏而不是被取消
CBS_DROPDOWN	创建下拉型组合框
CBS_DROPDOWNLIST	创建下拉列表型组合框
CBS_HASSTRINGS	创建"记忆"加入字符串的组合框　默认时传统组合框具有此样式，而由所有者绘制的组合框没有
CBS_LOWERCASE	将组合框中所有文本转换为小写
CBS_NOINTEGRALHEIGHT	避免组合框的列表框高度必须具有项目高度的整数倍
CBS_OEMCONVERT	组合框中的编辑控件对所有字符都执行 ANSI 到 OEM 再到 ANSI 的转换，以便应用程序在执行自己的 ANSI 到 OEM 转换时不会得到意外的结果。已过时
CBS_OWNERDRAWFIXED	创建一个自制组合框其中项目具有相同的高度
CBS_OWNERDRAWVARIABLE	创建一个自制组合框其中项目高度不同
CBS_SIMPLE	创建简单型组合框
CBS_SORT	自动在项目加入时排序
CBS_UPPERCASE	将组合框中所有文本转换为大写

不必奇怪，CComboBox 成员函数列表与 CEdit 和 ClistBox 的成员函数列表非常相似。例如：用 CComboBox::AddString 和 CComboBox::InsertString 来给组合框添加项目，用 CComboBox::LimitText 来设置组合框编辑控件的最大字符数。CComboBox 从 CWnd 继承来的 GetWindowText 和 SetWindowText 函数可以获取和设置编辑控件中的文本。组合框独有的函数包括：GetLBText，用来检索从 0 开始的索引号所标识项目的文本；GetLBTextLen，返回项目的字符长度；ShowDropDown，隐藏或显示下拉列表框；以及 GetDroppedState，返回一个值表明下拉列表框是否已被显示。

组合框通知

组合框像编辑控件和列表框那样给它们的父窗口发送通知。表 7-12 列出了父窗口可能会得到的通知，相应的 MFC 消息映射宏，以及通知所适用的组合框类型。

表 7-12 组合框通知

通知	消息映射宏	简单型	下拉型	下拉列表型
CBN_DROPDOWN 在下拉列表显示时发送	ON_CBN_DROPDOWN		√	√
CBN_CLOSEUP 在下拉列表关闭时发送	ON_CBN_CLOSEUP		√	√
CBN_DBLCLK 在项目被双击时发送	ON_CBN_DBLCLK	√		
CBN_SELCHANCE 选项修改时发送	ON_CBN_SELCHANGE	√	√	√
CBN_SELENDOK 选中时发送	ON_CBN_SELENDOK	√	√	√
CBN_SELENDCANCEL 取消选择时发送	ON_CBN_SELENDCANCEL		√	√
CBN_EDITUPDATE 编辑控件中的文本将要更改时发送	N_CBN_EDITUPDATE	√	√	
CBN_EDITCHANGE 编辑控件中的文本已经修改时发送	ON_CBN_EDITCHANGE	√	√	
CBN_KILLFOCUS 组合框失去焦点时发送	ON_CBN_KILLFOCUS	√	√	√
CBN_SETFOCUS 组合框获得焦点时发送	ON_CBN_SETFOCUS	√	√	√
CBN_ERRSPACE 由于内存不足操作失败时发送	ON_CBN_ERRSPACE	√	√	√

并非所有通知都适用于每一个组合框类型。例如：CBN_DROPDOWN 和 CBN_CLOSEUP 通知就不会由 CBS_SIMPLE 组合框发送，因为简单型组合框的列表框不会打开和关闭。同样，CBS_DROPDOWN 和 CBS_DROPDOWNLIST 样式的组合框也不接收 CBN_DBLCLK 通知，因为它们列表框中的项目不可以被双击。(为什么？因为在第一次单击后列表框就关闭了。)CBN_EDITUPDATE 和 CBN_EDITCHANGE 通知与编辑控件发送的 EN_UPDATE 和 EN_CHANGE 通知等价，而且组合框中的 CBN_SELCHANGE 与列表框中的 LBN_SELCHANGE 相同。

在处理 CBN_SELCHANGE 通知时要注意到一点细微差别，即在通知到来时编辑控件可能还没来得及更新以实现与列表框中所选项目的匹配。因此，应该使用 GetLBText 而不是 GetWindowText 来检索最近选中的文本。可以用 CComboBox::GetCurSel 来获得选中项目的索引号。

7.1.8　CScrollBar 类

MFC 的 CScrollBar 类封装了由“SCROLLBAR”WNDCLASS 创建的滚动条控件。滚动条控件在许多方面与第 2 章 Accel 应用程序中使用的“窗口”滚动条相同。而窗口滚动条是通过在窗口样式中添加 WS_VSCROLL 和 WS_HSCROLL 标志创建的，滚动条控件却是明确地用 CScrollBar::Create 创建的。窗口滚动条可以放置在整个窗口客户区长度上并且本质上是属于窗口边框的，而滚动条控件可以放置在窗口的任何地方也可以设置任意的高度和宽度。

指定 SBS_VERT 样式可以创建垂直滚动条而指定 SBS_HORZ 可以创建水平滚动条。如果 m_wndVScrollBar 和 m_wndHScrollBar 是 CScrollBar 对象，语句：

```
m_wndVScrollBar.Create(WS_CHILD WS_VISIBLE|WS_BORDER |
    SBS_VERT, rectVert, this, IDC_VSCROLLBAR);
m_wndHScrollBar.Create(WS_CHILD|WS_VISIBLE|WS_BORDER |
    SBS_HORZ, rectHorz, this, IDC_HSCROLLBAR);
```

将创建两个滚动条控件，一个垂直一个水平。

用::GetSystemMetrics API 函数可以向 Windows 查询垂直滚动条的标准宽度或水平滚动条的标准高度。下列代码片段将 nWidth 和 nHeight 设置为系统标准滚动条的宽度和高度：

```
int nWidth = ::GetSystemMetrics(SM_CXVSCROLL);
int nHeight = ::GetSystemMetrics(SM_CYHSCROLL);
```

创建具有标准高度或宽度滚动条的另一种方法是在创建它时指定 SBS_TOPALIGN、SBS_BOTTOMALIGN、SBS_LEFTALIGN 或 SBS_RIGHTALIGN 样式。SBS_LEFTALIGN 和 SBS_RIGHTALIGN 沿着调用 Create 时所指定的矩形的左或右边来对齐垂直滚动条控件并使它具有标准宽度。SBS_TOPALIGN 和 SBS_BOTTOMALIGN 沿着矩形的顶或底边对齐水平滚动条并使它具有标准的高度。

与其他传统控件不同，滚动条控件不发送 WM_COMMAND 消息而是发送 WM_VSCROLL 和 WM_HSCROLL 消息。在第 2 章已经讲过，MFC 应用程序用 OnVScroll 和 OnHScroll 处理程序来处理这些消息，在第 2 章中没有提到两个滚动条通知代码，这是因为它们只适用于滚动条控件。SB_TOP 意味着在滚动条具有输入焦点时用户按下了 Home 键，SB_BOTTOM 则是按下了 End 键。

MFC 的 CScrollBar 类中包含一组处理滚动条的函数，对于其中的大多数您可能会比较熟悉，因为它们的功能与名为 CWnd 函数的功能相似。CScrollBar::GetScrollPos 和 CScrollBar::SetScrollPos 获取和设置滚动条滑块的位置。CScrollBar::GetScrollRange 和 CScrollBar::SetScrollRange 获取和设置滚动条的范围。可以使用 CScrollBar::SetScrollInfo 一步就设置好滚动条的范围、位置以及滑块大小。详细内容请参考第二章中对 CWnd::SetScrollInfo 的介绍。

7.2 高级控件程序设计

以 MFC 的方式进行控件程序设计的优点之一是通过从 MFC 控件类派生自己的类可以很方便地修改控件的功能。例如：可以很容易地创建一个只接受数字输入的编辑控件或显示图像而不是文本的列表框。还可以创建可重复使用响应它们自己通知消息的自含型控件类。

本章剩下的部分将讲述一些技术，用它们可以将 C++ 和 MFC 的长处结合起来生成自己想要的控件。

7.2.1 数字编辑控件

MFC 控件类本身就非常有用，因为它对内置的控件类型提供了面向对象的接口。但是它的用处还不止这些，您可以将它作为自己控件类的基类。通过给派生类添加消息处理程序或覆盖继承来的消息处理程序，可以修改控件的一部分功能而保留另一部分功能。

一个非常好的派生控件类的例子就是数字编辑控件。通常的编辑控件接受种类广泛的字符，包括数字、字母表中的字母以及标点符号。而数字编辑控件仅接受数值。适于用来输入电话号码、序列号、IP 地址以及其他数字数据。

由于 CEdit 中已定义了编辑控件的基本功能，所以在 MFC 应用程序中创建数字编辑控件并不很麻烦。要感谢 C++ 的继承和 MFC 的消息映射，您可以从 CEdit 中派生控件类并提供自定义的消息处理程序来修改控件对用户输入的响应。下列 CNumEdit 类创建一个编辑控件模型，该控件仅接受数字而拒绝所有其他字符：

```
class CNumEdit : public CEdit
{
protected:
    afx_msg void OnChar (UINT nChar, UINT nRepCnt, UINT nFlags);
    DECLARE_MESSAGE_MAP ()
};

BEGIN_MESSAGE_MAP (CNumEdit, CEdit)
    ON_WM_CHAR ()
END_MESSAGE_MAP ()

void CNumEdit::OnChar (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if (((nChar >= _T ('0')) && (nChar <= _T ('9'))) ||
        (nChar == VK_BACK))
            CEdit::OnChar (nChar, nRepCnt, nFlags);
    }
```

CNumEdit 是如何工作的呢？当编辑控件具有输入焦点且字符键被按下时，控件接收到 WM_CHAR 消息。从 CEdit 派生一个新类，将 WM_CHAR 消息映射到派生类的 OnChar 处理程序上，并要求当且仅当消息中编码的字符为数字时 OnChar 把 WM_CHAR 消息传递给基类，这样就创建了拒绝非数字字符的编辑控件。在可接受字符代码中包含了 VK_BACK 以便退格键还可以使用。没必要去检测其他编辑键如 Home 和 Del，因为它们和退格键不同，并不产生 WM_CHAR 消息。

7.2.2 自制列表框

默认情况下，列表框项目包含文本字符串。如果您需要一个显示图形而非文本的列表框，可以创建自制列表框（其中内容由应用程序绘制而不是由 Windows 绘制），按下列简单的两步操作即可。

1. 从 CListBox 派生新的列表框并覆盖 CListBox::MeasureItem 和 CListBox::DrawItem 覆盖 PreCreateWindow，并确保列表框样式中包括了 LBS_OWNERDRAWFIXED 和 LBS_OWNERDRAWVARIABLE。
2. 初始化派生类，并用 Create 或 CreateEx 来创建列表框。

就功能而言，自制列表框与由所有者绘制的菜单是相似的。当自制列表框中的项目需要绘制（或重画）时，Windows 将给列表框的父窗口发送一个 WM_DRAWITEM 消息，其中含有指向包含设备描述表句柄的 DRAWITEMSTRUCT 结构的指针，标识被画项目的索引号（从 0 开始）以及其他信息。在第一个 WM_DRAWITEM 消息到达之前，列表框的父窗口将接收到一个以上查询列表框项目高度的 WM_MEASUREITEM 消息。如果列表框样式为 LBS_OWNERDRAWFIXED，则 WM_MEASUREITEM 消息只发送一次。而要是 LBS_OWNERDRAWVARIABLE 列表框，WM_MEASUREITEM 消息对每个项目都要发送。MFC 在父窗口接收到 WM_DRAWITEM 消息时将调用虚拟 DrawItem 函数，而在接收到 WM_MEASUREITEM 时调用 MeasureItem。所以，可以不必去修改父窗口类或消息映射和消息处理程序，只要在列表框类中覆盖 DrawItem 和 MeasureItem，列表框就可以不需要父窗口的帮助而完成自己绘制的任务了。

除了 DrawItem 和 MeasureItem 以外，CListBox 还支持其他两个可覆盖函数。第一个是 CompareItem。如果自制列表框是用样式 LBS_SORT 创建的，并且用 AddString 给它添加项目，那么 CListBox::CompareItem 必须被覆盖，新的函数应该比较包含在 COMPAREITEMSTRUCT 结构中的两个任意项目。已覆盖函数的返回值必须是：如果项目 1 在项目 2 前面则返回 -1，如果项目在词典中位置相同则返回 0，如果项目 1 在项目 2 之后则返回 1。很少用 LBS_SORT 样式来创建自制列表框，因为非文本数据通常没有必然的顺序。（例如：您如何给一组颜色排序呢？）如果不使用 LBS_SORT，就没必要写 CompareItem 函数了。如果在派生的自制列表框类中不使用 CompareItem，就要小心覆盖 PreCreateWindow 并确保列表框样式不包含 LBS_SORT。

最后一个自制列表框的可覆盖函数是 DeleteItem。在下列情况中调用它：当用 DeleteString 删除一个项目时，用 ResetContent 删除列表框中的内容时，以及包含一个以上项目的列表框被销毁时。DeleteItem 对每个项目都调用一次，并接收一个指向包含项目信息的 DELETEITEMSTRUCT 结构的指针。如果列表框使用了单个项目资源(如：位图)，而这些资源需要在项目被删除或列表框被销毁时释放，覆盖并使用 DeleteItem 就可以完成此工作。

以下给出的 COwnerDrawListBox 类几乎就是 LBS_OWNERDRAWFIXED 样式的自制列表框的完整 C++ 实现：

```
class COwnerDrawListBox : public CListBox
{
public:
    virtual BOOL PreCreateWindow (CREATESTRUCT&);
    virtual void MeasureItem (LPMEASUREITEMSTRUCT);
    virtual void DrawItem (LPDRAWITEMSTRUCT);
};

BOOL COwnerDrawListBox::PreCreateWindow (CREATESTRUCT& cs)
{
    if (! CListBox::PreCreateWindow (cs))
        return FALSE;

    cs.style &= ~(LBS_OWNERDRAWVARIABLE|LBS_SORT);
    cs.style|= LBS_OWNERDRAWFIXED;
    return TRUE;
}

void COwnerDrawListBox::MeasureItem (LPMEASUREITEMSTRUCT lpmis)
{
    lpmis -> itemHeight = 32; // Item height in pixels
}

void COwnerDrawListBox::DrawItem (LPDRAWITEMSTRUCT lpdis)
{
    CDC dc;
    dc.Attach (lpdis -> hDC);
    CRect rect = lpdis -> rcItem;
    UINT nIndex = lpdis -> itemID;

    CBrush * pBrush = new CBrush (::GetSysColor ((lpdis -> itemState &
        ODS_SELECTED) ? COLOR_HIGHLIGHT : COLOR_WINDOW));
    dc.FillRect (rect, pBrush);
    delete pBrush;

    if (lpdis -> itemState & ODS_FOCUS)
        dc.DrawFocusRect (rect);

    if (nIndex != (UINT) -1) {
```

```
        // Draw the item.

    dc.Detach ();
}
```

把 COwnerDrawListBox 用在您自己的应用程序中以前，将 COwnerDrawListBox::MeasureItem 中的 32 改为以像素为单位所期望的项目高度，并将 COwnerDrawListBox::DrawItem 中的备注"Draw the item"用绘制索引号为 nIndex 项目的代码替换。使用设备描述表对象 dc 来执行绘制任务并将输出结果限制在由 rect 指定的矩形中，列表框的功能应该完美无缺。（注意保存设备描述表的状态以便在使用前后其值保持不变。）CownerDrawListBox 中 DrawItem 的实现是依靠在项目被选中时（如果 lpdis -> itemState 的 ODS_SELECTED 位被设置）用系统颜色 COLOR_HIGHLIGHT 绘制项目的背景，如果没选中就用 COLOR_WINDOW，并且在项目有输入焦点时（如果 lpdis -> itemState 的 ODS_FOCUS 位被设置）就画一个焦点矩形。您所要做的只有绘制项目本身。PreCreateWindow 覆盖函数确保了设置 LBS_OWNERDRAWFIXED 而不设置 LBS_OWNERDRAWVARIABLE。它同时也清除了 LBS_SORT 位以防调用 CompareItem。

将 COwnerDrawListBox 变为完善的类所需的最后一个功能就是 AddItem 函数，它被调用来给列表框添加非文本项目。例如：在显示图标的列表框中，AddItem 如下：

```
int COwnerDrawListBox::AddItem (HBITMAP hBitmap)
{
    int nIndex = AddString (_T (""));
    if ((nIndex != LB_ERR) && (nIndex != LB_ERRSPACE))
        SetItemData (nIndex, (DWORD) hBitmap);
    return nIndex;
}
```

在这个例子中，AddItem 使用 SetItemData 将位图句柄和列表框索引号关联在了一起。对于给定的项目，列表框的 DrawItem 函数可以用 GetItemData 来检索位图句柄并画出位图。位图是一种资源，不需要时，必须删除。既可以留给列表框的父窗口去删除位图，也可以通过覆盖 CListBox::DeleteItem 让列表框自己删除它们。

图 7-8 所示的 IconView 应用程序使用了名为 CIconListBox 的自制列表框类来显示图标。CIconListBox 覆盖了从 CListBox 继承来的 PreCreateWindow、MeasureItem 和 DrawItem 函数并增加了两个自己的函数。AddIcon 给列表框添加图标，而 ProjectImage 将图标"投影"到显示表面，并对其进行缩放以适合指定矩形的大小。IconView 的源程序代码在图 7-9 中。

IconView 唯一的输入形式是接受拖放。试一下，用鼠标左键抓住 EXE，DLL 或 ICO 文件，将其拖到 IconView 窗口并释放鼠标键。文件中包含的所有图标都会显示在列表框中，第一个图标的放大图形也会出现在 Detail 窗口。想仔细查看文件中其他图标，可以单击该图标或用上下箭头键移动光标来选择。

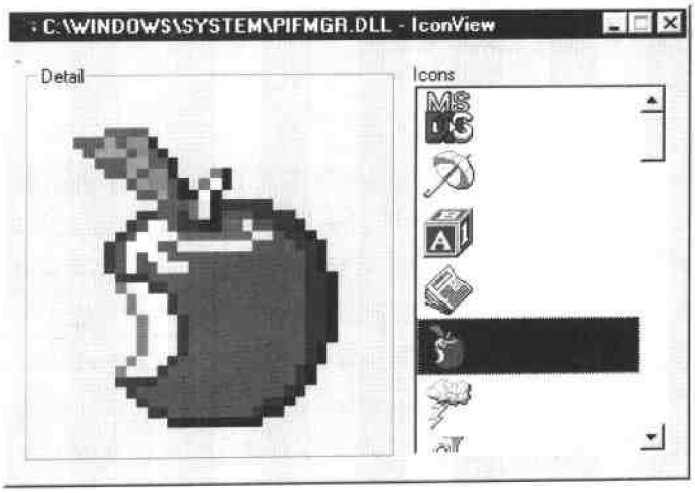

图 7-8 IconView 显示包含在 Pifmgr.dll 中的图标

IconView 用 MFC 中便于使用的 CDC::DrawIcon 函数在列表框中绘制图标。CIconListBox::DrawItem 中关键代码如下:

```
if (nIndex != (UINT) -1)
    dc.DrawIcon (rect.left + 4, rect.top + 2,
        (HICON) GetItemData (nIndex));
```

