第12章　特定的应用程序

本章的实例相同之处很少，但它们的创建过程一般较简单或者经常被创建，或是二者皆有。MFC已经准备了两个精心制作的编辑器的示例 (记事本(Notepad)和写字板(Wordpad))，但正由于这两个实例制作太细致，很难看出如果创建一个不加修饰的实例时会是多么的简单。资源管理器(Explorer)界面也是一个像SDI和MDI界面那样广泛使用的界面，但创建起来并不是很容易。本章的实例包括：

实例41　创建简单的文本编辑器。本例将创建一个十分完整的文本编辑器，但并不需要太多的工作。

实例42　生成简单的RTF编辑器。本例将创建一个具有操纵字体和图片功能的编辑器，类似于Word处理函数。

实例43　创建资源管理器界面。本例将创建一个类似资源管理器的应用程序，左边具有一个树形控件，右边则具有一个列表控件。

实例44　创建简单的ODBC数据库编辑器。本例将创建一个简单的数据库编辑器，它可以操作符合ODBC标准的数据库。

实例45　创建简单的DAO数据库编辑器。和实例44一样，但具有操作DAO数据库功能。

实例46　创建简单的向导，本例将用属性表生成一个简单的向导。

12.1　实例41：创建简单的文本编辑器

1. 目标

如图12-1所示，使用编辑控件创建一个简单的文本编辑器。

2. 策略

本例将通过AppWizard使用MFC的CEditView，创建一个简单的文本编辑器。还将提供利用 ClassWizard为现有应用程序创建文本编辑器的方法。

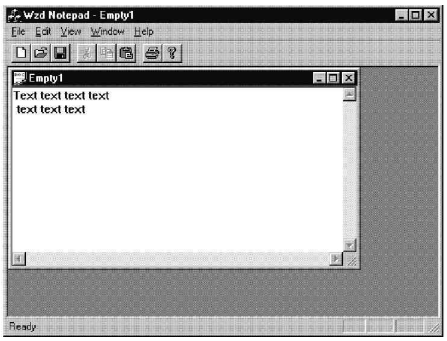

图12-1　简单文本编辑器

3. 步骤

1) 使用AppWizard生成文本编辑器

使用AppWizard，生成一个SDI或是MDI应用程序。

当仍处于 AppWizard中时，使用 Advanced Options，为该*.txt工程文件生成“文件扩展名(File Extension)”。

在AppWizard最后一步将会看到一个为这个工程生成类的详细清单。选择 CXxxView类，其中Xxx是自己定义的工程文件名。然后将注意到下面的基类组合框被启用，此时应当从该组合框中选择CEditView。

这样便可以了。AppWizard此时将创建合适的视类和文档类。视类支持剪切和粘贴功能，

文档类将被修改以装载和保存文本文件。普通的文件对话框将用于提醒用户使用前面指定的*.txt扩展名以装载和保存文件。

2) 使用ClassWizard生成一个文本编辑器视

在一个现有的应用程序中，使用ClassWizard生成一个由CEditView派生而来的新视类。

使用ClassWizard生成一个由CDocument派生而来的新文档类。

在这个新的文档类中，加入下面的代码到序列化函数中，以允许装载和保存文本文件:

```
void CWzdDoc::Serialize(CArchive& ar)
{
   POSITION pos = GetFirstViewPosition();
   ( ( CWzdEditView * )GetNextView( pos ) ) -> SerializeRaw( ar );
}
```

如果正在创建SDI应用程序，当新的文件被装载时，需要取消视类。为此需要添加下面的代码到文档类的OnNewDocument()函数中：

```
BOOL CWzdDoc::OnNewDocument()
{
   if ( !CDocument::OnNewDocument() )
      return FALSE;

   POSITION pos = GetFirstViewPosition();              <<< add
   ( ( CWzdEditView * )GetNextView( pos ) ) ->
      SetWindowText(NULL);                             <<< add

   return TRUE;
}
```

使用这两个新类就可以在应用程序类中创建新的文件模板类。

4. 注意

■ CEditView类是CView类和CEdit控件类的混合物。CEditView类同时也包含成员函数SerializeRaw()，该函数使装载和保存文件变得更容易。