用 SetItemData 保存图标句柄并用 GetItemData 检索它。如果 nIndex(当前所选列表框项目的索引号)为 -1,则对 DrawIcon 的调用就被跳过。这一点很重要,因为在空的列表框接收到输入焦点时会用列表框索引号 -1 来调用 DrawItem。此时 DrawItem 的工作是给不存在的项目 0 画一个焦点矩形。不能假定 DrawItem 总是用有效的项目索引号来调用。

CmainWindow 的 OnPaint 处理程序所做的工作仅仅是构造一个画图设备描述表并调用列表框的 ProjectImage 函数来绘制当前在窗口的客户区中选中图标的放大图形。ProjectImage 使用 CDC 函数的 BitBlt 和 StretchBlt 来投影图像。现在您对这些代码可能还不理解,但在第 15 章学习了图标之后就会明白了。

IconView 使用的拖放机制是 Windows 3.1 中引入的原始形式。简单地说,在 CMainWindow::OnCreate 中调用 DragAcceptFiles 将 CMainWindow 注册为被拖放目标。一旦注册,无论何时文件从内部命令被拖放到窗口的顶层,窗口都会接收到 WM_DROPFILES 消息。CMainWindow::OnDropFiles 通过使用 ::DragQueryFile API 函数来检索被拖放文件的名字来响应 WM_DROPFILES 消息。然后使用::ExtractIcon 来提取文件中的图标并使用 CIconListBox::AddIcon 将图标加入列表框。

在第 19 章中,会学到丰富的 OLE 拖放形式。在 32 位 Windows 中,仍然支持"旧"的拖放,但它的使用就不如 OLE 拖放那样灵活了。这就是我不去过多讨论它的原因。一旦您看到 OLE 拖放的应用,我相信您会同意把时间花在理解 Windows 3.1 类型的拖放上还不如花

在其他地方呢。

IconView.h

```
class CMyApp : public CWinApp
{
public:
    virtual BOOL InitInstance ();
};

class CIconListBox : public CListBox
{
public:
    virtual BOOL PreCreateWindow (CREATESTRUCT& cs);
    virtual void MeasureItem (LPMEASUREITEMSTRUCT lpmis);
    virtual void DrawItem (LPDRAWITEMSTRUCT lpdis);
    int AddIcon (HICON hIcon);
    void ProjectImage (CDC * pDC, LPRECT pRect, COLORREF clrBackColor);
};

class CMainWindow : public CWnd
{
protected:
    int m_cxChar;
    int m_cyChar;

    CFont m_font;
    CRect m_rcImage;

    CButton m_wndGroupBox;
    CIconListBox m_wndIconListBox;
    CStatic m_wndLabel;

public:
    CMainWindow ();

protected:
    virtual void PostNcDestroy ();

    afx_msg int OnCreate (LPCREATESTRUCT lpcs);
    afx_msg void OnPaint ();
    afx_msg void OnSetFocus (CWnd * pWnd);
    afx_msg void OnDropFiles (HDROP hDropInfo);
    afx_msg void OnSelChange ();

    DECLARE_MESSAGE_MAP ()
};
```

IconView.cpp

```
#include <afxwin.h>
```

```
#include "IconView.h"

#define IDC_LISTBOX 100

CMyApp myApp;

//////////////////////////////////////////////////////////////////////////
// CMyApp member functions

BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow;
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

//////////////////////////////////////////////////////////////////////////
// CMainWindow message map and member functions

BEGIN_MESSAGE_MAP (CMainWindow, CWnd)
    ON_WM_CREATE ()
    ON_WM_PAINT ()
    ON_WM_SETFOCUS ()
    ON_WM_DROPFILES ()
    ON_LBN_SELCHANGE (IDC_LISTBOX, OnSelChange)
END_MESSAGE_MAP ()

CMainWindow::CMainWindow ()
{
    CString strWndClass = AfxRegisterWndClass (
        0,
        myApp.LoadStandardCursor (IDC_ARROW),
        (HBRUSH) (COLOR_3DFACE + 1),
        myApp.LoadStandardIcon (IDI_WINLOGO)
    );
    CreateEx (0, strWndClass, _T ("IconView"),
        WS_OVERLAPPED|WS_SYSMENU|WS_CAPTION|WS_MINIMIZEBOX,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, NULL);

    CRect rect (0, 0, m_cxChar * 84, m_cyChar * 21);
    CalcWindowRect (&rect);

    SetWindowPos (NULL, 0, 0, rect.Width (), rect.Height (),
        SWP_NOZORDER|SWP_NOMOVE|SWP_NOREDRAW);
}

int CMainWindow::OnCreate (LPCREATESTRUCT lpcs)
```

```
{
    if (CWnd::OnCreate (lpcs) == -1)
        return -1;

    m_font.CreatePointFont (80,_T ("MS Sans Serif"));

    CClientDC dc (this);
    CFont * pOldFont = dc.SelectObject (&m_font);
    TEXTMETRIC tm;
    dc.GetTextMetrics (&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight + tm.tmExternalLeading;
    dc.SelectObject (pOldFont);

    m_rcImage.SetRect (m_cxChar * 4, m_cyChar * 3, m_cxChar * 46,
        m_cyChar * 19);

    m_wndGroupBox.Create (_T ("Detail"), WS_CHILD|WS_VISIBLE|BS_GROUPBOX,
        CRect (m_cxChar * 2, m_cyChar, m_cxChar * 48, m_cyChar * 20),
        this, (UINT) -1);

    m_wndLabel.Create (_T ("Icons"), WS_CHILD|WS_VISIBLE|SS_LEFT,
        CRect (m_cxChar * 50, m_cyChar, m_cxChar * 82, m_cyChar * 2),
        this);

    m_wndIconListBox.Create (WS_CHILD|WS_VISIBLE|WS_VSCROLL |
        WS_BORDER|LBS_NOTIFY|LBS_NOINTEGRALHEIGHT,
        CRect (m_cxChar * 50, m_cyChar * 2, m_cxChar * 82, m_cyChar * 20),
        this, IDC_LISTBOX);

    m_wndGroupBox.SetFont (&m_font);
    m_wndLabel.SetFont (&m_font);
    DragAcceptFiles ();
    return 0;
}
void CMainWindow::PostNcDestroy ()
{
    delete this;
}

void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);
    m_wndIconListBox.ProjectImage (&dc, m_rcImage,
        ::GetSysColor (COLOR_3DFACE));
}

void CMainWindow::OnSetFocus (CWnd * pWnd)
```

```
}
    m_wndIconListBox.SetFocus ();
}

void CMainWindow::OnDropFiles (HDROP hDropInfo)
{
    //
    // Find out how many files were dropped.
    //
    int nCount = ::DragQueryFile (hDropInfo, (UINT) -1, NULL, 0);

    if (nCount == 1) { // One file at a time, please
        m_wndIconListBox.ResetContent ();
        //
        // Extract the file's icons and add them to the list box.
        //
        char szFile[MAX_PATH];
        ::DragQueryFile (hDropInfo, 0, szFile, sizeof (szFile));
        int nIcons = (int) ::ExtractIcon (NULL, szFile, (UINT) -1);

        if (nIcons) {
            HICON hIcon;
            for (int i = 0; i < nIcons; i++) {
                hIcon = ::ExtractIcon (AfxGetInstanceHandle (),
                    szFile, i);
                m_wndIconListBox.AddIcon (hIcon);
            }
        }

        //
        // Put the file name in the main window's title bar.
        //
        CString strWndTitle = szFile;
        strWndTitle += _T (" - IconView");
        SetWindowText (strWndTitle);
        //
        // Select item number 0.
        //
        CClientDC dc (this);
        m_wndIconListBox.SetCurSel (0);
        m_wndIconListBox.ProjectImage (&dc, m_rcImage,
            ::GetSysColor (COLOR_3DFACE));
    }
    ::DragFinish (hDropInfo);
}

void CMainWindow::OnSelChange ()
```

```
{
    CClientDC dc(this);
    m_wndIconListBox.ProjectImage(&dc, m_rcImage,
        ::GetSysColor(COLOR_3DFACE));
}

/////////////////////////////////////////////////////////////////////////////
// CIconListBox member functions

BOOL CIconListBox::PreCreateWindow(CREATESTRUCT& cs)
{
    if(! CListBox::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle = WS_EX_CLIENTEDGE;
    cs.style &= ~(LBS_OWNERDRAWVARIABLE|LBS_SORT);
    cs.style|= LBS_OWNERDRAWFIXED;
    return TRUE;
}

void CIconListBox::MeasureItem(LPMEASUREITEMSTRUCT lpmis)
{
    lpmis->itemHeight = 36;
}

void CIconListBox::DrawItem(LPDRAWITEMSTRUCT lpdis)
{
    CDC dc;
    dc.Attach(lpdis->hDC);
    CRect rect = lpdis->rcItem;
    int nIndex = lpdis->itemID;

    CBrush* pBrush = new CBrush;
    pBrush->CreateSolidBrush(::GetSysColor((lpdis->itemState &
        ODS_SELECTED) ? COLOR_HIGHLIGHT : COLOR_WINDOW));
    dc.FillRect(rect, pBrush);
    delete pBrush;

    if(lpdis->itemState & ODS_FOCUS)
        dc.DrawFocusRect(rect);

    if(nIndex != (UINT) -1)
        dc.DrawIcon(rect.left + 4, rect.top + 2,
            (HICON) GetItemData(nIndex));

    dc.Detach();
}

int CIconListBox::AddIcon(HICON hIcon)
{
```

```
    int nIndex = AddString (_T (""));
    if ((nIndex != LB_ERR) && (nIndex != LB_ERRSPACE))
        SetItemData (nIndex, (DWORD) hIcon);
    return nIndex;
}

void CIconListBox::ProjectImage (CDC * pDC, LPRECT pRect,
    COLORREF clrBackColor)
{
    CDC dcMem;
    dcMem.CreateCompatibleDC (pDC);

    CBitmap bitmap;
    bitmap.CreateCompatibleBitmap (pDC, 32, 32);
    CBitmap * pOldBitmap = dcMem.SelectObject (&bitmap);

    CBrush * pBrush = new CBrush (clrBackColor);
    dcMem.FillRect (CRect (0, 0, 32, 32), pBrush);
    delete pBrush;

    int nIndex = GetCurSel ();
    if (nIndex != LB_ERR)
        dcMem.DrawIcon (0, 0, (HICON) GetItemData (nIndex));

    pDC -> StretchBlt (pRect -> left, pRect -> top, pRect -> right - pRect -> left,
        pRect -> bottom - pRect -> top, &dcMem, 0, 0, 32, 32, SRCCOPY);
    dcMem.SelectObject (pOldBitmap);
}
```

图 7-9 The IconView 应用程序

7.2.3 图形按钮

MFC 包含三个自有的派生控件类：CcheckListBox、CdragListBox 和 CbitmapButton。CCheckListBox 将常用的列表框转换为"复选"列表框(列表框中每个项目都带有复选框)并添加了像 GetCheck 和 SetCheck 这样的函数来获取和设置列表框状态。CDragListBox 创建的列表框支持自身的原始拖放形式。CBitmapButton 封装了自制按钮，显示图像而非文本。它提供了自己的 DrawItem 处理程序在响应 WM_DRAWITEM 消息时绘制按钮。您所要做的只是创建按钮并提供代表按钮不同状态的四个位图。

在 16 位 Windows 时代，CBitmapButton 是一个非常有用的工具，使用它可以简化创建图形按钮的任务。然而在今天却很少使用自制按钮了。首先在 Windows 95 中引入的两个按钮类型——BS_BITMAP 和 BS_ICON——使生成按钮变得非常轻松，只要有一个简单的图形就可用它创建一个按钮。BS_BITMAP 样式的按钮(从此以后称为位图按钮)在按钮的表面显

示一个位图。BS_ICON 样式的按钮(图标按钮)则显示图标。大多数程序员喜欢使用图标按钮,这是因为图标与位图不同,它可以具有透明像素。由于透明像素可以将图形的背景色与按钮颜色区分开,所以它们对在按钮上显示非矩形图像很有用。

创建一个图标按钮需要两步操作:

1. 创建一个按钮,它的样式中包含 BS_ICON 标示。
2. 调用按钮的 SetIcon 函数并给它传递一个图标句柄。

下面给出的例子用资源 ID 为 IDI_OK 的图标创建一个图标按钮:

```
m_wndIconButton.Create( _T(""), WS_CHILD|WS_VISIBLE|BS_ICON,
    rect, this, IDC_BUTTON);
m_wndIconButton.SetIcon(AfxGetApp()->LoadIcon(IDI_OK));
```

图标会画在按钮的中央,可以使用表 7-13 中的一个以上的按钮样式修改对齐方式来调整它的位置。

表 7-13　按 钮 样 式

按钮样式	说　明
BS_LEFT	沿按钮表面的左边对齐图标图像
BS_RIGHT	沿按钮表面的右边对齐图标图像
BS_TOP	沿按钮表面的顶边对齐图标图像
BS_BOTTOM	沿按钮表面的底边对齐图标图像
BS_CENTER	在水平方向中间显示图标图像
BS_VCENTER	在垂直方向中间显示图标图像

第 8 章中 Phone 应用程序在对话框内使用图标按钮来代表"确定"和"取消"按钮。

创建位图按钮的过程与创建图标按钮的过程几乎完全相同。只要将 BS_ICON 改为 BS_BITMAP,将 SetIcon 改为 SetBitmap 并进行设置即可。当然,还要用加载位图的代码来替换对 LoadIcon 的调用。在第 15 章中就会学到具体做法了。

在使用图标按钮时要注意的一个问题是按钮失效时会发生什么事情。Windows 由按钮图标生成失效的按钮图形,但结果总不尽人意。一般来说图形越简单越好。在失效时未填满的图像要比填满的图形效果好。

7.2.4　自定义控件的颜色

Windows 控件体系结构中最显著的缺陷就是没有明确的办法来修改控件的颜色。可以用 SetFont 来修改控件的字体,但没有类似的函数可以修改控件的颜色。

MFC 支持两种修改控件的机制。两种机制都要求在控件绘制自身之前要给其父窗口发送一个包含设备描述表句柄的消息,正是用该设备描述表来完成绘制工作的。父窗口可

以使用设备描述表上的 CDC::SetTextColor 和 CDC::SetBkColor 来修改控件所绘文本的属性。通过返回画笔句柄(HBRUSH)还可以修改控件的背景颜色。

在绘制之前给复选框发送的消息根据控件类型的不同而不同。例如:列表框发送 WM_CTLCOLORLISTBOX 消息,静态控件发送 WM_CTLCOLORSTATIC 消息。但在任何情况下,消息的 wParam 总是保存设备描述表句柄,而 lParam 保存控件的窗口句柄。如果窗口处理静态控件的 WM_CTLCOLORSTATIC 消息如下:将设备描述表的文本颜色设置为红色、背景设置成白色、并返回蓝色画笔的画笔句柄,那么控件的文本就会为红色,字符内部和字符间隙为白色,而控件的背景(在控件边框内没有被文本占据的地方)为蓝色。

MFC 的 ON_WM_CTLCOLOR 消息映射宏将所有类型控件的 WM_CTLCOLOR 消息映射给了名为 OnCtlColor 的处理程序。OnCtlColor 的原型如下:

```
afx_msg HBRUSH OnCtlColor (CDC * pDC, CWnd * pWnd, UINT nCtlColor)
```

pDC 是指向设备描述表的指针,pWnd 是一个 CWnd 指针标识控件自身,nCtlColor 标识引发调用的 WM_CTLCOLOR 消息的类型。表 7-14 中是一些 nCtlColor 的可能取值。

表 7-14 nCtlColor 的可能值

nCtlColor	控件类型或窗口类型
CTLCOLOR_BTN	按钮。处理此消息对按钮外观无影响
CTLCOLOR_DLG	对话框
CTLCOLOR_EDIT	编辑控件及组合框的编辑控件部分
CTLCOLOR_LISTBOX	列表框及组合框的列表框部分
CTLCOLOR_MSGBOX	消息框
CTLCOLOR_SCROLLBAR	滚动条
CTLCOLOR_STATIC	静态控件,复选框,单选框,组框,只读或无效编辑控件,以及无效组合框中的编辑控件

5 个 nCtlColor 值属于控件,两个(CTLCOLOR_DLG 和 CTLCOLOR_MSGBOX)用于对话框和消息框。(通过处理 WM_CTLCOLOR 消息可以修改对话框和消息框的颜色。)并不是只有静态控件发送 WM_CTLCOLORSTATIC 消息。您或许会认为单选按钮会发送 WM_CTLCOLORBTN 消息,可事实上在 32 位 Windows 中,它发送 WM_CTLCOLORSTATIC 消息。

那么修改控件颜色的一种方法就是在父窗口类中应用 OnCtlColor。下面给出的 OnCtlColor 操作将名为 m_wndText 的静态控件的颜色在框架窗口中改为了红底白字:

```
HBRUSH CMainWindow::OnCtlColor (CDC * pDC, CWnd * pWnd,
    UINT nCtlColor)
{
    if (m_wndText.m_hWnd == pWnd->m_hWnd)
        pDC->SetTextColor (RGB (255, 255, 255));
```

```
        pDC -> SetBkColor (RGB (255, 0, 0));
        return (HBRUSH) m_brRedBrush;
    }
    CFrameWnd::OnCtlColor (pDC, pWnd, nCtlColor);
}
```

m_brRedBrush 是类型为 CBrush 的 CMainWindow 的一个数据成员，它被实例化如下：

```
m_brRedBrush.CreateSolidBrush (RGB (255, 0, 0));
```

注意在 OnCtlColor 中将想要更改颜色的控件的窗口句柄与产生消息的控件的窗口句柄进行了比较。如果两者不同，消息将被传递给基类。如果不进行这种检查，OnCtlColor 会影响 CmainWindow 中的所有控件而不仅是 m_wndText。

这就是修改控件颜色的一种方法。问题是这种手段就将修改工作交给父窗口去做了如果希望派生一个自己的控件类并在其中包括 SetColor 函数来修改控件颜色该怎么办呢？

派生控件类可以设置自己的颜色，方法是使用 MFC 的 ON_WM_CTLCOLOR_REFLECT 宏将不会被控件父窗口处理的 WM_CTLCOLOR 消息传递回控件自己这里。下面给出类似于 CStatic 控件的程序代码，它把自己绘制成红底白字：

```
class CColorStatic : public CStatic
{
public:
    CColorStatic ();

protected:
    CBrush m_brRedBrush;
    afx_msg HBRUSH CtlColor (CDC* pDC, UINT nCtlColor);
    DECLARE_MESSAGE_MAP ()
};

BEGIN_MESSAGE_MAP (CColorStatic, CStatic)
    ON_WM_CTLCOLOR_REFLECT ()
END_MESSAGE_MAP ()

CColorStatic::CColorStatic ()
{
    m_brRedBrush.CreateSolidBrush (RGB (255, 0, 0));
}

HBRUSH CColorStatic::CtlColor (CDC* pDC, UINT nCtlColor)
{
    pDC -> SetTextColor (RGB (255, 255, 255));
    pDC -> SetBkColor (RGB (255, 0, 0));
    return (HBRUSH) m_brRedBrush;
}
```

CtlColor 与 OnCtlColor 相似,但不接收 OnCtlColor 所接收的 pWnd 参数。因为在函数调用中已经隐含了使用消息的控件,所以它并不需要该参数。

图 7-10 所示的 ColorText 应用程序使用了颜色可以配置的静态控件。用 CColorStatic 实现控件。这个 CColorStatic 比上段中给出的用途更广,不仅可以使用硬编码颜色,它还提供了名为 SetTextColor 和 SetBkColor 的成员函数用来修改自己的颜色。在 ColorText 的"Red"、"Green"或"Blue"单选按钮被单击之后,控件的文本颜色也随着改变。按钮单击激活调用控件 SetTextColor 函数的处理程序(参见图 7-11)。ColorText 并不使用控件的 SetBkColor 函数,包括它是出于对程序完整性的考虑。SetBkColor 控制着文本后面所绘的填充颜色。CcolorStatic 默认颜色为黑色(前台)而系统颜色为 COLOR_3DFACE(背景),但是若要修改它们,一个简单函数就够用了。

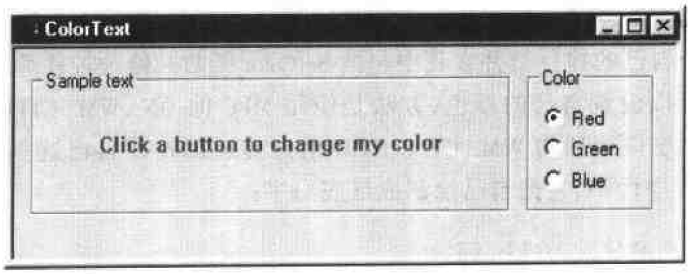

图 7-10 ColorText 程序窗口

ColorText.h

```
#define IDC_RED    100
#define IDC_GREEN   101
#define IDC_BLUE    102

class CColorStatic : public CStatic
{
protected:
    COLORREF m_clrText;
    COLORREF m_clrBack;
    CBrush m_brBkgnd;

public:
    CColorStatic ();
    void SetTextColor (COLORREF clrText);
    void SetBkColor (COLORREF clrBack);

protected:
    afx_msg HBRUSH CtlColor (CDC * pDC, UINT nCtlColor);
    DECLARE_MESSAGE_MAP ()
```

```
};

class CMyApp : public CWinApp
{
public:
    virtual BOOL InitInstance ();
};

class CMainWindow : public CFrameWnd
{
protected:
    int m_cxChar;
    int m_cyChar;
    CFont m_font;

    CColorStatic m_wndText;
    CButton m_wndRadioButtonRed;
    CButton m_wndRadioButtonGreen;
    CButton m_wndRadioButtonBlue;
    CButton m_wndGroupBox1;
    CButton m_wndGroupBox2;

public:
    CMainWindow ();

protected:
    afx_msg int OnCreate (LPCREATESTRUCT lpcs);
    afx_msg void OnRedButtonClicked ();
    afx_msg void OnGreenButtonClicked ();
    afx_msg void OnBlueButtonClicked ();

    DECLARE_MESSAGE_MAP ()
};
```

ColorText.cpp

```
#include <afxwin.h>
#include "ColorText.h"

CMyApp myApp;

//////////////////////////////////////////////////////////////////////////////
// CMyApp member functions

BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow;
    m_pMainWnd->ShowWindow (m_nCmdShow);
```

```
    m_pMainWnd -> UpdateWindow ();
    return TRUE;
}

//////////////////////////////////////////////////////////////////////////
// CMainWindow message map and member functions
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_CREATE ()
    ON_BN_CLICKED (IDC_RED, OnRedButtonClicked)
    ON_BN_CLICKED (IDC_GREEN, OnGreenButtonClicked)
    ON_BN_CLICKED (IDC_BLUE, OnBlueButtonClicked)
END_MESSAGE_MAP ()

CMainWindow::CMainWindow ()
{
    CString strWndClass = AfxRegisterWndClass (
        0,
        myApp.LoadStandardCursor (IDC_ARROW),
        (HBRUSH) (COLOR_3DFACE + 1),
        myApp.LoadStandardIcon (IDI_WINLOGO)
    );

    Create (strWndClass, _T ("ColorText"));
}

int CMainWindow::OnCreate (LPCREATESTRUCT lpcs)
{
    if (CFrameWnd::OnCreate (lpcs) == -1)
        return -1;

    m_font.CreatePointFont (80, _T ("MS Sans Serif"));

    CClientDC dc (this);
    CFont * pOldFont = dc.SelectObject (&m_font);
    TEXTMETRIC tm;
    dc.GetTextMetrics (&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight + tm.tmExternalLeading;
    dc.SelectObject (pOldFont);

    m_wndGroupBox1.Create (_T ("Sample text"), WS_CHILD|WS_VISIBLE |
        BS_GROUPBOX, CRect (m_cxChar * 2, m_cyChar, m_cxChar * 62,
        m_cyChar * 8), this, UINT (-1));

    m_wndText.Create (_T ("Click a button to change my color"),
        WS_CHILD|WS_VISIBLE|SS_CENTER, CRect (m_cxChar * 4,
        m_cyChar * 4, m_cxChar * 60, m_cyChar * 6), this);
```

```
    m_wndGroupBox2.Create (_T ("Color"), WS_CHILD|WS_VISIBLE |
        BS_GROUPBOX, CRect (m_cxChar * 64, m_cyChar, m_cxChar * 80,
        m_cyChar * 8), this, UINT (-1));
    m_wndRadioButtonRed.Create (_T ("Red"), WS_CHILD|WS_VISIBLE |
        WS_GROUP|BS_AUTORADIOBUTTON, CRect (m_cxChar * 66, m_cyChar * 3,
        m_cxChar * 78, m_cyChar * 4), this, IDC_RED);
    m_wndRadioButtonGreen.Create (_T ("Green"), WS_CHILD|WS_VISIBLE |
        BS_AUTORADIOBUTTON, CRect (m_cxChar * 66, (m_cyChar * 9) / 2,
        m_cxChar * 78, (m_cyChar * 11) / 2), this, IDC_GREEN);

    m_wndRadioButtonBlue.Create (_T ("Blue"), WS_CHILD|WS_VISIBLE|
        BS_AUTORADIOBUTTON, CRect (m_cxChar * 66, m_cyChar * 6,
        m_cxChar * 78, m_cyChar * 7), this, IDC_BLUE);

    m_wndRadioButtonRed.SetCheck (1);
    m_wndText.SetTextColor (RGB (255, 0, 0));

    m_wndGroupBox1.SetFont (&m_font, FALSE);
    m_wndGroupBox2.SetFont (&m_font, FALSE);
    m_wndRadioButtonRed.SetFont (&m_font, FALSE);
    m_wndRadioButtonGreen.SetFont (&m_font, FALSE);
    m_wndRadioButtonBlue.SetFont (&m_font, FALSE);
    return 0;
}

void CMainWindow::OnRedButtonClicked ()
{
    m_wndText.SetTextColor (RGB (255, 0, 0));
}

void CMainWindow::OnGreenButtonClicked ()
{
    m_wndText.SetTextColor (RGB (0, 255, 0));
}

void CMainWindow::OnBlueButtonClicked ()
{
    m_wndText.SetTextColor (RGB (0, 0, 255));
}

////////////////////////////////////////////////////////////////////////
// CColorStatic message map and member functions

BEGIN_MESSAGE_MAP (CColorStatic, CStatic)
    ON_WM_CTLCOLOR_REFLECT ()
END_MESSAGE_MAP ()

CColorStatic::CColorStatic ()
```

```
    :
        m_clrText = RGB (0, 0, 0);
        m_clrBack = ::GetSysColor (COLOR_3DFACE);
        m_brBkgnd.CreateSolidBrush (m_clrBack);
    }

    void CColorStatic::SetTextColor (COLORREF clrText)
    {
        m_clrText = clrText;
        Invalidate ();
    }

    void CColorStatic::SetBkColor (COLORREF clrBack)
    {
        m_clrBack = clrBack;
        m_brBkgnd.DeleteObject ();
        m_brBkgnd.CreateSolidBrush (clrBack);
        Invalidate ();
    }

    HBRUSH CColorStatic::CtlColor (CDC * pDC, UINT nCtlColor)
    {
        pDC -> SetTextColor (m_clrText);
        pDC -> SetBkColor (m_clrBack);
        return (HBRUSH) m_brBkgnd;
    }
```

图 7-11 ColorText 应用程序

不同的控件以不同的方式响应 OnCtlColor 和 CtlColor 处理程序所执行的操作。已经看到静态控件是如何响应 CDC::SetTextColor 和 CDC::SetBkColor 了。对于滚动条控件,SetTextColor 和 SetBkColor 什么也不做,除了由 WM_CTLCOLORSCROLLBAR 消息处理程序返回的画笔句柄给滚动条的干体设置颜色以外。对于列表框,SetTextColor 和 SetBkColor 影响非加亮显示的列表框项目,对加亮显示的项目无作用,并且画笔句柄控制着列表框背景(没有绘制文本的空行或非加亮显示行)的颜色。对于按钮,OnCtlColor 和 CtlColor 无论如何也不会起作用,因为 Windows 使用系统颜色来绘制按钮控件。如果 nCtlType 包含代码 CTLCOLOR_BTN,您不妨把它传递给基类,因为对设备描述表所做的任何工作都不会影响到控件的绘制。

7.2.5 消息反射

ON_WM_CTLCOLOR_REFLECT 是 MFC 4.0 中引入的几个消息映射宏中的一个,允许通知消息被反射回发送它们的控件。消息反射对于创建可重用控件类是一个功能强大的工具,因为它使得派生控件类具有了独立于父窗口执行自身操作的能力。以前的 MFC 版本使

用名为 OnChildNotify 的虚拟 CWnd 函数来将某种消息反射回发送它的控件。而在 MFC 的现在版本中使消息反射概念更一般化了，以致派生类都可以将任何发送给父窗口的消息映射给自己的成员函数。在上一节中您已经看到了消息反射的一个例子，我们从 CStatic 派生了一个新类并允许它处理自己的 WM_CTLCOLOR 消息。

表 7-15 中列出了 MFC 提供的消息反射宏以及其功能说明。

表 7-15　MFC 消息反射宏

宏	说　明
ON_CONTROL_REFLECT	反射由 WM_COMMAND 消息接替的通知
ON_NOTIFY_REFLECT	反射由 WM_NOTIFY 消息接替的通知
ON_UPDATE_COMMAND_UI_REFLECT	反射传递给工具栏、状态栏、以及其他用户接口对象的更新通知
ON_WM_CTLCOLOR_REFLECT	反射 WM_CTLCOLOR 消息
ON_WM_DRAWITEM_REFLECT	反射由自制控件发送的 WM_DRAWITEM 消息
ON_WM_MEASUREITEM_REFLECT	反射由自制控件发送的 WM_MEASUREITEM 消息
ON_WM_COMPAREITEM_REFLECT	反射由自制控件发送的 WM_COMPAREITEM 消息
ON_WM_DELETEITEM_REFLECT	反射由自制控件发送的 WM_DELETEITEM 消息
ON_WM_CHARTOITEM_REFLECT	反射由列表框发送的 WM_CHARTOITEM 消息
ON_WM_VKEYTOITEM_REFLECT	反射由列表框发送的 WM_VKEYTOITEM 消息
ON_WM_HSCROLL_REFLECT	反射由滚动条发送的 WM_HSCROLL 消息
ON_WM_VSCROLL_REFLECT	反射由滚动条发送的 WM_VSCROLL 消息
ON_WM_PARENTNOTIFY_REFLECT	反射 WM_PARENTNOTIFY 消息

假设您想编写一个列表框类，它通过显示包含被双击项目的文本的消息框来响应自己的 LBN_DBLCLK 通知。在 SDK 样式的应用程序中，是列表框的父窗口来处理通知并弹出一个消息框。而在 MFC 应用程序中，列表框自己就可以处理通知并显示消息框。以下给出可以实现此功能的派生列表框类：

```
class CMyListBox : public CListBox
{
protected:
    afx_msg void OnDoubleClick();
    DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(CMyListBox, CListBox)
    ON_CONTROL_REFLECT(LBN_DBLCLK, OnDoubleClick)
END_MESSAGE_MAP()

void CMyListBox::OnDoubleClick()
```

```
{
    CString string;
    int nIndex = GetCurSel ();
    GetText (nIndex, string);
    MessageBox (string);
}
```

派生类的消息映射表中的 ON_CONTROL_REFLECT 输入项告诉 MFC 每当列表框给父窗口发送 LBN_DBLCLK 通知时就调用 CMyListBox::OnDoubleClick。值得注意的是只有在父窗口不处理通知时才将通知反射回来,也就是在父窗口的消息映射表中没有包含 ON_LBN_DBLCLK 输入项时。父窗口优先接收,这与 Windows 希望父窗口处理感兴趣的通知是一致的。

第 8 章　对话框和属性表

在实际运用中，大部分控件不是出现在顶层菜单，而是对话框中。“对话框”也称为dialog，是一个窗口，弹出时要求用户输入。例如：在应用程序 File 菜单中选中 Open 时，会出现一个对话窗口；又例如，选中 File-Print，也会出现一个窗口。创建对话框要比创建普通窗口简单，因为 RC 文件中，已有几个语句完整地定义了对话框和其中包含的所有控件。

对话框主要有两类：模式和无模式。在清除对话框之前，“模式”对话框使它所属的窗口——它的所有者——一直处于无效状态。这是应用程序的一种表达，“如果您不提供我需要的输入，我无法做其他任何事。”“无模式”对话框的行为方式更像传统窗口。甚至在对话框显示时，它的所有者也能被激活。

MFC 将模式和无模式对话框的功能都封装在 CDialog 类中。虽然用 Microsoft Windows SDK 编制对话框比较容易，但是用 MFC 就更加容易。为了加速程序开发和减少出现错误的可能性，常常需要建立一些更复杂的对话框，而您只需要几行代码就可以达到目的。MFC 还提供方便的用 C++ 实现的 Windows“公用对话框”——Open 对话框、Print 对话框以及 Windows应用程序中常见的其他对话框。

与对话框相近的是属性表。“属性表”实际上是具有制表页的对话框。属性表的强大之处在于它把对话框中的控件更好地组织在一起。它们对空间的利用率也很高，允许更多的控件进入有限的空间，因此它们在 Windows 应用程序中也愈来愈普遍。借助于 CPropertySheet 和 CPropertyPage 类，MFC 简化了属性表的处理。可以借鉴已有对话框的处理方法：如果您已经编制了几个属性表，但还未形成类库，您会希望 MFC 按照基本方式处理属性表，和处理其他普通对话框一样简单，而不是更加复杂。

8.1　模式对话框和 CDialog 类

创建模式对话框只需要 3 步：

1. 创建一个对话框模板，描述对话框和其中包含的控件。
2. 构造一个 CDialog 对象，并封装该对话框模板。
3. 调用 CDialog::DoModal 显示对话框。

对于简单的对话框，有时您可以直接将 CDialog 实例化。然而，更常见的是，您需要派生一个自己的对话框类，这样您就可以规定它的行为了。首先，我们研究模式对话框的组成部分。然后，我们再把所学的用在无模式对话框和属性表中。

8.1.1 对话框模板

创建对话框的第 1 步是创建一个对话框模板。模板定义了对话框的基本属性,从对话框包含的控件到它的宽和高。尽管用手工编程方式通过把 DLGTEMPLATE 和 DLGITEMTEMPLATE 结构装载到内存中创建对话框是可能的,但是,大部分对话框模板是由应用程序 RC 文件中的语句编译过来的。这些语句可以是手工编写的,但大多数时候它们都是由支持可视化编辑对话框模板的资源编辑器写入 RC 文件的。

下面的 RC 语句定义了一个对话框模板,其资源 ID 为 IDD_MYDIALOG。该模板描述的对话框包含 4 个控件:用于输入文字的单行编辑控件,作为编辑控件标签的静态文本控件,一个 OK 按钮,以及一个 Cancel 按钮:

```
IDD_MYDIALOG DIALOG  0, 0, 160, 68
STYLE DS_MODALFRAME  WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_SYSMENU
CAPTION "Enter Your Name"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT           "&Name", -1, 8, 14, 24, 8
    EDITTEXT        IDC_NAME, 34, 12, 118, 12, ES_AUTOHSCROLL
    DEFPUSHBUTTON   "OK", IDOK, 60, 34, 40, 14, WS_GROUP
    PUSHBUTTON      "Cancel", IDCANCEL, 112, 34, 40, 14, WS_GROUP
END
```

第一行中的数字确定了模板的资源 ID(IDD_MYDIALOG)、对话框的默认位置(0,0 会把对话框放在它的所有者的用户区左上角,但事实上 MFC 会自动将位置为 0,0 的模式对话框挪到中心位置)和对话框尺寸(160,68)。所有尺寸用对话框单位表示。水平方向上,一个对话框单位等于对话框字体中字符平均宽度的四分之一。竖直方向上,一个对话框单位等于字符高的八分之一。因为字符高一般是宽的两倍,所以水平对话框单位代表的距离和竖直对话框单位代表的距离基本相等。如果采用对话框单位量度距离,而不是像素,则在定义对话框相对比例时,您就不必考虑屏幕分辨率了。

对话框模板中的 STYLE 语句确定了对话框的窗口样式。在对话框的窗口样式中总要包含 WS_POPUP。作为特色,还应该加入 WS_VISIBLE,这样,您就不必调用 ShowWindow 在屏幕上显示对话框了。WS_CAPTION 使对话框获得一个标题栏,而 WS_SYSMENU 则在标题栏中添加一个关闭按钮。字首为 DS_的样式是对话框特有的。按照惯例,模板对话框的样式指定为 DS_MODALFRAME。在 Windows 早期版本中,这个样式会给对话框加一个粗边框。现在,DS_MODALFRAME 只会对对话框的行为产生细微的影响,对对话框的外观则毫无作用。其他有趣的对话框样式包括:DS_CENTER,它把对话框放置在屏幕中心位置;DS_ABSALIGN,它把对话框放置在相对于屏幕左上角的地方,而不是相对于其所有者左上角的地方;以及 DS_CONTEXTHELP,它在对话框标题栏中添加一个问号按钮,方便用户获得关于

对话框控件的与上下文有关的帮助。

如果 STYLE 语句包含 DS_SYSMODAL，则您能创建一个“系统模式”对话框。16 位 Windows中，在系统模式对话框被清除之前，系统中的其他窗口全部处于失效状态。系统模式对话框主要用来报告严重错误，这些错误必须在程序进一步执行前得到考虑。在 32 位 Windows中，程序彼此是被操作系统隔离开的，DS_SYSMODAL 使对话框成为最顶层窗口，即该对话框总是出现在其他窗口的上面。虽然它覆盖其他所有窗口，但是用户可以在对话框显示的同时随意切换到其他应用程序。

CAPTION 语句指定了要在对话框标题栏中显示的文字。用 CDialog 对象从 CWnd 继承来的 SetWindowText 函数通过手工编程也能设置标题。

FONT 指定了对话框字体。自动地，该字体被指派给对话框中的所有控件。语句

```
FONT 8, "MS Sans Serif"
```

有些多余，因为 8 磅的 MS Sans Serif 是目前 Windows 版本中的默认字体。如果您的对话框要用在较老的 Windows 版本（Windows 95 之前）中，该语句能保证对话框用 8 磅的 MS Sans Serif 字体。当然还可以用 CWnd::SetFont 改变对话框中单个控件原有的字体。

BEGIN 和 END 间的语句定义了对话框的控件。一个语句定义一个控件，其中要指定控件类型（按钮、复选框、列表框等等）、控件 ID、控件位置、控件宽度和高度以及控件样式。对于静态控件和按钮控件，您还可以指定控件正文。在下面的示例中，LTEXT 语句创建了一个静态文本控件，其 ID 为 -1，文字在控件矩形中左对齐，它的左上角位于对话框左上角 8 个水平对话框单位以右，14 个竖直对话框单位以下，而宽和高分别为 24 个水平对话框单位和 8 个竖直对话框单位。控件文字中的“&”使 Alt-N 成为下一行创建的编辑控件的快捷键。