■ 在MFC C++软件包中随之附带了几个优秀的文本编辑器实例。然而，因为它们都包含了一些用起来比较花哨的功能，结果代码看上去有些复杂，这就使得用户对使用 MFC的内置特征创建文本编辑器实际上非常简单这一事实产生误解。

5. 使用光盘时注意

当运行随书附送光盘上的工程时，将会注意到它具备了一个简单文本编辑器的所有功能。

12.2 实例42：生成简单的RTF编辑器

1. 目标

如图4-2所示，使用丰富编辑控件(rich edit control)生成一个简单的RTF编辑器。

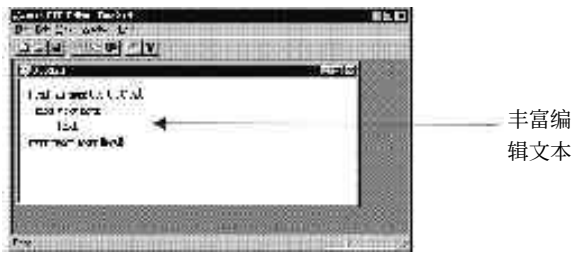

图12-2 简单RTF编辑器

2. 策略

本例使用MFC的CRichEditView和CRichEditDoc，用AppWizard制作一

个简单的多信息文本格式(RTF)的文本编辑器。同时还将看到如何使用 ClassWizard为一个现有的应用程序添加RTF文本编辑器。

3. 步骤

1) 使用AppWizard生成一个文本编辑器

使用AppWizard，创建一个SDI或是MDI应用程序。

当仍处于AppWizard时，使用 Advanced Options为该*.rtf工程文件生成“文件扩展名 (File Extension)”。

在AppWizard的最后一步，将会看到一个将为这个工程而生成的类详细清单。选择CXxxView类，其中Xxx是自己的工程文件名。然后将会注意到在下面基类的组合框被启动。此时应当从这个组合框中选择CRichEditView。

这样便可以了。此时 AppWizard将生成合适的视类和文档类。前者支持剪切和粘贴功能，后者将被修改以装载和保存 RTF文件。普通的文件对话框将用于提醒用户使用前面上面定义的*.rtf扩展名以装载和保存文件。

2) 使用ClassWizard生成一个RTF编辑器视

使用ClassWizard生成一个由CRichEditView派生而来的新的视类。

由于当前无法使用 ClassWizard创建一个由CRichEditDoc派生而来的新文档类，因此使用ClassWizard生成一个由 CDocument派生而来的新文档类作为替代。然后手工编辑生成的 .h和.cpp文件，用CRichEditDoc 代替所有在文件中出现的CDocument。

手工添加以下代码以重载文档类。对 CRichEditDoc而言，这是一个辅助函数，它将返回下一步将要创建的类，并允许文档中包含诸如 RTF文本的COM对象:

```
CRichEditCntrItem* CWzdDoc::CreateClientItem( REOBJECT* pReo ) const
{
   return new CWzdCntrItem( pReo, ( CWzdDoc* )this );
}
```

从资源中手工创建一个类，详细代码可以在本例结尾的程序清单—丰富编辑容器项目类(Rich Edit Container Item Class)中找到。

使用这两个新类就可以在应用程序类中建立新的文件模板类。

4. 注意

■ CRichEditView类是CView类和CRichEditCtrl控件类的混合物。CRichEditDoc类使装载和保存 RTF文件变得更容易。一个叫做 CRichEidtCntrItem的第三方 MFC类可帮助CRichEditDoc类装载和保存RTF文件。

■ 如果只打算使用CRichEditCtrl控件类，则必须在应用程序类的 OnInitInstance()函数中加入对AfxInitRichEdit()函数的调用。这将初始化应用程序以使用 RTF控件。CRichEditView类也同时为用户调用这个函数。

■ 嵌入到丰富编辑视(Rich Edit View)中的丰富文本控件同样具有内置功能，可以设置字体和段落格式，请参看有关CRichEditCtrl类的MFC文档以了解如何实现这些功能。

5. 使用光盘时注意

当运行随书附送光盘上的工程时，将会注意到它可以打开和显示 .rtf文件。

6. 程序清单 —— 丰富编辑容器项目类(Rich Edit Container Item Class)

```
// WzdCntrItem.h : interface of the CWzdCntrItem class
```