LTEXT 是几个资源声明中的一个，用来定义对话框模板里的控件（表 8-1 中有完整的控件列表）。在第 7 章通过将控件类实例化，并调用生成对象的 Create 或 CreateEx 函数，我们创建了几类控件，而这些语句在本质上是创建这几类控件的快捷方法。每个关键字都有相关的默认样式，并且所有关键字都设立在样式 WS_CHILD 和 WS_VISIBLE 中。按钮和编辑控件也包含在样式 WS_TABSTOP 中，所以可以用 Tab 键在按钮和编辑控件上跳转。如果需要，可以用 NOT 运算符取消隐含样式。例如：下面的资源声明创建了一个编辑控件，其中取消了默认样式 WS_TABSTOP：

```
EDITTEXT IDC_EDIT, 32, 16, 96, 12, NOT WS_TABSTOP
```

也可以给对话框控件定义更加常见的关键字 CONTROL。有时会这样定义对话框模板：

```
IDD_MYDIALOG DIALOG 0, 0, 160, 68
STYLE DS_MODALFRAME|WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_SYSMENU
CAPTION "Enter Your Name"
BEGIN
    CONTROL     "&Name", -1, "STATIC", SS_LEFT, 8, 14, 24, 8
```

```
    CONTROL     "", IDC_NAME, "EDIT", WS_BORDER|ES_AUTOHSCROLL |
                ES_LEFT|WS_TABSTOP, 34, 12, 118, 12
```

表 8-1　创建对话框控件的资源语句

关键字	控件类型	默认样式
LTEXT	正文左对齐静态控件	SS_LEFT | WS_GROUP
CTEXT	正文居中静态控件	SS_CENTER | WS_GROUP
RTEXT	正文右对齐静态控件	SS_RIGHT | WS_GROUP
PUSHBUTTON	按钮	BS_PUSHBUTTON | WS_TABSTOP
DEFPUSHBUTTON	默认按钮	BS_DEFPUSHBUTTON |WS_TABSTOP
EDITTEXT	编辑控件	ES_LEFT | WS_BORDER |WS_TABSTOP
CHECKBOX	复选框	BS_CHECKBOX | WS_TABSTOP
AUTOCHECKBOX	自动复选框	BS_AUTOCHECKBOX |WS_TABSTOP
STATE3	三态复选框	BS_3STATE | WS_TABSTOP
AUTO3STATE	三态自动复选框	BS_AUTO3STATE | WS_TABSTOP
RADIOBUTTON	单选按钮	BS_RADIOBUTTON | WS_TABSTOP
AUTORADIOBUTTON	自动单选按钮	BS_AUTORADIOBUTTON |WS_TABSTOP
GROUPBOX	组框	BS_GROUPBOX
LISTBOX	列表框	LBS_NOTIFY | WS_BORDER
COMBOBOX	组合框	CBS_SIMPLE
SCROLLBAR	滚动条	SBS_HORZ
ICON	静态图标控件	SS_ICON

```
    CONTROL     "OK", IDOK, "BUTTON", BS_DEFPUSHBUTTON |
                WS_TABSTOP|WS_GROUP, 60, 34, 40, 14
    CONTROL     "Cancel", IDCANCEL, "BUTTON", BS_PUSHBUTTON |
                WS_TABSTOP|WS_GROUP, 112, 34, 40, 14
    END
```

这个对话框模板与 341 页上的模板相同。样式 WS_CHILD 和 WS_VISIBLE 是隐含在 CONTROL 语句中的,但是其他样式必须显式指定。CONTROL 语句的第三个参数指定控件的基础 WNDCLASS: BUTTON 代表按钮、单选按钮、复选框和组框,EDIT 代表编辑控件等等。因为 WNDCLASS 是显式指定的,所以可以用 CONTROL 创建自定义控件,并用::RegisterClass 登录控件的 WNDCLASS。实际上,在往对话框添加进度栏、微调按钮和其他通用控件时,都

用的是 CONTROL 语句。在第 16 章将更详细地介绍通用控件。

如今程序员很少用手工编程的方法创建对话框模板。借助于 Visual C++ 的 Insert/Resource命令，您可以把对话框资源插入项目，并通过可视化界面编辑它。图 8-1 所示为处于工作状态的 Visual C++ 对话框编辑器。往对话框添加控件时，先在 Controls 工具栏中选出所需控件，并照原样把它们画到对话窗口。（如果 Controls 工具栏不显示，可以在 Tools 菜单中选择 Customize，单击 Toolbars 标签，并在 Controls 旁加一个复选标记。这样，Controls 工具栏就能显示出来了。）右击对话框并在上下文菜单中选择 Properties，可以打开对话框属性表。通过在属性表中选择，您可以修改对话框的属性，如 STYLE、CAPTION、FONT 等等。

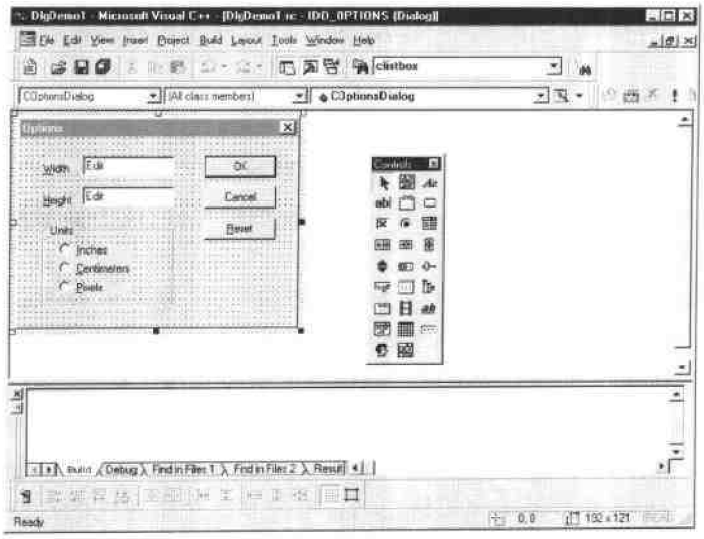

图 8-1　Visual C++ 对话框编辑器

对话框键盘界面

Windows 给每一个对话框提供一个键盘接口，允许用户使用 Tab 键在控件间移动输入焦点，用箭头键使输入焦点在一组控件中循环等等。在 RC 文件中定义对话框模板时，也要显式定义对话框键盘接口。对话框模板中有几个因素会影响键盘接口：

- 控件定义的顺序
- 在控件正文中使用“&”指定快捷键
- 使用 WS_GROUP 样式集成控件
- 使用 DEFPUSHBUTTON 指定默认按钮

对话框模板中控件的创建顺序决定了制表键控制次序，即：用户按 Tab 或 Shift-Tab 键时输入焦点在控件间的移动次序。大部分对话框编辑器，包括 Visual C++ 中的对话框编辑器，允许在可视化界面中指定制表键控制次序。在内部，编辑器调整资源声明的顺序，使它

与指定的制表键控制次序一致。控件必须包含样式 WS_TABSTOP,否则控件无法获得焦点。正是这个原因,前面介绍的许多资源声明中都以默认方式包含 WS_TABSTOP。

由于一些用户更喜欢使用键盘,对话框因此还支持快捷键。通过在控件文本中快捷方式字符前加一个"&",您可以给按钮、单选按钮或复选框控件定制快捷键,如:

```
PUSHBUTTON "&Reset", IDC_RESET, 112, 34, 40, 24, WS_GROUP
```

按下 Alt-R(如果输入焦点落在其他按钮控件上,则只按下 R 即可)就会点中 Reset 按钮。但是如果另一个控件也具有相同的助记符,那么重复按快捷键就会使输入焦点在两个控件间来回跳转。对于本身没有控件文本的列表框、编辑控件和其他控件,可以这样定义快捷键:在创建这些控件的语句前加一个创建静态控件的语句,并在该静态控件的正文中包含一个"&"。例如:语句

```
LTEXT       "&Name", -1, 8, 14, 24, 8
EDITTEXT    IDC_NAME, 34, 12, 118, 12, ES_AUTOHSCROLL
```

创建了一个标记为 Name 的静态控件,还有一个单行编辑控件,位于静态控件的右侧。按下 Alt-N 将输入焦点移至编辑控件。

创建对话框模板时,尤其是对话框包含单选按钮时,还要考虑到键盘接口的另一个方面,这就是如何集成控件。回忆一下第 7 章中的 BS_AUTORADIOBUTTON 样式的单选按钮,如果一组中某个按钮被选中,Windows 要取消选中组中的其他按钮时,则该样式单选按钮必须要集成。Windows 也根据单选按钮组合方式决定在箭头键按下时如何在单选按钮间循环输入焦点。如果要定义一组单选按钮,首先要保证所有按钮在制表键控制次序中的位置是连续的。也就是说,如果组中第 1 个按钮在制表键控制次序中对应的是控制号 5,第 2 个按钮则对应的是 6,第 3 个按钮则对应的是 7 等等如此这般。然后给组中第 1 个单选按钮和制表键控制次序中该组后的第 1 个控件指定样式 WS_GROUP。Windows 编程人员常常也给按钮和复选框指定 WS_GROUP,这样,当输入焦点落在按钮或复选框时,箭头键就不能移动输入焦点了。

在设计对话框键盘接口时还要考虑的一点是:哪个按钮应该作为默认值。在大部分对话框中,有两种方式可以将一个按钮指定(典型的是 OK 按钮)为默认按钮。一个是用 DEFPUSHBUTTON 语句创建按钮,另一个是给按钮指定 BS_DEFPUSHBUTTON 样式。按下 Enter 键,Windows 则模仿单击对话框中默认按钮动作。如果输入焦点落在非按钮控件上,默认按钮则为对话框模板中指定的那一个。然而,当输入焦点在按钮间循环时,"默认按钮"也随之移动。由于默认按钮周围有粗边框,所以您总能辨出哪个按钮是默认按钮。

对话框键盘接口中的所有构成都能在 Visual C++ 对话框编辑器中指定。通过在 Layout 菜单中选中 Tab Order 并按顺序单击控件,您可以指定制表键控制次序。对话框编辑器用标有序号的边框显示制表键控制次序,如图 8-2 所见。如果要把样式 WS_GROUP 赋给控件,则复选选中控件属性表中 Group 组框。用鼠标右键单击控件,并选中 Properties,属性表就打

开了。如果要把一个按钮设置为默认按钮，则复选选中按钮属性表中的Default Button 框。

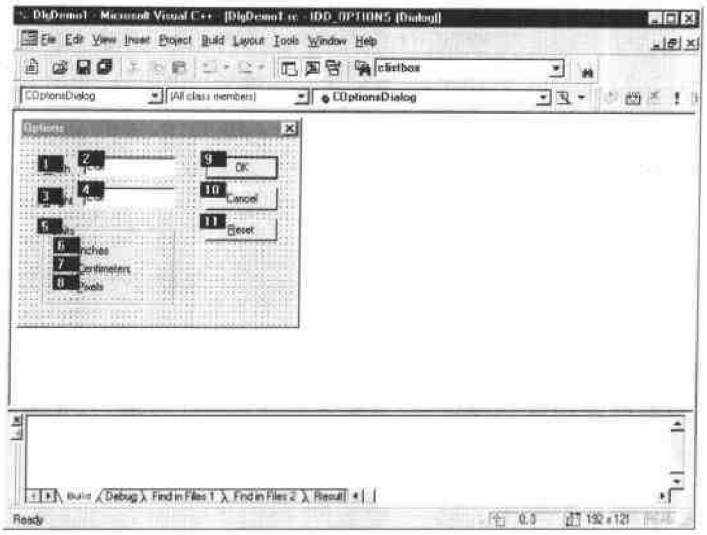

图 8-2　Visual C++ 对话框编辑器中的对话框制表键控制次序

8.1.2　CDialog 类

除了常见的对话框，创建模式对话框的第 2 步是从 CDialog 继承到一个类，并用它来定义对话框的行为。CDialog 包含 3 个关键函数：OnInitDialog、OnOK 和 OnCancel。覆盖这 3 个函数可以将对话框初始化，并响应单击 OK 和 Cancel 按钮。尽管这 3 个函数每个都响应对话框消息，但是您不需要借助消息映射表来处理它们。因为 CDialog 会处理消息映射关系，并把相应的函数作为普通虚拟函数给出。CDialog 还提供 3 个函数的默认实现，如果您使用 MFC Dialog Data Exchange 和 Dialog Data Validation 机制，那么不必覆盖 3 个函数中的任何一个，您就能达到执行 3 个函数的目的。

对话框创建后，它和其他窗口一样接收 WM_CREATE 消息。但是 WM_CREATE 消息传来时，如果对话框模板中指定的控件还没有创建，则控件无法初始化。这时，对话框实际上是空的。而 Windows 用来处理对话框消息的内部窗口程序会响应 WM_CREATE 消息，创建对话框控件。创建控件后，对话框接收到一个 WM_INITDIALOG 消息。此时，对话框就有机会进行必要的初始化了，包括初始化控件。在 CDialog 的派生类中，WM_INITDIALOG 消息激活了对话框的 OnInitDialog 函数。其原型如下：

```
virtual BOOL OnInitDialog ()
```

用 OnInitDialog 可以为实现运用对话框作准备工作，例如：复选选中单选按钮或将正文插入编辑控件。调用 OnInitDialog 时，对话框在内存中，但在屏幕上并不显示。所以用户看不见

您在 OnInitDialog 上的工作,但是会看到它的结果。

OnInitDialog 的返回值告诉 Windows 如何处置输入焦点。如果 OnInitDialog 返回 TRUE,则 Windows 将输入焦点指派给制表键控制次序中的第 1 个控件。如果要把输入焦点指派给其他任意控件,则在 OnInitDialog 中调用控件的 SetFocus 函数,并且 OnInitDialog 返回 FALSE,禁止 Windows 自己设置输入焦点位置。通过将控件 ID 传递给 GetDlgItem,您可以获得调用 SetFocus所需的 CWnd 指针。示范如下:

```
GetDlgItem(IDC_EDIT)->SetFocus();
```

如果要覆盖 OnInitDialog,则须调用基类的 OnInitDialog 处理程序。其中原因我很快就会介绍到。

用户单击对话框 OK 按钮时,对话框接收到 WM_COMMAND 消息,获知按钮被单击了,接着 MFC 调用对话框的虚函数 OnOK。如果要保证该过程正常运行,则须给 OK 按钮指定特殊的 ID 值 IDOK,如下面的资源语句所示:

```
DEFPUSHBUTTON "OK", IDOK, 60, 34, 40, 24, WS_GROUP
```

可以在清除对话框之前覆盖 OnOK,实现某些特殊处理,比如从对话框控件中提取数据并确认数据(例如:确认从编辑控件中获取的数值是否落在允许范围内)。如果您确实使用了 OnOK,记着一定要清除 OnOK。有两种方法可以达到这个目的:或通过调用 EndDialog 清除对话框,或调用基类的 OnOK 处理程序。否则,单击 OK 按钮后对话框并不消失。

必须给 Cancel 按钮指定 OnCancel 需要的预定义 ID,IDCANCEL。单击按钮时,OnCancel 则被调用。注意即使对话框没有 Cancel 按钮,如果按下 Esc 键或单击对话框标题栏中的关闭按钮,OnCancel 都会被调用。如果控件变化被取消,不需要再从对话框控件读取数据,则通常不用覆盖 OnCancel。CDialog::OnCancel 通过参数 IDCANCEL 调用 EndDialog,清除对话框并通知调用者应忽略对话框控件中的变化。

除了 WM_INITDIALOG 消息是对话框特有的,对话框和传统窗口接收的消息是完全相同的,可以在消息映射表中为每个消息找到对应的对话框类成员函数。例如:如果对话框包含一个 Reset 按钮,其 ID 为 IDC_RESET,并且您希望单击按钮时调用 OnReset,则可用下面的消息映射表项把两者联系起来:

```
ON_BN_CLICKED(IDC_RESET, OnReset)
```

对话框甚至能处理 WM_PAINT 消息;虽然这种处理不常见,但是是可以实现的。由于控件所在的对话框区域无效时,控件会重画自身,所以大部分对话框不需要 OnPaint 处理程序。

向 ClassWizard 寻求帮助

尽管用手工编程方式从 CDialog 派生对话框类非常有效,但是如今大部分 MFC 编程人

员喜欢由 ClassWizard 替他们派生对话框。这很简单：打开 ClassWizard，单击 Add Class 按钮，在按钮下显示的菜单中选择 New，并填入类名、基类名（CDialog）和资源 ID，如图 8-3 所示。指定的资源 ID 必须是对话框的资源 ID。如果您想在派生类中覆盖 OnInitDialog、OnOK、或 OnCancel，则要在 ClassWizard 实现派生类后才能进行。

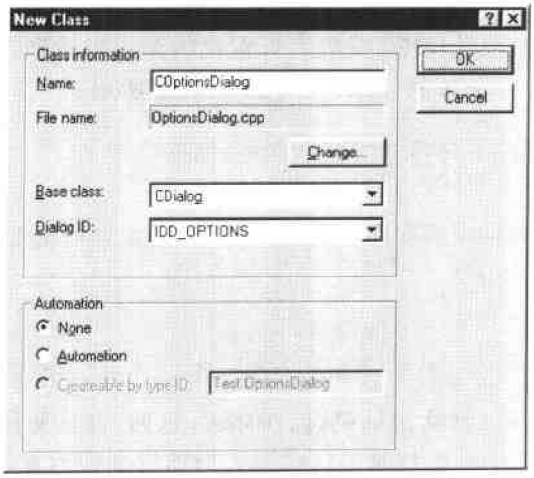

图 8-3　使用 ClassWizard 从 CDialog 派生新类

这是得到 ClassWizard New Class 对话框的一种方法，但不是唯一的一种。在对话框编辑器中双击对话框体，Visual C++ 会提示您：是否想创建一个新类。如果单击 OK，则弹出 ClassWizard，并显示 New Class 对话框，其中已填入基类名和资源 ID。

还可以用 ClassWizard 为对话框控件编写消息处理程序。假定您想给一个按钮编写 BN_CLICKED 处理程序，其中按钮的控件 ID 为 IDC_RESET，那么要这样进行：

1. 在 ClassView 中右击对话框类。
2. 在上下文菜单中选中 Add Windows Message Handler。
3. 在 Class Or Object To Handle 框中选择按钮 ID（IDC_RESET）。
4. 在 New Windows Messages/Events 框中单击 BN_CLICKED。
5. 单击 Add Handler 按钮，并输入函数名。

这些操作完成之后，与该函数名相应的函数就出现在对话框类中，并通过对话框消息映射表中的 ON_BN_CLICKED 项与按钮关联起来。

8.1.3　创建模式对话框

一旦定义了对话框模板并声明了对话框类，创建模式对话框就变得很简单：只要在 CDialog 派生类中构造一个对象，并调用该对象的 DoModal 函数即可。DoModal 直到对话框被清除后才返回值，其返回值是传递给 EndDialog 的值。应用程序特别地要检测 DoModal 的返

回值,如果返回值为 IDOK,告知对话框已用 OK 按钮取消,应用程序才会有响应。如果返回值不是 IDOK(很有可能是 IDCANCEL),则忽略输入对话框的信息。