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

#if !defined WZDCNTRITEM
#define WZDCNTRITEM

class CWzdDoc;
class CWzdRichEditView;

class CWzdCntrItem : public CRichEditCntrItem
{
    DECLARE_SERIAL( CWzdCntrItem )

// Constructors
public:
    CWzdCntrItem( REOBJECT* pReo = NULL, CWzdDoc* pContainer = NULL );

// Attributes
public:
    CWzdDoc* GetDocument()
        { return (CWzdDoc*)COleClientItem::GetDocument(); }
    CWzdRichEditView* GetActiveView()
        { return (CWzdRichEditView*)COleClientItem::GetActiveView(); }

    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL(CWzdCntrItem)
public:
protected:
    // }}AFX_VIRTUAL

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

/////////////////////////////////////////////////////////////////////////////
#endif
// WzdCntrItem.cpp : implementation of the CWzdCntrItem class
//

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

#include "WzdDoc.h"
#include "WzdRichEditView.h"
#include "WzdCntrItem.h"

#ifdef _DEBUG
```

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

/////////////////////////////////////////////////////////////////////////////
// CWzdCntrItem implementation

IMPLEMENT_SERIAL( CWzdCntrItem, CRichEditCntrItem, 0 )

CWzdCntrItem::CWzdCntrItem( REOBJECT *pReo, CWzdDoc* pContainer )
    : CRichEditCntrItem( pReo, pContainer )
{
}

/////////////////////////////////////////////////////////////////////////////
// CWzdCntrItem diagnostics

#ifdef _DEBUG
void CWzdCntrItem::AssertValid() const
{
    CRichEditCntrItem::AssertValid();
}

void CWzdCntrItem::Dump( CDumpContext& dc ) const
{
    CRichEditCntrItem::Dump( dc );
}

#endif
```

12.3 实例43：创建资源管理器界面

1. 目标

创建一个界面类似于Windows资源管理器的应用程序，如图12-3所示。

2. 策略

Windows资源管理器是一个 SDI应用程序，它只有一个文档但是有两个视。第一个视是树形视，第二个是列表控件视。这两个视实际上都是在另一个名为分离窗口 (splitter window)的窗口中，该窗口填充了整个主窗口。 Visual C++6.0版本中的AppWizard允许自动选择和生成这个界面(在其第5步中选择),然而它不填充使这两个视同步所需的逻辑代码.

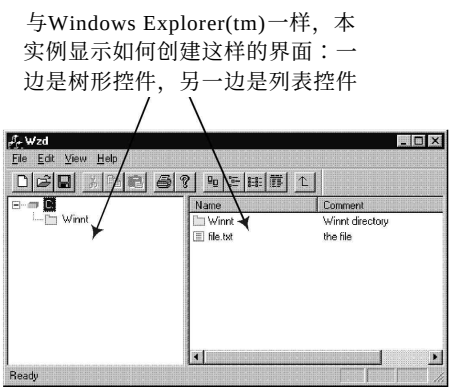

图12-3 资源管理器界面

因此，在本例中，对于没有使用 6.0版本的情况将首先手工创建这个界面。方法是修改CMainframe类，将树形视分解为两个部分，并将列表视加入到第二个部分。然后使用标准的CTreeView和CListView类调用来填充这些视。同时将同步这些视使树形视中的内容与列表视

中的内容相关联，反之亦然。为使每一级的子目录都能被显示，将大量使用递归调用。本例中将显示所有的文件目录，对任何与之相似的层次结构都可以使用这些方法显示。

3. 步骤

1) 创建应用程序

使用Application Wizard生成一个SDI应用程序。在最后一步将得到一个为工程文件生成的类的详细清单。选择 CXxxView类，其中 Xxx是自己的工程文件名。选择 Base class下的CTreeView，单击Finish以完成工程文件。