CDialog 定义了两个构造函数:一个函数接收字符串型对话框模板资源 ID 和标识对话框所有者的 CWnd 指针,另一个函数接收整型对话框模板资源 ID 和标识对话框所有者的 CWnd 指针。如果应用程序主窗口是对话框的所有者,则 CWnd 指针可以省略。为了使派生的对话框类更加完备和对象化,MFC 编程人员常常为派生对话框类建立自己的构造函数。可以为 CMyDialog 编写一个简单的 inline 构造函数,如下所示:

```
CMyDialog::CMyDialog (CWnd * pParentWnd = NULL) :
    CDialog (IDD_MYDIALOG, pParentWnd) {}
```

该构造函数简化了创建对话框的程序代码,并消除了无意中把错误资源标识传递给该构造函数的可能性。

```
CMyDialog dlg;
dlg.DoModal ();
```

用户通过单击 OK 或 Cancel 清除对话框时,DoModal 返回,并且调用 DoModal 的函数继续执行。如果调用 DoModal 后的动作与是否认可输入对话框的数据有关(几乎是必然的),则要检测返回值,如下所示:

```
CMyDialog dlg;
if (dlg.DoModal () == IDOK) {
    // The user clicked OK: do something!
}
```

默认方式下,DoModal 只会返回 IDOK 和 IDCANCEL。然而,通过调用参数不再是IDOK或IDCANCEL 的 EndDialog,您可以编写自己的对话框类返回其他值。例如:在对话框中构建一个 End This Application 按钮,并把它连接到程序中:

```
// In the dialog class
BEGIN_MESSAGE_MAP (CMyDialog, CDialog)
    ON_BN_CLICKED (IDC_ENDAPP, OnEndThisApplication)
END_MESSAGE_MAP ()
  .
  .
  .
void CMyDialog::OnEndThisApplication ()
{
    EndDialog (IDC_ENDAPP);
}

// Elsewhere in the application
```

```
CMyDialog dlg;
int nReturn = dlg.DoModal ();
if (nReturn == IDOK) {
    // The user clicked OK; do something!
}
else if (nReturn == IDC_ENDAPP)
    PostMessage (WM_CLOSE, 0, 0);
```

用户单击对话框中 End This Application 时，返回值 IDC_ENDAPP 提示调用函数用户要结束应用程序。因此，WM_CLOSE 消息进入消息队列，准备启动关闭程序。与传递给 EndDialog 的 IDC_ENDAPP 和其他用户定义的值相对应的 ID 值要大于等于 3，这样可以避免与预定义的 IDOK 和 IDCANCEL 的按钮 ID 值相冲突。

8.1.4　对话框数据交换和对话框数据校验

典型的对话框给用户提供一列选项，同时把相关的输入汇集起来，并使输入的数据符合应用程序的要求。一种获取用户输入的方便方法是把输入赋给对话框类的公用成员变量。这样，使用了对话框的应用程序可以通过读或写对话框对象的成员变量来访问数据。

假定您编写的对话框包含两个单行编辑控件，一个用来输入名字，另一个用来输入电话号码。为了使应用程序能获取用户输入的名字和电话号码，在对话框类中声明两个 CString 成员变量：

```
class CMyDialog : public CDialog
{
public:
    CMyDialog::CMyDialog (CWnd* pParentWnd = NULL) :
        CDialog (IDD_MYDIALOG, pParentWnd) {}
    CString m_strName;
    CString m_strPhone;
      .
      .
      .
};
```

如果要得到用户输入的名字和电话号码，则显示对话框并在对话框清除后取回 m_strName 和 m_strPhone 的值：

```
CMyDialog dlg;
if (dlg.DoModal () == IDOK) {
    CString strName = dlg.m_strName;
    CString strPhone = dlg.m_strPhone;
```

```
    TRACE (_T ("Name = %s, Phone = %s"), strName, strPhone);
}
```

如果要给编辑控件设置默认名字和电话号码,则需稍稍改动程序代码:

```
CMyDialog dlg;
dlg.m_strName = _T ("Jeff");
dlg.m_strPhone = _T ("555-1212");
if (dlg.DoModal () == IDOK) {
    CString strName = dlg.m_strName;
    CString strPhone = dlg.m_strPhone;
    TRACE (_T ("Name = %s, Phone = %s"), strName, strPhone);
}
```

这几个示例假定 m_strName 和 m_strPhone 和对话框编辑控件有固有的联系,也就是说,赋给两变量的字符串不知如何就显示在编辑控件中,并且从变量中读出的字符串正是用户输入编辑控件的字符串。

对话框控件和数据成员的结合并不是自己实现的,需要编程人员来实现。一种实现方法是在派生的对话框类中覆盖 OnInitDialog 和 OnOK,并增添实现控件和数据成员间数据传送的程序代码。假定编辑控件的 ID 是 IDC_NAME 和 IDC_PHONE,下面是修改后的 CMyDialog,其中运用了这种方法:

```
class CMyDialog : public CDialog
{
public:
    CMyDialog::CMyDialog (CWnd* pParentWnd = NULL) :
        CDialog (IDD_MYDIALOG, pParentWnd) {}
    CString m_strName;
    CString m_strPhone;
protected:
    virtual BOOL OnInitDialog ();
    virtual void OnOK ();
};

BOOL CMyDialog::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    SetDlgItemText (IDC_NAME, m_strName);
    SetDlgItemText (IDC_PHONE, m_strPhone);
    return TRUE;
}

void CMyDialog::OnOK ()
```

```
    :
    GetDlgItemText (IDC_NAME, m_strName);
    GetDlgItemText (IDC_PHONE, m_strPhone);
    CDialog::OnOK();
}
```

按照这种方式构建 CMyDialog 可以确保对话框创建之前赋给 m_strName 和 m_strPhone 的字符串能出现在编辑控件中，并且在用 OK 按钮清除对话框时能把输入编辑控件的字符串复制到 m_strName 和 m_strPhone 中。

可以想像：如果不必在 OnInitDialog 中初始化控件并在 OnOK 中又将它们读取出来，也就是说，如果您能建立一个成员变量和相关控件的“数据映射表”，实现 CMyDialog 非常容易。听上去似乎很勉强吧。但事实确实如此。实际上，这正是 MFC 对话框数据交换（Dialog Data Exchange，DDX）机制的功用。DDX 使用起来很简单，在大多数情况下，即便对话框包含很多控件，您也不必提供自定义 OnInitDialog 和 OnOK 函数。

通过在每个 CDialog 的派生类中覆盖虚函数 DoDataExchange 可以创建 DDX。在覆盖过程中，可以使用 MFC 提供的 DDX 函数实现对话框控件和数据成员间的数据传送。下面 DoDataExchange的实现将两个 CString 数据成员（m_strName 和 m_strPhone）与一对编辑控件联系起来（IDC_NAME and IDC_PHONE）：

```
void CMyDialog::DoDataExchange(CDataExchange* pDX)
{
    DDX_Text(pDX, IDC_NAME, m_strName);
    DDX_Text(pDX, IDC_PHONE, m_strPhone);
}
```

对话框创建时（对话框接收到 WM_INITDIALOG 消息时），MFC 调用一次 DoDataExchange，OK 按钮被单击时，再调用一次 DoDataExchange。pDX 参数是指向 MFC 提供的 DoDataExchange 对象的指针。其中，CDataExchange 对象通报 DDX_Text 信息的流向，也就是说，数据是从数据成员传送到控件呢，还是从控件传送到数据成员。一旦确定了数据流向，DDX_Text 实现实际的数据传送。因而，一个 DoDataExchange 函数足可以实现：对话框创建时把数据成员中的数据复制到控件，并在清除对话框时将控件中的数据复制到数据成员。

DDX_Text 是 MFC 提供的几个 DDX 函数之一，请见表 8-2 中列出的部分 DDX 函数。控件和数据成员间的关系取决于联系两者的 DDX 函数。例如：有整型变量通过 DDX_Radio函数与一组单选按钮联系在一起，其中保存着基于 0 的索引值，用来标识组中成员。如果对话框创建时该整型值为 2，则 DDX_Radio 单选选中组中第 3 个按钮。当单击 OK 按钮时，DDX_Radio 把当前选中按钮的索引值复制到成员变量中。

表 8-2　对话框数据交换(DDX)函数

DDX 函数	说　明
DDX_Text	将 BYTE、int、short、UINT、long、DWORD、Cstring、string、float、double、COleDateTime、或 COleCurrency 变量与编辑控件联系起来
DDX_Check	将整型变量与复选框控件联系起来
DDX_Radio	将整型变量和一组单选按钮联系起来
DDX_LBIndex	将整型变量和列表框联系起来
DDX_LBString	将 CString 变量和列表框联系起来
DDX_LBStringExact	将 CString 变量和列表框联系起来
DDX_CBIndex	将整型变量和组合框联系起来
DDX_CBString	将 CString 变量和组合框联系起来
DDX_CBStringExact	将 CString 变量和组合框联系起来
DDX_Scroll	将整型变量和滚动条联系起来

通过 DDX_Scroll 与滚动条相联系的整型变量确定滚动条滑块的位置。通过 DDX_Check 与复选框相联系的整型变量确定复选框的状态——BST_CHECKED 或 BST_UNCHECKED;对于三态复选框来说,则为 BST_INDETERMINATE。如果一个整型变量通过 DDX_Text 函数与编辑控件联系起来,MFC 会在给编辑控件传送值时将整型值转换为正文字符串,而在给变量传送值时将字符串转换为整型值。

还有一个相关的机制,对话框数据校验(DDV),它允许 MFC 在清除对话框之前校验输入对话框控件的值。DDV 函数分为两类:一些函数用来校验数值,确信数值落在指定的范围内,还有一个函数用来校验 CString 变量,确信其长度没有超过特定值。下面是 DoDataExchange函数,它运用 DDX_Text 将一个整型成员变量和一个编辑控件联系起来,并在对话框 OK 按钮被单击时对值进行校验,确信值在要求的范围内:

```
void CMyDialog::DoDataExchange(CDataExchange* pDX)
{
    DDX_Text(pDX, IDC_COUNT, m_nCount);
    DDV_MinMaxInt(pDX, m_nCount, 0, 100);
}
```

如果 OK 按钮被单击时显示在编辑控件中的值小于 0 或大于 100,DDV_MinMaxInt 将输入焦点传送到控件,并显示一个错误提示消息。对于给定的数据成员,DDV 函数调用应该紧接在 DDX 函数调用之后,这样,如果经校验得知值不符合要求,MFC 就可以将输入焦点放置在适当的控件中。

DDV_MinMaxInt 是 MFC 提供的几个 DDV 函数之一。表 8-3 中所列的是一些属于经典控件的 DDV 函数。由于没有重载 DDV 范围校验例程,不能接受多种数据类型,所以如果想手工编制一个 DoDataExchange 函数,则一定要使函数和数据类型匹配。

表 8-3　对话框数据校验(DDV)函数

函数	说　明
DDV_MinMaxByte	验证 BYTE 值是否落在指定范围内
DDV_MinMaxInt	验证整型值是否落在指定范围内
DDV_MinMaxLong	验证 long 值是否落在指定范围内
DDV_MinMaxUInt	验证 UINT 值是否落在指定范围内
DDV_MinMaxDWord	验证 DWORD 值是否落在指定范围内
DDV_MinMaxFloat	验证 float 值是否落在指定范围内
DDV_MinMaxDouble	验证 double 值是否落在指定范围内
DDV_MaxChars	输入时,用 EM_LIMITTEXT 消息限定可输入编辑控件的字符数;退出时,验证控件包含的字符数是否超过指定的字符数

驱动 DDX 和 DDV 的程序代码在 CDialog 中。创建对话框时,CDialog::OnInitDialog 调用参数为 FALSE 的 UpdateData 函数。该函数是从 CWnd 继承来的对话框对象。UpdateData创建一个 CDataExchange 对象,并调用该对话框的 DoDataExchange 函数,将指向 CDataExchange 对象的指针传递给该函数。这时,被 DoDataExchange 调用的每个 DDX 函数都用成员变量的值初始化控件。而后,用户单击 OK 时,CDialog::OnOK 通过参数 TRUE 调用 UpdateData,DDX 函数则将控件中的值复制到成员变量中。而 DoDataExchange 中每个 DDV 函数会在这时检验用户输入,确定其合法性。以前我曾提到:如果在派生类中覆盖了 OnOK 和 OnInitDialog 函数,则一定要调用基类中的这两个函数。现在您就明白为什么了吧。如果没有调用基类的这两个函数,上结构就无法调用 UpdateData,而 DDX 和 DDV 也不能起作用了。

总的来说,运用 DDX 和 DDV 能够方便地从控件中获取数据或将数据传送给控件,并对数据进行简单的校验。实际上,如果只为了实现数据在对话框控件和数据成员间的传送,运用 DDX 和 DDV 则可以避免覆盖 OnInitDialog 和 OnOK。

向 ClassWizard 寻求更多的帮助

在用 MFC 向导构建的应用程序中,您可以通过手工编程方式将 DDX 和 DDV 函数添加到 DoDataExchange 中,或者也可以让 ClassWizard 替您做这件事。ClassWizard 甚至能替您在对话框类中添加成员变量。给对话框类添加一个数据成员并通过 DDX 或 DDV 将该数据成员和控件联系起来需要以下几个步骤:

1. 打开 ClassWizard,来到 Member Variables 页(参见图 8-4)。

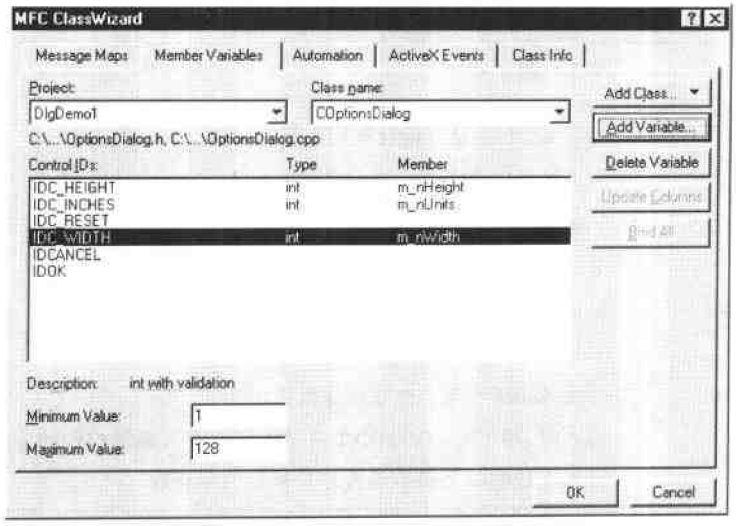

图 8-4 ClassWizard 的 Member Variables 页

2. 在 Class Name 框中选中对话框类的名称。
3. 在 Control ID 框中选中要和成员变量关联的控件的 ID,并单击 Add Variable 按钮。
4. 把成员变量名称敲入 Add Member Variable 对话框,如图 8-5 所示,并在 Variable Type 框中选择变量类型,然后单击 OK。

取消 ClassWizard 后您如果检查对话框类的资源代码,会发现 ClassWizard 已在类声明中添加了成员变量,并在 DoDataExchange 中添加了一个 DDX 语句,将该成员变量和控件联系起来。如果变量是数值型,可以把最大和最小值输入 Member Variables 页底部的编辑控件,而 ClassWizard 就会再添加一个 DDV_MinMax 语句。对于字符串型变量,可以输入最大字符数,ClassWizard 会添加一个 DDV_MaxChars 语句。

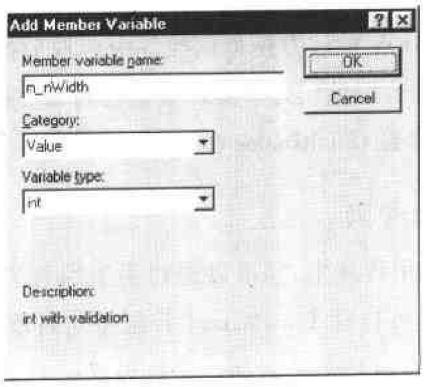

图 8-5 ClassWizard 的 Add Member Variable 对话框

8.1.5　与对话框中的控件相互作用

是不是有了 DDX 和 DDV 就意味着不必再编写代码与对话框中控件相互作用了？答案是否定的。例如：如果想在 OnInitDialog 中向列表框添加字符串，您可能需要对列表框控件调用 CListBox 函数。为此，您需要一个指向列表框的 CListBox 指针。问题是：怎样才能获得这个指针呢？

通过调用 CWnd::GetDlgItem 可以得到一个能指向对话框中任意控件的指针。下面的程序示例使用了 GetDlgItem 和 CWnd::EnableWindow，共同使 ID 为 IDC_CHECK 的控件有效：

```
CWnd * pWnd = GetDlgItem (IDC_CHECK);
pWnd -> EnableWindow (TRUE);
```

由于 GetDlgItem 返回一个普通的 CWnd 指针，而 EnableWindow 又是 CWnd 函数，所以该程序运行效果很好。但是看看下面这个程序示例：

```
CListBox * pListBox = (CListBox * ) GetDlgItem (IDC_LIST);
pListBox -> AddString (_T ("One"));
pListBox -> AddString (_T ("Two"));
pListBox -> AddString (_T ("Three"));
```

这个程序之所以有效只在于 MFC 提供了特别的支持。因为由 GetDlgItem 返回 CWnd 指针，接着把该指针变成 CListBox 指针，并通过改变后的指针调用 CListBox 函数，这种编程方法在某些场合中虽拙劣但尚可用，在某些场合中则很危险。实际上，在某些情况下，这种方法根本不起作用。

如果要对对话框中控件调用非 CWnd 函数，较好的解决办法是用 MFC 的 CWnd::Attach 函数。借助于 Attach 函数，您可以声明一个控件类实例（例如：ClistBox）并动态地把它挂接到对话框控件上。关于如何运用 Attach 把字符串添加到列表框，请参见下面的示例：

```
CListBox wndListBox;
wndListBox.Attach (GetDlgItem (IDC_LIST) -> m_hWnd);
wndListBox.AddString (_T ("One"));
wndListBox.AddString (_T ("Two"));
wndListBox.AddString (_T ("Three"));
wndListBox.Detach ();
```

正如该示例所示的那样，如果 CListBox 对象是在栈上声明的，则在 CListBox 对象失效之前一定要调用 Detach。否则，CListBox 的析构函数就会清除列表框，而该列表框也突然在对话框中消失。

MFC 的 DDX_Control 函数提供了一个完美的机制，可以把 MFC 控件类的实例挂接到对话框的控件上。如果把下面的语句放在派生对话框类的 DoDataExchange 函数中，显然就将

CListBox 数据成员 m_wndListBox 和 ID 为 IDC_LIST 的列表框控件联系起来了:

```
DDX_Control (pDX, IDC_LIST, m_wndListBox);
```

现在只要对 m_wndListBox 调用 AddString 就可以把字符串添加到列表框中:

```
m_wndListBox.AddString (_T ("One"));
m_wndListBox.AddString (_T ("Two"));
m_wndListBox.AddString (_T ("Three"));
```

DDX_Control 提供附加值,因为它不只像 Attach 那样将控件的窗口句柄封装起来,DDX_Control 还动态地将控件纳为子类,这样,传送给控件的消息首先通过 DDX_Control 第三个参数中指定的对象。这是使对话框控件与派生控件类对象行为接近的最简单且最有效的方法。例如:要使编辑控件行为与 CNumEdit 相似,而不是普通的 CEdit。在本章末的 Phones 应用程序中您会看到有关示例展示如何在对话框中运用派生控件类。