使用ClassWizard创建一个由CListView派生而来的新的视类。

在CMainFrame类中嵌入一个分离窗口类，如下所示：

```
private:
   CSplitterWnd m_wndSplitter;
```

然后，使用ClassWizard为OnCreateClient()函数添加并重载一个CMainFrame类。在其中首先创建一个具有两栏的可分离窗口：

```
BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT lpcs,
   CCreateContext* pContext )
{
   // create a split window with 1 row and 2 columns
   if ( !m_wndSplitter.CreateStatic( this, 1, 2 ) )
   {
      TRACE0( "Failed to CreateStaticSplitter\n" );
      return FALSE;
   }
```

接下来在OnCreateClient()中告诉分离窗口使用传递到这个函数的树形视作为第一栏中的视。

```
CRect rect;
GetClientRect( &rect );
if ( !m_wndSplitter.CreateView( 0, 0, pContext ->
   m_pNewViewClass, CSize( rect.Width()/4, 50 ), pContext ) )

{
   TRACE0( “Failed to create first pane\n” );
   return FALSE;
}
```

注意正在设置应用程序主窗口的宽度以及该栏的宽度。

接下来在OnCreateClient()中，告诉可分离窗口第二栏是用 ClassWizard生成的列表视:

```
if ( !m_wndSplitter.CreateView( 0, 1, RUNTIME_CLASS( CWzdListView ),
   CSize( 200,50), pContext ) )
{
   TRACE0( "Failed to create second pane\n" );
   return FALSE;
}
```

最后在OnCreateClient()中将告诉CMainFrame该列表视在初始时是被激活的视，也可以返回TRUE而不调用初始化的 CFrameWnd::OnCreateClient()，这将取消刚才所进行的全部工作。

```
SetActiveView( ( CView* )m_wndSplitter.GetPane( 0,1 ) );

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

2) 填写树形视类

使用ClassWizard在树形视类中加入一个重载的 OnInitialUpdate()。在其中将创建和装载位图，这些位图在树形控件中显示:

```
void CWzdTreeView::OnInitialUpdate()
{
    CTreeView::OnInitialUpdate();

    m_ImageList.Create( IDB_SMALL_BITMAP, 16, 1, RGB( 0,0,0 ) );
    GetTreeCtrl().SetImageList( &m_ImageList, TVSIL_NORMAL );

}
```

资源管理器类型界面的最基本的操作是单击树形视的某一项目时，将会打开树形视中相对应的文件。树形视中的这一操作在列表视中可以通过双击来实现。这里将使用文档类中的UpdateAllViews()来同步这两个视。因此这两个视必须重载 OnUpdate()函数，使用提供的lHint来确定其操作，如下面所示。

使用ClassWizard重载树形视类的OnUpdate()函数。然后检查是否这一修改只是修改了列表视，如果是则立即返回:

```
void CWzdTreeView::OnUpdate( CView* pSender, LPARAM lHint,
    CObject* pHint )
{
    if ( lHint&LIST_VIEW_ONLY )
    {
        return;
    }
```

接下来在这个OnUpdate()中，使用一个Case语句来确定如何去修改这个视。缺省的情况下只是使用一个辅助函数并用文档来填充该视，如下所示：

```
switch ( lHint )
{
    default:
        GetTreeCtrl().DeleteAllItems();
        AddBranch( TVI_ROOT, GetDocument() -> GetWzdList() );
        break;
```

该辅助函数AddBranch()可以在本例结尾的程序清单—树形视类中找到。它使用了许多的递归调用来填充该视，其中不仅包括名字，而且还包括在将来进一步标识列表项目的数据对象指针。其中的数据对象包含有关名字、目录 (FILE、DIRCTORY等)、注释、还有项目的对象标识号的信息。对象标识号在试图从位于其他视中的视中定位某一项目时显得非常重要，如下面所示。

如果视的这种修改是由用户在列表视中选择了新的项目引起的，在此便需要在树形视中选择相同的项目。为此首先在树形视中定位这个相同的项目。然后如下所示选择它：

```
case TREE_VIEW_SELECT:
```

```
    pInfo = GetDocument() -> GetSelection();
    if ( (hitem =
       FindTreeItem( GetTreeCtrl().GetChildItem( GetTreeCtrl().GetRootItem() ),
       pInfo -> m_nObjectID ) ) != NULL )
    {
       GetTreeCtrl().SelectItem( hitem );
    }
    break;
```

这里可以看到，通过使用列表视项目的对象标识号来定位树形视中与之相同的项目。

同样也可以处理这种情况，即用户希望沿树形层次结构向上(从该子目录到其上一层目录)。在这个例子中，用户通过一个工具栏按钮完成此操作。

```
       case TREE_VIEW_UP_LEVEL:
          hitem = GetTreeCtrl().GetSelectedItem();
          hitem = GetTreeCtrl().GetParentItem( hitem );
          GetTreeCtrl().SelectItem( hitem );
          break;

    }
```

为确定用户何时选择树形视中一个新的项目，可以使用 ClassWizard添加TVN_CHANGED控件通知消息处理函数，以通知文档类用户的选择已经改变，文档类然后使用UpdateAllView()函数去通知列表视需要进行修改。

```
void CWzdTreeView::OnSelchanged( NMHDR* pNMHDR, LRESULT* pResult )
{
    NM_TREEVIEW* pNMTreeView = ( NM_TREEVIEW* )pNMHDR;

    GetDocument() -> SaveSelection( ( CWzdInfo * )pNMTreeView ->
       itemNew.lParam,LIST_VIEW_ONLY );

    *pResult = 0;
}
```

参考本实例结尾的程序清单 —— 树形视类，查看树形视类的完整代码列表。

3) 填写列表视类

使用ClassWizard重载OnInitialUpdate()。在此应当装载一些大小不一的位图，以便在列表视中使用。在用户使用报表风格时还应当初始化列表视的各栏目以及对应的栏目名称：

```
void CWzdListView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    // attach image lists to list control
    m_ImageList.Create( IDB_NORMAL_BITMAP, 32, 1, RGB( 0,0,0 ) );
    GetListCtrl().SetImageList( &m_ImageList, LVSIL_NORMAL );
    m_ImageSmallList.Create( IDB_SMALL_BITMAP, 16, 1, RGB( 0,0,0 ) );
    GetListCtrl().SetImageList( &m_ImageSmallList, LVSIL_SMALL );

    // initialize list control
    CClientDC dc( this );
    TEXTMETRIC tm;
```

```
    dc.GetTextMetrics( &tm );
    GetListCtrl().ModifyStyle( 0,LVS_REPORT|LVS_SHOWSELALWAYS );
    GetListCtrl().SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE,0,
      LVS_EX_FULLROWSELECT );
    GetListCtrl().InsertColumn( 0,"Name",LVCFMT_LEFT,
      20*tm.tmAveCharWidth,0 );
    GetListCtrl().InsertColumn( 1,"Comment", LVCFMT_LEFT,
      30*tm.tmAveCharWidth,1 );
}
```

接下来，使用ClassWizard重载OnUpdate()函数。如果这只是一次树形视的修改则返回，并且再次使用Case语句来处理修改操作：

```
void CWzdListView::OnUpdate( CView* pSender, LPARAM lHint,
    CObject* pHint )
{
    if ( lHint&TREE_VIEW_ONLY )
    {
        return;
    }

    GetListCtrl().DeleteAllItems();

    switch ( lHint )
    {
```

列表视有四种风格：大图标、小图标、列表以及报表。在本例中，用户可以通过工具栏中的按钮选择其中的一种，同时文档类在此发送用户请求，并相应地修改该列表视中的控件：

```
        case ICON_LIST_VIEW:
            GetListCtrl().ModifyStyle( LVS_LIST|LVS_SMALLICON|LVS_REPORT,
                LVS_ICON );
            break;

        case LIST_LIST_VIEW:
            GetListCtrl().ModifyStyle( LVS_ICON|LVS_SMALLICON|LVS_REPORT,
                LVS_LIST );
            break;

        case SMALL_ICON_LIST_VIEW:
            GetListCtrl().ModifyStyle( LVS_LIST|LVS_ICON|LVS_REPORT,
                LVS_SMALLICON );
            break;

        case REPORT_LIST_VIEW:
            GetListCtrl().ModifyStyle( LVS_LIST|LVS_SMALLICON|LVS_ICON,
                LVS_REPORT );
            break;
    }
```

接下来在OnUpdate()中用文档数据填充列表视。再次使用辅助函数 AddLine()，它将再次被多次递归调用。在程序清单 —— 列表视类中可以发现其中有一个 AddLine()函数的列表。其

中首先列出子目录中的所有文件夹，然后列出其中所有的文件:

```
        // list all folders in selection
        POSITION pos;
        CWzdInfo *pSelInfo = GetDocument() -> GetSelection();
        for ( pos = pSelInfo -> m_list.GetHeadPosition(); pos; )
        {
            CWzdInfo *pInfo = pSelInfo -> m_list.GetNext( pos );
            if ( pInfo -> m_nCategory == CWzdInfo::FOLDER )
            {
                AddLine( pInfo );
            }
        }

        // list all files in selection
        for ( pos = pSelInfo -> m_list.GetHeadPosition(); pos; )
        {
            CWzdInfo *pInfo = pSelInfo -> m_list.GetNext(pos);
            if (pInfo -> m_nCategory == CWzdInfo::FILE)
            {
                AddLine(pInfo);
            }
        }
    }
```

如上面所提到的，用户可以用双击列表视项目的方法来改变树形视的相应选择项目。使用ClassWizard添加WM_LBUTTONDBLCLK消息处理函数到列表视类中。在其中查找用户所选择的项目，并且在发现时通知文档类。文档类将把这个消息传递给树形视类:

```
void CWzdListView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    int ndx;
    if ( GetListCtrl().GetSelectedCount() == 1 &&
        ( ndx = GetListCtrl().GetNextItem( -1, LVIS_SELECTED ) ) != -1 )
    {
        GetDocument() ->
            SaveSelection( (CWzdInfo *)GetListCtrl().GetItemData(ndx),
            TREE_VIEW_SELECT);
    }

    CListView::OnLButtonDblClk( nFlags, point );
}
```

这样就完成了列表视类的编写。关于这个类的完整的代码列表，请参考下面的程序清单—列表视类。

文档类负责维护数据项，同时还负责同步两个视类 (在本例中使用的数据项可以在下面的程序清单—数据类中找到)。改变列表视风格的工具栏按钮和使树形视向上一层的按钮都在文档类中处理。如前面所述， UpdateAllViews()用来与视通信。

4) 填充文档类

使用工具栏编辑器(Toolbar Editor)为列表视中的四种风格：大图标、小图标、列表和报表

添加按钮。

使用ClassWizard为文档类中的这些新工具栏按钮加入命令处理函数。

在这些命令处理函数中，使用 UpdateAllViews()函数通知列表视哪种风格将被使用，如下所示：

```
void CWzdDoc::OnIconMode()
{
   UpdateAllViews( NULL,ICON_LIST_VIEW );
}

void CWzdDoc::OnListMode()
{
   UpdateAllViews( NULL,LIST_LIST_VIEW );
}

void CWzdDoc::OnReportMode()
{
   UpdateAllViews( NULL,REPORT_LIST_VIEW );
}

void CWzdDoc::OnSmalliconMode()
{
   UpdateAllViews( NULL,SMALL_ICON_LIST_VIEW );
}
```

在此使用的命令(例如：SMALL_ICON_VIEW)是在文档类包含文件中定义的。

使用工具栏编辑器和 ClassWizard添加一个按钮控件，它将允许用户转向上一层目录，如下所示填充有关按钮的代码：

```
void CWzdDoc::OnUplevel()
{
   UpdateAllViews( NULL,TREE_VIEW_UP_LEVEL );
}
```

使用文本编辑器在文档类中加入一个新的函数，该函数可以被两个视中的任一个调用，这样便可以修改当前被选择的数据项。该函数应当如下所示：