还可以运用 ClassWizard 把 DDX_Control 语句添加到 DoDataExchange 中。为此,首先来到 Member Variables 页,然后用 Add Variable 按钮将成员变量添加到对话框类中。但是这次在 Add Member Variable 对话框的 Category 框中选择 Control 而不是 Value。然后在 Variable Type 框中选择一个控件类,并在退出 ClassWizard 前单击 OK。现在,如果检查对话框类,您会发现 ClassWizard 已添加了成员变量和把变量与控件联系起来的 DDX_Control 语句。

8.1.6 DlgDemo1 应用程序

图 8-6 所示的 DlgDemo1 应用程序是一个基于视图的简单程序。它在视图左上角画了一个带颜色的矩形。File 菜单有一个 Options 命令。它显示一个对话框,通过该对话框您可以改变矩形的宽、高和尺寸单位。

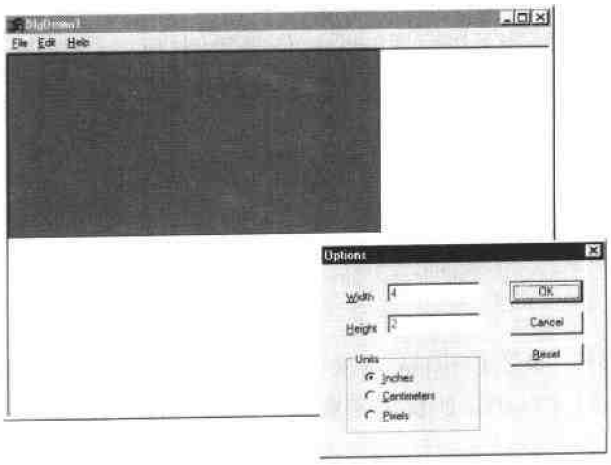

图 8-6 DlgDemo1 窗口和对话框

在这里我用 AppWizard 创建了 DlgDemo1，并借助于 ClassWizard 派生出对话框类，编写消息处理程序等等。部分资源代码如图 8-7 所示。COptionsDialog 封装了对话框资源 IDD_OPTIONS，而对话框是 COptionsDialog 的一个实例。COptionsDialog 有 3 个公用数据成员：m_nWidth、m_nHeight 和 m_nUnits，它们分别代表矩形的宽、高和尺寸单位。每个数据成员通过 DDX 与对话框中一个控件（或一组控件）联系起来。m_nUnits 值等于 0 时代表英寸，等于 1 时代表厘米，等于 2 时代表像素。视图类 CChildView 包含名称相同的成员变量，CChildView::OnPaint 则用这些变量画出矩形。

在 File 菜单中选中 Options 命令时，命令处理程序 CChildView::OnFileOptions 将 COptionsDialog 实例化，并把当前宽、高和尺寸单位复制到对话框对象的成员变量中，最后用 DoModal 显示对话框。如果 DoModal 返回 IDOK，OnFileOptions 则从对话框数据成员中读取宽、高和尺寸单位，并将它们复制到视图的数据成员中。然后调用 Invalidate 重画视图，更新矩形，使矩形与新参数保持一致。

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__AC8095E8_902A_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_MAINFRM_H__AC8095E8_902A_11D2_8E53_006008A82731__INCLUDED_

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

#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
    CMainFrame();
protected:
    DECLARE_DYNAMIC(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
```

```
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
        AFX_CMDHANDLERINFO* pHandlerInfo);
    //}}AFX_VIRTUAL
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
    CChildView    m_wndView;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg void OnSetFocus(CWnd *pOldWnd);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif
// !defined(AFX_MAINFRM_H__AC8095E8_902A_11D2_8E53_006008A82731__INCLUDED_)
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "DlgDemo1.h"
#include "MainFrm.h"

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

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

```
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)

    ON_WM_SETFOCUS()
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction
CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    cs.lpszClass = AfxRegisterWndClass(0);
    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnSetFocus(CWnd* pOldWnd)
```

```
{
    // forward focus to the view window
    m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
    AFX_CMDHANDLERINFO* pHandlerInfo)
{
    // let the view have first crack at the command
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;
    // otherwise, do default handling
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
        CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL))
        return -1;

    return 0;
}
```

ChildView.h

```
// ChildView.h : interface of the CChildView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDVIEW_H__AC8095EA_902A_11D2_8E53_006008A82731__
INCLUDED_)

#define AFX_CHILDVIEW_H__AC8095EA_902A_11D2_8E53_006008A82731__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CChildView window

class CChildView : public CWnd
{
// Construction
public:
    CChildView();
```

```
// Attributes
public:

// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildView)
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildView();

    // Generated message map functions
protected:
    int m_nUnits;
    int m_nHeight;
    int m_nWidth;
    //{{AFX_MSG(CChildView)
    afx_msg void OnPaint();
    afx_msg void OnFileOptions();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
//defined(AFX_CHILDVIEW_H__AC8095EA_902A_11D2_8E53_006008A82731__INCLUDED_)
```

ChildView.cpp

```
// ChildView.cpp : implementation of the CChildView class
//

#include "stdafx.h"
#include "DlgDemo1.h"
#include "OptionsDialog.h"
#include "ChildView.h"

#ifdef _DEBUG
```

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

/////////////////////////////////////////////////////////////////////////////
// CChildView

CChildView::CChildView()
{
    m_nWidth = 4;
    m_nHeight = 2;
    m_nUnits = 0;
}

CChildView::~CChildView()
{
}

BEGIN_MESSAGE_MAP(CChildView,CWnd )
    //{{AFX_MSG_MAP(CChildView)
    ON_WM_PAINT()
    ON_COMMAND(ID_FILE_OPTIONS, OnFileOptions)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CChildView message handlers

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle|= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
        ::LoadCursor(NULL, IDC_ARROW), HBRUSH(COLOR_WINDOW+1), NULL);

    return TRUE;
}

void CChildView::OnPaint()
{
    CPaintDC dc(this); // Device context for painting.

    CBrush brush (RGB (255, 0, 255));
    CBrush* pOldBrush = dc.SelectObject (&brush);
    switch (m_nUnits) {

    case 0: // Inches.
```

```
            dc.SetMapMode (MM_LOENGLISH);
            dc.Rectangle (0, 0, m_nWidth * 100, -m_nHeight * 100);
            break;

        case 1: // Centimeters.
            dc.SetMapMode (MM_LOMETRIC);
            dc.Rectangle (0, 0, m_nWidth * 100, -m_nHeight * 100);
            break;

        case 2: // Pixels.
            dc.SetMapMode (MM_TEXT);
            dc.Rectangle (0, 0, m_nWidth, m_nHeight);
            break;
        }
        dc.SelectObject (pOldBrush);
    }

    void CChildView::OnFileOptions()
    {
        COptionsDialog dlg;

        dlg.m_nWidth = m_nWidth;
        dlg.m_nHeight = m_nHeight;
        dlg.m_nUnits = m_nUnits;

        if (dlg.DoModal () == IDOK) {
            m_nWidth = dlg.m_nWidth;
            m_nHeight = dlg.m_nHeight;
            m_nUnits = dlg.m_nUnits;
            Invalidate ();
        }
    }
```

OptionsDialog.h

```
#if !defined(AFX_OPTIONSDIALOG_H__AC8095F0_902A_11D2_8E53_006008A82731__
INCLUDED_)
#define AFX_OPTIONSDIALOG_H__AC8095F0_902A_11D2_8E53_006008A82731__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// OptionsDialog.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// COptionsDialog dialog
```

```
class COptionsDialog : public CDialog
{
// Construction
public:
    COptionsDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(COptionsDialog)
    enum { IDD = IDD_OPTIONS };
    int   m_nWidth;
    int   m_nHeight;
    int   m_nUnits;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(COptionsDialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(COptionsDialog)
    afx_msg void OnReset();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
//!defined(
//  AFX_OPTIONSDIALOG_H__AC8095F0_902A_11D2_8E53_0006008A82731__INCLUDED_)
```

OptionsDialog.cpp

```
// OptionsDialog.cpp : implementation file
//
#include "stdafx.h"
#include "DlgDemo1.h"
```

```
#include "OptionsDialog.h"

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

/////////////////////////////////////////////////////////////////////////////
// COptionsDialog dialog

COptionsDialog::COptionsDialog(CWnd* pParent /* = NULL */)
    : CDialog(COptionsDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(COptionsDialog)
    m_nWidth = 0;
    m_nHeight = 0;
    m_nUnits = -1;
    //}}AFX_DATA_INIT
}

void COptionsDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(COptionsDialog)
    DDX_Text(pDX, IDC_WIDTH, m_nWidth);
    DDV_MinMaxInt(pDX, m_nWidth, 1, 128);
    DDX_Text(pDX, IDC_HEIGHT, m_nHeight);
    DDX_Radio(pDX, IDC_INCHES, m_nUnits);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(COptionsDialog, CDialog)
    //{{AFX_MSG_MAP(COptionsDialog)
    ON_BN_CLICKED(IDC_RESET, OnReset)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COptionsDialog message handlers

void COptionsDialog::OnReset()
{
    m_nWidth = 4;
    m_nHeight = 2;
```

```
    m_nUnits = 0;
    UpdateData (FALSE);
}
```

图 8-7 DlgDemo1 应用程序

下面是编写 DlgDemo1 的步骤。按照该步骤您也能创建这个应用程序。

1. 用 AppWizard 创建名为 DlgDemo1 的项目。在 AppWizard Step 1 对话框中,选择Single Document 作为应用程序类型,并复选选中 Document/View Architecture Support 框。在 Step 3 和 Step 4 对话框中取消选中下面的控件:
   - ActiveX Controls
   - 3D Controls
   - Docking Toolbar
   - Initial Status Bar

   在其他地方接受 AppWizard 中的默认值。
2. 在 CChildView 中添加表 8-4 中的几个成员变量。使它们成为受保护成员变量,并在视图的构造函数中将它们初始化。

表 8-4 添加的成员变量

变量名称	类型	初始值
m_nWidth	int	4
m_nHeight	int	2
m_nUnits	int	0

3. 实现视图的 OnPaint 函数。
4. 为了弥补 Visual C++ 6.0 的一个缺点,在主窗口类 CmainFrame 中添加 WM_CREATE 消息处理程序,并在其中添加代码创建视图。
5. 运用 Insert-Resource 命令给项目添加新对话框资源。通过右击 ResourceView 中的 IDD_DIALOG1,在上下文菜单中选择 Properties 并输入新 ID,将对话框的资源 ID 改变为 IDD_OPTIONS。这时,把对话框标题改为"Options"。
6. 编辑对话框,使它如图 8-6 所示的那样。表 8-5 中列出对话框中的控件和它们的 ID。一定要一个接一个地创建单选按钮,使它们赋得连续的控件 ID。OK 和 Cancel 按钮已提供给您了,所以不必再分别添加它们。

表 8-5　对话框中的控件及其 ID

控件类型	控件正文	控件 ID
静态控件	"&Width"	IDC_STATIC
静态控件	"&Height"	IDC_STATIC
编辑框	None	IDC_WIDTH
编辑框	None	IDC_HEIGHT
组框	"Units"	IDC_STATIC
单选按钮	"&Inches"	IDC_INCHES
单选按钮	"&Centimeters"	IDC_CENTIMETERS
单选按钮	"&Pixels"	IDC_PIXELS
按钮	"&Reset"	IDC_RESET
按钮	"OK"	IDOK
按钮	"Cancel"	IDCANCEL

7. 在 Layout 菜单中选择 Tab Order，并设定制表键控制次序，如图 8-2 所示。通过重选 Tab Order 退出制表键控制次序模式，在 Layout 菜单中选择 Test，并使用 Tab 键在对话框控件间跳转输入焦点，来测试制表键控制次序。注意必须在 Tab 键移动输入焦点前手动选中一个单选按钮。

8. 通过给组中第一个控件（Inches 单选按钮）和按 Tab 键控制次序在该组最后一个控件之后的第一个控件（OK 按钮）用样式 WS_GROUP 做个标记，使 Windows 了解到这 3 个单选按钮是一组。通过在 Layout 菜单中选择 Test，单击其中一个单选按钮并按几次向上或向下箭头键，您能够测试该组合方式。如果单选按钮适当地组合在一起，则输入焦点会在单选按钮间循环。

9. 在对话框编辑器中双击对话框，并运用 ClassWizard 派生出名为 COptionsDialog 的对话框类。在清除 ClassWizard 之后，COptionsDialog 应出现在 ClassView 窗口中。

10. 运用 ClassWizard 给对话框类添加 3 个整型成员变量：m_nUnits 连接到单选按钮 IDC_INCHES，m_nWidth 连接到 IDC_WIDTH 编辑控件，而 m_nHeight 连接到 IDC_HEIGHT 编辑控件。将 m_nWidth 和 m_nHeight 的最大值和最小值分别设置成 1 和 128。

注意，通过在 Add Member Variable 对话框的 Variable Type 域中选中 int，确保给编辑控件创建的是整型成员变量，而不是 CString 成员变量。CString 为默认值。如果弄错了，则使用 Delete Variable 按钮删除给该成员变量并重试一遍。

11. 打开菜单资源 IDR_MAINFRAME，在 File 菜单中添加 Options 命令。输入"&Options..."作为菜单项正文，ID_FILE_OPTIONS 作为命令 ID。

12. 在视图的 CPP 文件中添加下面语句:

```
#include "OptionsDialog.h"
```

然后在视图类中添加名为 OnFileOptions 的命令处理程序。该程序在选中 Options 命令时被调用。如图 8-7 实现该函数。

13. 在对话框类中添加一个 BN_CLICKED 处理程序,名为 OnReset。当单击 Reset 按钮时,它将 m_nWidth 设置为 4,m_nHeight 设置为 2,m_nUnits 设置为 0。如图 8-7 实现该处理程序。

14. 运行该应用程序,并使用 File/Options 命令显示 Options 对话框。通过输入不同的宽度和高度,并选择不同的尺寸单位测试手工编制的程序。

请注意 COptionsDialog::OnReset 是如何实现的。自己调用 UpdateData 在对话框控件和数据成员间实现数据传送是完全合法的。在这种情况下,通过 FALSE 参数调用UpdateData可以实现数据从成员变量到控件的传送。如果要把数据从控件读出并赋给成员变量,则给 UpdateData传递一个参数 TRUE。

8.2 无模式对话框

一旦掌握了模式对话框,您会发现无模式对话框在模式对话框的基础上只是稍稍有所改动。模式对话框和无模式对话框的共同点要比不同点更多。两者间的主要区别在于:

- 显示模式对话框需要调用 CDialog::DoModal,而显示无模式对话框则要调用 CDialog::Create。DoModal 要等到对话框被清除后才返回。而和 DoModal 不同,一旦建立对话框,Create 就返回。因此,Create 返回时对话框还处于显示状态。
- 清除无模式对话框要调用 DestroyWindow,而不是 EndDialog。禁止对无模式对话框调用 CDialog::OnOK 或 CDialog::OnCancel,因为两者都调用 EndDialog。
- 模式对话框类通常在栈上实例化,所以析构是自动实现的。而无模式对话框通过 new 实例化,所以该对话框对象不会过早地被清除。如果要确保清除对话框时删除无模式对话框对象,一种方法是在派生的对话框类中覆盖 CDialog::PostNcDestroy 并执行 delete this 语句。

在 MFC 提供的模式和无模式对话框间还有其他区别。例如:在使用了无模式对话框的 SDK 应用程序中,必须修改消息循环,调用::IsDialogMessage 将消息传送到对话框。而 MFC 应用程序则不需要这样的修改,因为::IsDialogMessage 调用是自动进行的。

一般来说,MFC 对对话框的处理具有普遍性,所以使用无模式对话框和使用模式对话框没有什么不同。让我们通过把 DlgDemo1 的对话框转换为无模式对话框来证明这一点。

DlgDemo2 应用程序

图 8-8 中的 DlgDemo2 应用程序与 DlgDemo1 在功能上是完全相同的,除却一点:那就是

Options 对话框不再是模式对话框了。按照惯例，OK 和 Cancel 按钮现在分别标着 Apply 和 Close。Apply 按钮把输入对话框的设置传给矩形，但并不清除对话框。和 DlgDemo1 中的 Cancel 按钮一样，Close 按钮将屏幕上的对话框清除，并忽略所有的变化。不管名称如何变化，按钮 ID 依旧是 IDOK 和 IDCANCEL。这意味着 OnOK 和 OnCancel 还可以用来处理按钮单击事件，而 Enter 和 Esc 键还能起这两个按钮的作用。

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__7040DB88_9039_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_MAINFRM_H__7040DB88_9039_11D2_8E53_006008A82731__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
    CMainFrame();
protected:
    DECLARE_DYNAMIC(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
        AFX_CMDHANDLERINFO* pHandlerInfo);
    //}}AFX_VIRTUAL

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

```
#endif
    CChildView    m_wndView;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg void OnSetFocus(CWnd * pOldWnd);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    afx_msg LRESULT OnApply (WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnDialogDestroyed (WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(AFX_MAINFRM_H__7040DB88_9039_11D2_8E53_006008A82731__INCLUDED_)
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "DlgDemo2.h"
#include "OptionsDialog.h"
#include "MainFrm.h"

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

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_SETFOCUS()
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
```

```
    ON_MESSAGE (WM_USER_APPLY, OnApply)
    ON_MESSAGE (WM_USER_DIALOG_DESTROYED, OnDialogDestroyed)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    cs.lpszClass = AfxRegisterWndClass(0);
    return TRUEX;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnSetFocus(CWnd * pOldWnd)
{
    // forward focus to the view window
    m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void * pExtra,
    AFX_CMDHANDLERINFO * pHandlerInfo)
```

```
}
    // let the view have first crack at the command
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    // otherwise, do default handling
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
        CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL))
        return -1;
    return 0;
}

LRESULT CMainFrame::OnApply (WPARAM wParam, LPARAM lParam)
{
    m_wndView.SendMessage (WM_USER_APPLY, wParam, lParam);
    return 0;
}

LRESULT CMainFrame::OnDialogDestroyed (WPARAM wParam, LPARAM lParam)
{
    m_wndView.SendMessage (WM_USER_DIALOG_DESTROYED, wParam, lParam);
    return 0;
}
```

ChildView.h

```
// ChildView.h : interface of the CChildView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDVIEW_H__7040DB8A_9039_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_CHILDVIEW_H__7040DB8A_9039_11D2_8E53_006008A82731__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CChildView window

class CChildView : public CWnd
{
```

```
// Construction
public:
    CChildView();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildView)
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildView();

    // Generated message map functions
protected:
    COptionsDialog * m_pDlg;
    int m_nUnits;
    int m_nHeight;
    int m_nWidth;
    //{{AFX_MSG(CChildView)
    afx_msg void OnPaint();
    afx_msg void OnFileOptions();
    //}}AFX_MSG
    afx_msg LRESULT OnApply (WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnDialogDestroyed (WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(
//    AFX_CHILDVIEW_H__7040DB8A_9039_11D2_8E53_006008A82731__INCLUDED_)
```

ChildView.cpp

```
// ChildView.cpp : implementation of the CChildView class
```

```
//

#include "stdafx.h"
#include "DlgDemo2.h"
#include "OptionsDialog.h"
#include "ChildView.h"

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

/////////////////////////////////////////////////////////////////////////////
// CChildView

CChildView::CChildView()
{
    m_nWidth = 4;
    m_nHeight = 2;
    m_nUnits = 0;
    m_pDlg = NULL;
}
CChildView::~CChildView()
{
}

BEGIN_MESSAGE_MAP(CChildView,CWnd )
    //{{AFX_MSG_MAP(CChildView)
    ON_WM_PAINT()
    ON_COMMAND(ID_FILE_OPTIONS, OnFileOptions)
    //}}AFX_MSG_MAP
    ON_MESSAGE (WM_USER_APPLY, OnApply)
    ON_MESSAGE (WM_USER_DIALOG_DESTROYED, OnDialogDestroyed)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildView message handlers

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;
    cs.dwExStyle|= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
        ::LoadCursor(NULL, IDC_ARROW), HBRUSH(COLOR_WINDOW+1), NULL);

    return TRUE;
```

```
}

void CChildView::OnPaint()
{
    CPaintDC dc(this); // Device context for painting.

    CBrush brush (RGB (255, 0, 255));
    CBrush * pOldBrush = dc.SelectObject (&brush);

    switch (m_nUnits) {

    case 0: // Inches.
        dc.SetMapMode (MM_LOENGLISH);
        dc.Rectangle (0, 0, m_nWidth * 100, -m_nHeight * 100);
        break;

    case 1: // Centimeters.
        dc.SetMapMode (MM_LOMETRIC);
        dc.Rectangle (0, 0, m_nWidth * 100, -m_nHeight * 100);
        break;

    case 2: // Pixels.
        dc.SetMapMode (MM_TEXT);
        dc.Rectangle (0, 0, m_nWidth, m_nHeight);
        break;
    }
    dc.SelectObject (pOldBrush);
}
void CChildView::OnFileOptions()
{
    //
    // If the dialog box already exists, display it.
    //
    if (m_pDlg != NULL)
        m_pDlg -> SetFocus ();

    //
    // If the dialog box doesn't already exist, create it.
    //
    else {
        m_pDlg = new COptionsDialog;
        m_pDlg -> m_nWidth = m_nWidth;
        m_pDlg -> m_nHeight = m_nHeight;
        m_pDlg -> m_nUnits = m_nUnits;
        m_pDlg -> Create (IDD_OPTIONS);
        m_pDlg -> ShowWindow (SW_SHOW);
    }
}
```

```
LRESULT CChildView::OnApply (WPARAM wParam, LPARAM lParam)
{
    RECTPROP* prp = (RECTPROP*) lParam;
    m_nWidth = prp->nWidth;
    m_nHeight = prp->nHeight;
    m_nUnits = prp->nUnits;
    Invalidate ();
    return 0;
}

LRESULT CChildView::OnDialogDestroyed (WPARAM wParam, LPARAM lParam)
{
    m_pDlg = NULL;
    return 0;
}
```

OptionsDialog.h

```
#if !defined(AFX_OPTIONSDIALOG_H__7040DB90_9039_11D2_8E53_006008A82731__
INCLUDED_)
#define AFX_OPTIONSDIALOG_H__7040DB90_9039_11D2_8E53_006008A82731__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// OptionsDialog.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// COptionsDialog dialog

class COptionsDialog : public CDialog
{
// Construction
public:
    COptionsDialog(CWnd* pParent = NULL);   // standard constructor
// Dialog Data
    //{{AFX_DATA(COptionsDialog)
    enum { IDD = IDD_OPTIONS };
    int        m_nWidth;
    int        m_nHeight;
    int        m_nUnits;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(COptionsDialog)
    protected:
```

```
    virtual void DoDataExchange(CDataExchange * pDX);    // DDX/DDV support
    virtual void PostNcDestroy();
    //{{AFX_VIRTUAL
    virtual void OnOK();
    virtual void OnCancel();

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(COptionsDialog)
    afx_msg void OnReset();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(
//  AFX_OPTIONSDIALOG_H__7040DB90_9039_11D2_8E53_006008A82731__INCLUDED_)
```

OptionsDialog.cpp

```
// OptionsDialog.cpp : implementation file
//

#include "stdafx.h"
#include "DlgDemo2.h"
#include "OptionsDialog.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COptionsDialog dialog

COptionsDialog::COptionsDialog(CWnd* pParent /*=NULL*/)
    : CDialog(COptionsDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(COptionsDialog)
    m_nWidth = 0;
    m_nHeight = 0;
    m_nUnits = -1;
```

```
    //}}AFX_DATA_INIT
}

void COptionsDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(COptionsDialog)
    DDX_Text(pDX, IDC_WIDTH, m_nWidth);
    DDX_Text(pDX, IDC_HEIGHT, m_nHeight);
    DDX_Radio(pDX, IDC_INCHES, m_nUnits);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(COptionsDialog, CDialog)
    //{{AFX_MSG_MAP(COptionsDialog)
    ON_BN_CLICKED(IDC_RESET, OnReset)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COptionsDialog message handlers

void COptionsDialog::OnReset()
{
    m_nWidth = 4;
    m_nHeight = 2;
    m_nUnits = 0;
    UpdateData(FALSE);
}

void COptionsDialog::OnOK()
{
    UpdateData(TRUE);

    RECTPROP rp;
    rp.nWidth = m_nWidth;
    rp.nHeight = m_nHeight;
    rp.nUnits = m_nUnits;
    AfxGetMainWnd()->SendMessage(WM_USER_APPLY, 0, (LPARAM) &rp);
}

void COptionsDialog::OnCancel()
{
    DestroyWindow();
}
```

```
void COptionsDialog::PostNcDestroy ()
{
    CDialog::PostNcDestroy ();
    AfxGetMainWnd () -> SendMessage (WM_USER_DIALOG_DESTROYED, 0, 0);
    delete this;
}
```

图 8-8　DlgDemo2 应用程序

和以前一样，在 File 菜单中选中 Options 会调用 Options 对话框。下面是 OnFileOptions 中的一段代码。该函数构造对话框对象，将对话框数据成员初始化，并创建该对话框：

```
m_pDlg = new COptionsDialog;
m_pDlg->m_nWidth = m_nWidth;
m_pDlg->m_nHeight = m_nHeight;
m_pDlg->m_nUnits = m_nUnits;
m_pDlg->Create (IDD_OPTIONS);
m_pDlg->ShowWindow (SW_SHOW);
```

为了避免自动析构，对话框对象在堆上创建。对话框指针保存在 CChildView::m_pDlg 中。首先 CChildView 的构造函数会把 CChildView::m_pDlg 初始化为 NULL，而在清除对话框时又将它重置为 NULL。通过检查 m_pDlg 是否为非 NULL 值，CChildView 的任何一个成员函数都能判断对话框当前是否处于显示状态。这种方法看来很有用。因为在创建 Options 对话框前，OnFileOptions 可以检查 m_pDlg，确定对话框是否已经显示。如果答案是肯定的，OnFileOptions利用 m_pDlg 指针将焦点设置在现存对话框中，而不是创建一个新对话框：

```
if (m_pDlg != NULL)
    m_pDlg->SetFocus ();
```

如果没有这个判断，即使已经有实例存在，每次选择 File-Options 都会创建一个新的对话框实例。正常情况下没理由在屏幕上同时出现两个或更多个同样的对话框，所以除非特殊情况允许，一般禁止用户打开多个无模式对话框实例。

处理 Apply 和 Close 按钮

通过 MFC 使用模式和无模式对话框的一个根本区别在于对话框类是如何处理 OnOK 和 OnCancel 的。因为在 CDialog 中默认实现 OnCancel 需要调用 EndDialog 关闭对话框并返回 IDCANCEL，所以模式对话框类几乎不覆盖 OnCancel。并且因为 CDialog 中实现 OnOK 需要调用 UpdateData 在清除对话框前更新对话框数据成员，所以也很少覆盖 OnOK。如果对话框控件和数据成员是通过 DDX 或 DDV 联系起来的，则 CDialog::OnOK 提供的默认方式就足够使了。

无模式对话框则相反，它几乎总要覆盖 OnOK 和 OnCancel。正如前面曾提到的那样，因为无模式对话框是用 DestroyWindow 清除的，而不是 EndDialog。所以在无模式对话框中必须

避免调用 CDialog::OnOK 和 CDialog::OnCancel。如果对话框中有按钮具有 ID IDOK,则要覆盖 OnOK。因为不管对话框是否包含 Cancel 按钮,只要用户按下 Esc 键或单击对话框关闭按钮,则有 IDCANCEL 消息发送,所以总要覆盖 OnCancel。

因为单击 DlgDemo2 的 Apply 和 Close 按钮会导致调用 OnOK 和 OnCancel,所以两个函数都覆盖在 COptionsDialog 中。COptionsDialog::OnOK 包含下列语句:

```
UpdateData (TRUE);

RECTPROP rp;
rp.nWidth = m_nWidth;
rp.nHeight = m_nHeight;
rp.nUnits = m_nUnits;

AfxGetMainWnd ()->SendMessage (WM_USER_APPLY, 0, (LPARAM) &rp);
```

第一个语句更新对话框的成员变量,使它们和控件的当前状态一致。因为对使用了 DDX 或 DDV 的无模式对话框来说,调用 CDialog::OnOK 并由该函数再调用 UpdateData 是不可能的,所以无模式对话框必须自己调用 UpdateData。下一段语句将在 Stdafx.h 中声明的 RECTPROP 结构实例化,并把对话框数据成员中的新设置复制到该数据结构。最后一个语句发送消息给应用程序主窗口,通知主窗口把 RECTPROP 结构中的设置值传给对话框。WM_USER_APPLY 是一个用户自定义消息,它在 Stdafx.h 中的定义方式如下:

```
#define WM_USER_APPLY WM_USER + 0x100
```

WM_USER 在头文件 *Winuser.h* 中定义为 0x400,确定了应用程序可用的消息 ID 的下界,避免和标准 Windows 消息如 WM_CREATE 和 WM_PAINT 的消息 ID 冲突。应用程序可自由使用的消息 ID 是从 WM_USER 的 0x400 到 0x7FFF。在这个范围中的消息称为"用户自定义消息"。因为对话框用到一些该范围中的消息 ID,所以为避免冲突,DlgDemo2 强制把 0x100 加入 WM_USER。

通过 SendMessage 传送的消息包含两个可用来传递数据的参数:一个是类型为 WPARAM 的 32 位值,另一个是类型为 LPARAM 的 32 位值。COptionsDialog::OnOK 给主窗口发送消息时,它同时还传送一个指向 RECTPROP 结构(该结构包含从对话框读取的设置信息)的指针。主窗口用 CMainFrame::OnApply 处理该消息。在消息映射表中用下面语句引用 CMainFrame::OnApply:

```
ON_MESSAGE (WM_USER_APPLY, OnApply);
```

一旦被激活,OnApply 把消息传送给视图:

```
LRESULT CMainFrame::OnApply (WPARAM wParam, LPARAM lParam)
{
    m_wndView.SendMessage (WM_USER_APPLY, wParam, lParam);
```

```
    return 0;
}
```

随后，CChildView::OnApply 把数据结构中的值复制到自己的数据成员中。然后清除旧视图，重画带有新设置的视图。

```
LRESULT CChildView::OnApply (WPARAM wParam, LPARAM lParam)
{
    RECTPROP * prp = (RECTPROP *) lParam;
    m_nWidth = prp->nWidth;
    m_nHeight = prp->nHeight;
    m_nUnits = prp->nUnits;
    Invalidate();
    return 0;
}
```

用户自定义消息处理程序的返回值通过 SendMessage 返回给调用者。DlgDemo2 中该返回值无意义，所以 CMainFrame::OnApply 和 CChildView::OnApply 都返回 0。

COptionsDialog::OnCancel 只包含一个语句。这个语句调用 DestroyWindow 清除对话框。最后该动作激活 COptionsDialog::PostNcDestroy。COptionsDialog::PostNcDestroy 是这样实现的：

```
void COptionsDialog::PostNcDestroy()
{
    CDialog::PostNcDestroy();
    AfxGetMainWnd()->SendMessage(WM_USER_DIALOG_DESTROYED, 0, 0);
    delete this;
}
```

SendMessage 向主窗口发送一个不同的用户自定义消息。

主窗口的 WM_USER_DIALOG_DESTROYED 处理程序，CMainFrame::OnDialogDestroyed，将消息传送给视图，而视图的 WM_USER_DIALOG_DESTROYED 处理程序作为响应将 m_pDlg设置为 NULL。到这里，工作基本上完成了，最后 PostNcDestroy 通过执行 delete this 语句删除由 CChildView::OnFileOptions 创建的对话框对象结束所有任务。

8.3　用对话框作为主窗口

如果应用程序的主要用户界面是对话框式的控件的集合，那么您应该考虑用对话框作主窗口。Charles Petzold 在他的 Programming Windows 书中通过 HEXCALC 程序使这种技巧成为经典方法。许多开发人员也运用类似技巧创建了一些小但实用的应用程序，其主窗口在对话框模板中定义起来要比在 OnCreate 处理程序编程规范下更容易些。

在 AppWizard 的帮助下,编写基于对话框的应用程序非常容易。在 AppWizard 的 Step 1 对话框中,有一个选项是标着 Dialog Based 的单选按钮。选中该按钮将促使 AppWizard 生成一个应用程序,其主窗口是一个对话框。AppWizard 替您创建对话框资源,并从 CDialog 派生出对话框类。它还发布特别版本的 InitInstance,使对话框类实例化,并在应用程序启动时调用该类的 DoModal 函数将对话框显示在屏幕上。您所要做的是在资源编辑器中往对话框添加控件,并编写消息处理程序响应控件事件。而其他一切事情则由 AppWizard 生成的代码完成。

图 8-9 所示的 DlgCalc 应用程序是一个基于对话框的 MFC 应用程序。DlgCalc 是一个计算器小应用程序。它和 Windows 提供的计算器小应用程序不同,主要区别在于:它使用了后缀记法,也叫做 reverse Polish notation,或 RPN。后缀记法是 Hewlett-Packard 计算器使用的数据输入形式。一旦习惯使用后缀记法,您就不想再用常规计算器了。

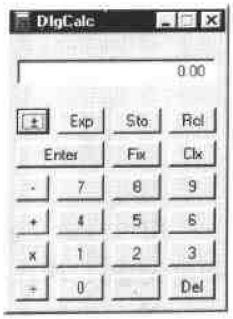

图 8-9 DlgCalc 窗口

DlgCalc 的资源代码请参见图 8-10。主窗口是在 CDlgCalcApp::InitInstance 中创建的。该函数构造一个 CDlgCalcDlg 对象,将对象的地址复制到应用程序对象的 m_pMainWnd 数据成员中,并调用 DoModal 显示窗口:

```
CDlgCalcDlg dlg;
m_pMainWnd = &dlg;
dlg.DoModal();
```

CDlgCalcDlg 是 AppWizard 从 CDialog 派生出的对话框类。由它创建的窗口实际上就是一个对话框,但因为它没有父窗口并且地址又存放在 m_pMainWnd 中,所以它同时又是主窗口。值得注意的是:我删除了 AppWizard 放在 InitInstance 中的一些代码,这些代码用来测试 DoModal 的返回值,但在这个应用程序中并没有什么用处。我还删除了 AppWizard 放在对话框类中的 WM_QUERYDRAGICON 处理程序,以及 AppWizard 生成的 OnPaint 程序代码,该程序代码在窗口最小化时画出应用程序图标。我之所以这样做是因为除非您的应用程序要在老版本的 Windows(尤其是使用 Windows 3.x 型外壳程序的老版本)上运行,否则根本用不着这两个程序。

DlgCalc.h

```
// DlgCalc.h : main header file for the DLGCALC application
//

#if !defined(AFX_DLGCALC_H__F42970C4_9047_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_DLGCALC_H__F42970C4_9047_11D2_8E53_006008A82731__INCLUDED_

#if _MSC_VER > 1000
#pragma once
```

```
#endif //_MSC_VER > 1000
#ifndef  AFXWIN_H
    #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"        // main symbols

/////////////////////////////////////////////////////////////////////////////
// CDlgCalcApp:
// See DlgCalc.cpp for the implementation of this class
//

class CDlgCalcApp : public CWinApp
{
public:
    CDlgCalcApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDlgCalcApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CDlgCalcApp)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(AFX_DLGCALC_H__F42970C4_9047_11D2_8E53_006008A82731__INCLUDED_)
```

DlgCalc.cpp

```
// DlgCalc.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "DlgCalc.h"
#include "DlgCalcDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
```

```
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgCalcApp

BEGIN_MESSAGE_MAP(CDlgCalcApp, CWinApp)
    //{{AFX_MSG_MAP(CDlgCalcApp)
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDlgCalcApp construction

CDlgCalcApp::CDlgCalcApp()
{
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CDlgCalcApp object

CDlgCalcApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CDlgCalcApp initialization

BOOL CDlgCalcApp::InitInstance()
{
    CDlgCalcDlg dlg;
    m_pMainWnd = &dlg;
    dlg.DoModal();
    return FALSE;
}
```

DlgCalcDlg.h

```
// DlgCalcDlg.h : header file
//

#if !defined(AFX_DLGCALCDLG_H__F42970C6_9047_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_DLGCALCDLG_H__F42970C6_9047_11D2_8E53_006008A82731__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

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

```
// CDlgCalcDlg dialog

class CDlgCalcDlg : public CDialog
{
// Construction
public:
    void UpdateDisplay (LPCTSTR pszDisplay);
    CDlgCalcDlg(CWnd * pParent = NULL);    // standard constructor

// Dialog Data
    //{{AFX_DATA(CDlgCalcDlg)
    enum { IDD = IDD_DLGCALC_DIALOG };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDlgCalcDlg)
    public:
    virtual BOOL PreTranslateMessage(MSG * pMsg);
    protected:
    virtual void DoDataExchange(CDataExchange * pDX);    // DDX/DDV support
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
    //}}AFX_VIRTUAL

// Implementation
protected:
    void DropStack();
    void LiftStack();
    void DisplayXRegister();

    double m_dblStack[4];
    double m_dblMemory;
    CString m_strDisplay;
    CString m_strFormat;
    CRect m_rect;
    int m_cxChar;
    int m_cyChar;

    BOOL m_bFixPending;
    BOOL m_bErrorFlag;

    BOOL m_bDecimalInString;
    BOOL m_bStackLiftEnabled;
    BOOL m_bNewX;

    HICON m_hIcon;
    HACCEL m_hAccel;

    // Generated message map functions
```

```
    //{{AFX_MSG(CDlgCalcDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg void OnAdd();
    afx_msg void OnSubtract();
    afx_msg void OnMultiply();
    afx_msg void OnDivide();
    afx_msg void OnEnter();
    afx_msg void OnChangeSign();
    afx_msg void OnExponent();
    afx_msg void OnStore();
    afx_msg void OnRecall();
    afx_msg void OnFix();
    afx_msg void OnClear();
    afx_msg void OnDecimal();
    afx_msg void OnDelete();
    //}}AFX_MSG
    afx_msg void OnDigit(UINT nID);
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(
//     AFX_DLGCALCDLG_H__F42970C6_9047_11D2_8E53_006008A82731__INCLUDED_)
```

DlgCalcDlg.cpp

```
// DlgCalcDlg.cpp : implementation file
//

#include "stdafx.h"
#include "DlgCalc.h"
#include "DlgCalcDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgCalcDlg dialog

CDlgCalcDlg::CDlgCalcDlg(CWnd* pParent /* = NULL */)
```

```
    : CDialog(CDlgCalcDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CDlgCalcDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_hAccel = ::LoadAccelerators(AfxGetInstanceHandle(),
        MAKEINTRESOURCE(IDR_ACCEL));

    m_bFixPending = FALSE;
    m_bErrorFlag = FALSE;
    m_bDecimalInString = FALSE;
    m_bStackLiftEnabled = FALSE;
    m_bNewX = TRUE;

    for (int i = 0; i < 4; i++)
        m_dblStack[i] = 0.0;
    m_dbMemory = 0.0;
    m_strFormat = _T("%0.2f");
}
void CDlgCalcDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlgCalcDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgCalcDlg, CDialog)
    //{{AFX_MSG_MAP(CDlgCalcDlg)
    ON_WM_PAINT()
    ON_BN_CLICKED(IDC_ADD, OnAdd)
    ON_BN_CLICKED(IDC_SUBTRACT, OnSubtract)
    ON_BN_CLICKED(IDC_MULTIPLY, OnMultiply)
    ON_BN_CLICKED(IDC_DIVIDE, OnDivide)

    ON_BN_CLICKED(IDC_ENTER, OnEnter)
    ON_BN_CLICKED(IDC_CHGSIGN, OnChangeSign)
    ON_BN_CLICKED(IDC_EXP, OnExponent)
    ON_BN_CLICKED(IDC_STO, OnStore)
    ON_BN_CLICKED(IDC_RCL, OnRecall)
    ON_BN_CLICKED(IDC_FIX, OnFix)
    ON_BN_CLICKED(IDC_CLX, OnClear)
    ON_BN_CLICKED(IDC_DECIMAL, OnDecimal)
    ON_BN_CLICKED(IDC_DEL, OnDelete)
    //}}AFX_MSG_MAP
    ON_CONTROL_RANGE(BN_CLICKED, IDC_0, IDC_9, OnDigit)
```

```
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CDlgCalcDlg message handlers

BOOL CDlgCalcDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    //
    // Set the application's icon.
    //
    SetIcon(m_hIcon, TRUE);
    SetIcon(m_hIcon, FALSE);

    //
    // Remove the Size and Maximize commands from the system menu.
    //
    CMenu* pMenu = GetSystemMenu(FALSE);
    pMenu->DeleteMenu(SC_SIZE, MF_BYCOMMAND);
    pMenu->DeleteMenu(SC_MAXIMIZE, MF_BYCOMMAND);

    //
    // Initialize m_rect with the coordinates of the control representing
    // the calculator's output window. Then destroy the control.
    //
    CWnd* pWnd = GetDlgItem(IDC_DISPLAYRECT);
    pWnd->GetWindowRect(&m_rect);
    pWnd->DestroyWindow();
    ScreenToClient(&m_rect);

    //
    // Initialize m_cxChar and m_cyChar with the average character width
    // and height.
    //
    TEXTMETRIC tm;
    CClientDC dc(this);
    dc.GetTextMetrics(&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight - tm.tmDescent;

    //
    // Initialize the calculator's output window and return.
    //
    DisplayXRegister();
    return TRUE;
}

void CDlgCalcDlg::OnPaint()
```

```
{
    CPaintDC dc(this);
    dc.DrawEdge(m_rect, EDGE_SUNKEN, BF_RECT);
    UpdateDisplay(m_strDisplay);
}

BOOL CDlgCalcDlg::PreTranslateMessage(MSG * pMsg)
{
    if (m_hAccel != NULL)
        if (::TranslateAccelerator(m_hWnd, m_hAccel, pMsg))
            return TRUE;

    return CDialog::PreTranslateMessage(pMsg);
}

BOOL CDlgCalcDlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    int nID = (int) LOWORD(wParam);

    if (m_bErrorFlag && (nID != IDC_CLX)) {
        ::MessageBeep(MB_ICONASTERISK);
        return TRUE;
    }

    if (m_bFixPending &&
        ((nID < IDC_0)||(nID > IDC_9)) &&
        (nID != IDC_CLX)) {
        ::MessageBeep(MB_ICONASTERISK);
        return TRUE;
    }
    return CDialog::OnCommand(wParam, lParam);
}
void CDlgCalcDlg::OnDigit(UINT nID)
{
    TCHAR cDigit = (char) nID;

    if (m_bFixPending) {
        m_strFormat.SetAt(3, cDigit - IDC_0 + 0x30);
        DisplayXRegister();
        m_bFixPending = FALSE;
        m_bStackLiftEnabled = TRUE;
        m_bNewX = TRUE;
        return;
    }

    if (m_bNewX) {
        m_bNewX = FALSE;
        if (m_bStackLiftEnabled) {
```

```
            m_bStackLiftEnabled = FALSE;
            LiftStack();
        }
        m_bDecimalInString = FALSE;
        m_strDisplay.Empty();
    }

    int nLength = m_strDisplay.GetLength();
    if ((nLength == MAXCHARS) ||
        ((nLength == (MAXCHARS - 10)) && !m_bDecimalInString))
        ::MessageBeep(MB_ICONASTERISK);
    else {
        m_strDisplay += (cDigit - IDC_0 + 0x30);
        UpdateDisplay(m_strDisplay);
        m_dblStack[0] = _tcstod(m_strDisplay.GetBuffer(0), NULL);
    }
}

void CDlgCalcDlg::OnAdd()
{
    m_dblStack[0] += m_dblStack[1];
    DisplayXRegister();
    DropStack();
    m_bStackLiftEnabled = TRUE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnSubtract()
{
    m_dblStack[0] = m_dblStack[1] - m_dblStack[0];
    DisplayXRegister();
    DropStack();
    m_bStackLiftEnabled = TRUE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnMultiply()
{
    m_dblStack[0] *= m_dblStack[1];
    DisplayXRegister();
    DropStack();
    m_bStackLiftEnabled = TRUE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnDivide()
{
    if (m_dblStack[0] == 0.0) {
```

```
        m_bErrorFlag = TRUE;
        ::MessageBeep (MB_ICONASTERISK);
        UpdateDisplay (CString (_T ("Divide by zero")));
    }
    else {
        m_dblStack[0] = m_dblStack[1] / m_dblStack[0];
        DisplayXRegister ();
        DropStack ();
        m_bStackLiftEnabled = TRUE;
        m_bNewX = TRUE;
    }
}

void CDlgCalcDlg::OnEnter()
{
    LiftStack ();
    DisplayXRegister ();
    m_bStackLiftEnabled = FALSE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnChangeSign()
{
    if (m_dblStack[0] != 0.0) {
        m_dblStack[0] = -m_dblStack[0];
        if (m_strDisplay[0] == _T ('-')) {
            int nLength = m_strDisplay.GetLength ();
            m_strDisplay = m_strDisplay.Right (nLength - 1);
        }

        else
            m_strDisplay = _T ("-") + m_strDisplay;
        UpdateDisplay (m_strDisplay);
    }
}

void CDlgCalcDlg::OnExponent()
{
    if (((m_dblStack[1] == 0.0) && (m_dblStack[0] < 0.0)) ||
        ((m_dblStack[1] == 0.0) && (m_dblStack[0] == 0.0)) ||
        ((m_dblStack[1] < 0.0) &&
        (floor (m_dblStack[0]) != m_dblStack[0]))) {
        m_bErrorFlag = TRUE;
        ::MessageBeep (MB_ICONASTERISK);
        UpdateDisplay (CString (_T ("Invalid operation")));
    }
    else {
```

```
        m_dblStack[0] = pow (m_dblStack[1], m_dblStack[0]);
        DisplayXRegister ();
        DropStack ();
        m_bStackLiftEnabled = TRUE;
        m_bNewX = TRUE;
    }
}

void CDlgCalcDlg::OnStore()
{
    DisplayXRegister ();
    m_dblMemory = m_dblStack[0];
    m_bStackLiftEnabled = TRUE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnRecall()
{
    LiftStack ();
    m_dblStack[0] = m_dblMemory;
    DisplayXRegister ();
    m_bStackLiftEnabled = TRUE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnFix()
{
    m_bFixPending = TRUE;
}
void CDlgCalcDlg::OnClear()
{
    if (m_bFixPending) {
        m_bFixPending = FALSE;
        return;
    }

    m_bErrorFlag = FALSE;
    m_dblStack[0] = 0.0;
    DisplayXRegister ();
    m_bStackLiftEnabled = FALSE;
    m_bNewX = TRUE;
}

void CDlgCalcDlg::OnDecimal()
{
    if (m_bNewX) {
        m_bNewX = FALSE;
        if (m_bStackLiftEnabled) {
```

```
            m_bStackLiftEnabled = FALSE;
            LiftStack();
        }
        m_bDecimalInString = FALSE;
        m_strDisplay.Empty();
    }

    int nLength = m_strDisplay.GetLength();
    if ((nLength == MAXCHARS)||(m_bDecimalInString))
        ::MessageBeep(MB_ICONASTERISK);
    else {
        m_bDecimalInString = TRUE;
        m_strDisplay += (char) 0x2E;
        UpdateDisplay(m_strDisplay);
        m_dblStack[0] = strtod(m_strDisplay.GetBuffer(0), NULL);
    }
}

void CDlgCalcDlg::OnDelete()
{
    int nLength = m_strDisplay.GetLength();

    if (! m_bNewX && (nLength != 0)) {
        if (m_strDisplay[nLength-1] == _T('.'))
            m_bDecimalInString = FALSE;
        m_strDisplay = m_strDisplay.Left(nLength-1);

        UpdateDisplay(m_strDisplay);
        m_dblStack[0] = strtod(m_strDisplay.GetBuffer(0), NULL);
    }
}

void CDlgCalcDlg::LiftStack()
{
    for (int i=3; i>0; i--)
        m_dblStack[i] = m_dblStack[i-1];
}

void CDlgCalcDlg::DropStack()
{
    for (int i=1; i<3; i++)
        m_dblStack[i] = m_dblStack[i+1];
}

void CDlgCalcDlg::DisplayXRegister()
{
    double dblVal = m_dblStack[0];

    if ((dblVal >= 1000000000000.0) || (dblVal <= -1000000000000.0)) {
```

```
        UpdateDisplay (CString ( T ("Overflow error")));
        m_bErrorFlag = TRUE;
        MessageBeep (MB_ICONASTERISK);
    }
    else {
        m_strDisplay.Format (m_strFormat, dblVal);
        UpdateDisplay (m_strDisplay);
    }
}

void CDlgCalcDlg::UpdateDisplay(LPCTSTR pszDisplay)
{
    CClientDC dc (this);
    CFont * pOldFont = dc.SelectObject (GetFont ());
    CSize size = dc.GetTextExtent (pszDisplay);

    CRect rect = m_rect;
    rect.InflateRect (-2, -2);
    int x = rect.right - size.cx - m_cxChar;
    int y = rect.top + ((rect.Height () - m_cyChar) / 2);

    dc.ExtTextOut (x, y, ETO_OPAQUE, rect, pszDisplay, NULL);
    dc.SelectObject (pOldFont);
}
```

图 8-10 DlgCalc 应用程序

在默认方式下,由 AppWizard 创建的基于对话框的应用程序中,主窗口没有最小化按钮。通过在对话框编辑器中打开对话框,并在对话框属性表中复选选中 Minimize Button,我在主窗口标题栏中添加了一个最小化按钮。

DlgCalcDlg.cpp 中大部分程序代码是用来处理单击计算器按钮事件的。因为有了这些代码,DlgCalc 用起来和真的 RPN 计算器一样。例如:要算 2 加 2,您需敲入

```
2 <Enter> 2 +
```

如果要算 3.46 乘 9,再加上 13,除以 10,最后再求所得结果的 2.5 次幂,您需敲入

```
3.46 <Enter> 9 * 13 + 10 / 2.5 <Exp>
```

Sto 键将计算器显示的数字复制到内存中(即存储起来),并由 Rcl 再调出。Clx 清除计算器屏幕显示的数字("Clx"中的"x"指代计算器的 X 寄存器,其内容一般都显示在计算器屏幕上),而 ± 按钮用来改变当前被显示的数字的符号。Fix 设定小数点后需要显示的位数。如果要把两个小数位改成四位,则单击 Fix,然后再单击 4 按钮。Del 按钮删除屏幕显示中最右边一个字符。对于计算器上的每个按钮,在键盘上都有对应的键,请参见表8-6。对应于 ± 按钮的 P 键是加(plus)或减(minus)的简略助记符。大部分用户都感到用鼠标单击计算器

按钮很慢，所以键盘快捷方式成为该应用程序用户界面的一个重要环节。

表 8-6　与 DlgCalc 计算器按钮相对应的键

按钮	键	按钮	键
±	P	0 - 9	0 - 9
Exp	E	-	-
Sto	S	+	+
Rcl	R	x	*
Enter	Enter	÷	/
Fix	F	.	.
Clx	C	Del	Del，Backspace

处理键盘消息

因为对话框要在 Windows 键盘接口基础上再实现自己的键盘接口，而这种情况并不多见，所以有必要仔细考察一下 DlgCalc 的键盘处理逻辑。

在对话框中处理击键事件有一个根本问题，那就是 WM_CHAR 消息是由::IsDialogMessage 处理的，该函数从每个 MFC 对话框的消息循环中调用。您可以给对话框类添加一个 OnChar 处理程序，但是如果::IsDialogMessage 在::TranslateMessage 之前发现键盘消息，则该处理程序根本没有机会被调用。另一个问题是：一旦某控件得到输入焦点，则后来发生的键盘消息就会被传送到控件，而不是对话框窗口。

为了避免这些问题，我决定使用加速键处理键盘输入。首先我通过在 Visual C++ 的 Insert菜单中选中 Resource 命令，并双击 Accelerator，创建了一个加速键资源。然后给计算器上的所有键都添加了加速键，1 对应着 IDC_1 按钮，2 对应着 IDC_2 按钮，等等。再下一步，我在 CDlgCalcDlg 中添加了一个 HACCEL 成员变量，并将下列语句插入 CdlgCalcDlg 的构造函数，用来加载加速键：

```
m_hAccel = ::LoadAccelerators (AfxGetInstanceHandle (),
    MAKEINTRESOURCE (IDR_ACCELL));
```

最后，覆盖 PreTranslateMessage 并用另一版本取代它，该版本对对话框接收到的每个消息都调用::TranslateAccelerator。

```
BOOL CCalcDialog::PreTranslateMessage (MSG * pMsg)
{
    if (m_hAccel != NULL)
        if (::TranslateAccelerator (m_hWnd, m_hAccel, pMsg))
            return TRUE;

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

这样，::TranslateAccelerator 甚至能在::IsDialogMessage 之前发现键盘消息，并且和加速键对应的各消息会自动转换为 WM_COMMAND 消息。由于赋给加速键和计算器按钮的命令 ID 一样，所以处理单击"和"按键事件的是同一个 ON_BN_CLICKED 处理程序。

预处理 WM_COMMAND 消息

在控件发出的 WM_COMMAND 消息经由类的消息映射表之前，MFC 调用类的虚函数 OnCommand。默认实现 OnCommand 是指令传递体系的起点，它可以确保一切和运行中应用程序有关的对象，包括文档、视图和文档/视图应用程序中用到的应用程序对象，能发现消息并有机会处理消息。如果需要，通过覆盖 OnCommand 应用程序还能够预处理 WM_COMMAND 消息。预处理完成后，应用程序能调用基类的 OnCommand 函数继续传递消息，进行下面正常的消息处理，或者可以通过不经调用基类直接返回"吃"掉消息。不调用基类的 OnCommand处理程序应返回 TRUE，通知 Windows 消息处理完毕。

DlgCalc 还做了 MFC 应用程序不常做的事：它覆盖 OnCommand，并且如果一对 CDlgCalcDlg 成员变量 m_bErrorFlag 或 m_bFixPending 中有一个为非零值，DlgCalc 则排除选中的 WM_COMMAND 消息。CDlgCalcDialog::OnCommand 从获取控件的 ID 开始，该控件根据 MFC 传递给它的 wParam 值的低位字生成消息：

```
int nID = (int) LOWORD (wParam);
```

然后检查 m_bErrorFlag。如果是非零值，则表示发生了被零除或其他错误。错误发生后，用户必须单击 Clx 清除所有显示。所以如果 m_bErrorFlag 为非零值，则 OnCommand 只响应 Clx：

```
if (m_bErrorFlag && (nID != IDC_CLX)) {
    ::MessageBeep (MB_ICONASTERISK);
    return TRUE;
}
```

类似地，如果设定了 m_bFixPending 标志(表示按了 Fix 键后，计算器在等待用户按下数字键)，除了 0 到 9 以及 Clx 键，OnCommand 不响应其他任何键。其中 Clx 键取消改变小数位操作：

```
if (m_bFixPending &&
    ((nID < IDC_0) || (nID > IDC_9)) &&
    (nID != IDC_CLX)) {
    ::MessageBeep (MB_ICONASTERISK);
    return TRUE;
}
```

在两种情况中，都调用了::MessageBeep API 函数，有声音发出，提示用户按了一个无效按钮。只有 m_bErrorFlag 和 m_bFixPending 均为零时，才调用基类的 OnCommand 处理程序。

将测试这些标志的程序代码放在 OnCommand 处理程序中，从而避免在每个 ON_BN_CLICKED 处理程序中复制这段代码。

和 WM_COMMAND 消息有关的另一个有趣的问题是：DlgCalc 用一个命令处理程序处理 0 到 9 这十个按钮上的单击事件。通过手工编程将 ON_CONTROL_RANGE 语句添加到消息映射表中。该语句将来自其中某个按钮的 BN_CLICKED 通知导向 CDlgCalcDlg::OnDigit。

```
ON_CONTROL_RANGE (BN_CLICKED, IDC_0, IDC_9, OnDigit)
```

ON_CONTROL_RANGE 处理程序接收到一个 UINT 参数(标识发送通知的控件)，并不返回值。对于 DlgCalc，可替代 ON_CONTROL_RANGE 的就是 10 个独立的 ON_BN_CLICKED 宏和一个处理程序，该处理程序调用 CWnd::GetCurrentMessage 在消息的 wParam 中获取控件 ID。显然一个消息映射表项要比十个映射表项更节省内存，而且在可能保证向下兼容性的情况下，从消息参数中提取控件 ID 的工作最好留给 MFC 去做。

8.4　属性表

所有编程人员都很欣赏 Windows 提供的属性表。属性表是包含控件页的标签式对话框，用户可通过单击鼠标在其间切换。属性表在公用控件库中，所有 Windows 都提供了这个库。对使用 Windows API 编程来说，属性表显得很琐碎，但是因为有了主结构的支持，在 MFC 中属性表则变得相当好用。实际上，给 MFC 应用程序添加属性表和添加对话框并没有太大的区别。使用了属性表的 MFC 应用程序，如果在 Windows 95 或后来在 Windows NT 3.51 上运行，则要用操作系统固有的属性表实现。在其他平台上，则使用 MFC 自己的属性表实现。

属性表的功能都合理地封装在一对 MFC 类——CPropertySheet 和 CPropertyPage——中。CPropertySheet 代表属性表自身，是从 CWnd 派生出来的。CPropertyPage 代表属性表的页，是从CDialog派生出来的。两者都是在头文件 Afxdlgs.h 中定义的。和对话框一样，属性表可以是模式和无模式的。CPropertySheet::DoModal 创建模式属性表，而 CPropertySheet::Create 创建无模式属性表。

创建模式属性表的步骤如下：

1. 针对属性表的每一页创建一个对话框模板，定义页的内容和特性。将对话框标题设置成您希望在属性表页上方标签中显现的标题。
2. 针对属性表的每一页由 CPropertyPage 派生出一个类似对话框的类，其中包含通过 DDX 或 DDV 与页中控件相联系的公用数据成员。
3. 由 CPropertySheet 派生出一个属性表类。将该属性表类和第 2 步中得到的属性表页类实例化。利用 CPropertySheet::AddPage 将各页按期望中的显示顺序添加到属性表中。