```
void CWzdDoc::SaveSelection( CWzdInfo *pSelectionInfo,int nMode )
{
   m_pSelectionInfo = pSelectionInfo;
   UpdateAllViews( NULL,nMode );
}
```

这样便可以了。单击任何一个视中的项目都将导致在另一个视中自动选择相同的项目，以此来产生响应。

4. 注意

■ Windows资源管理器是一个SDI应用程序，当然可以把它创建为一个 MDI应用程序—但它不能通过AppWizard自动创建。按照本实例开始，使用 AppWizard来创建一个MDI应用程序而不是 SDI应用程序。然后不重载 CMainFrame的OnCreateClient()函数，而是重载CChildFrame的OnCreateClient()函数来创建一个分离窗口，其他所有操作基本上都是一样的。也许用户考虑在分离窗口中创建四个视而不是两个。这样便将显示两组树形视和列表视。但

这需要在文档类中进行说明，因为所有这四个视将使用同一个文档。

■ 为将一个实际的目录信息填充到视中，可以使用 Windows API调用::FindFirstFile()和::FindNextFile()。

■ 如上面所述，资源管理器类型的界面除了维护文件目录以外，还可以用于其他许多应用程序。例如用于在CAD系统中访问窗体，其中的窗体作为文件排列在被称作模型 (model)的目录下面。另一个使用是维护网络，这里每一个网络节点是“目录”，而内部的网卡是“文件”。

5. 使用光盘时注意

当执行随书附带光盘上的工程时，将注意到它和 Windows资源管理器在外观上和感觉上都非常相似，尽管功能上有些差异。

6. 程序清单 —— 数据类

```
#ifndef WZDINFO_H
#define WZDINFO_H

#include "afxtempl.h"
#include "WzdInfo.h"

class CWzdInfo : public CObject
{
public:
   CWzdInfo( CString sName,CString sComment,int nCategory,int nObjectID );
   ~CWzdInfo();

   enum {
   DEVICE,
   FOLDER,
   FILE
   };

   // misc info
   CString m_sName;
   CString m_sComment;
   int m_nCategory;
   int m_nObjectID;

   CList<CWzdInfo*,CWzdInfo*> m_list;

};
#endif
// WzdInfo.cpp : implementation of the CWzdInfo class
//

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

/////////////////////////////////////////////////////////////////////////////
// CWzdInfo
```

```
CWzdInfo::CWzdInfo( CString sName,CString sComment,int nCategory,
    int nObjectID ) :
    m_sName( sName ), m_sComment( sComment ),m_nCategory( nCategory ),
    m_nObjectID( nObjectID )
{
}

CWzdInfo::~CWzdInfo()
{
    while (!m_list.IsEmpty())
    {
        delete m_list.RemoveHead();
    }
}
```

7. 程序清单 —— 树形视类

```
// WzdTreeView.h : interface of the CWzdTreeView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined WZDTREEVIEW_H
#define WZDTREEVIEW_H

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

#include <afxcview.h>

class CWzdTreeView : public CTreeView
{
protected:    // create from serialization only
    CWzdTreeView();
    DECLARE_DYNCREATE( CWzdTreeView )

// Attributes
public:
    CWzdDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdTreeView )
public:
    virtual void OnDraw( CDC* pDC );    // overridden to draw this view
    virtual BOOL PreCreateWindow( CREATESTRUCT& cs );
    virtual void OnInitialUpdate();
protected:
```

```
    virtual BOOL OnPreparePrinting( CPrintInfo* pInfo );
    virtual void OnBeginPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnEndPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint );
    // }}AFX_VIRTUAL

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

protected:

// Generated message map functions
protected:
    // {{AFX_MSG( CWzdTreeView )
    afx_msg void OnSelchanged( NMHDR* pNMHDR, LRESULT* pResult );
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CImageList m_ImageList;
    void AddBranch( HTREEITEM hTreeItem, CList<CWzdInfo*,CWzdInfo*> *pList );
    HTREEITEM AddLeaf( HTREEITEM hTreeItem, CWzdInfo *pInfo );
    HTREEITEM FindTreeItem( HTREEITEM hTreeItem, long nObjectID );
};

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

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

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

#endif
// WzdTreeView.cpp : implementation of the CWzdTreeView class
//

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

#include "WzdDoc.h"
#include "WzdTreeView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

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

/////////////////////////////////////////////////////////////////////////////
// CWzdTreeView

IMPLEMENT_DYNCREATE( CWzdTreeView, CTreeView )

BEGIN_MESSAGE_MAP( CWzdTreeView, CTreeView )
    // {{AFX_MSG_MAP( CWzdTreeView )
    ON_NOTIFY_REFLECT( TVN_SELCHANGED, OnSelchanged )
    // }}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND( ID_FILE_PRINT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_DIRECT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview )
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdTreeView construction/destruction

CWzdTreeView::CWzdTreeView()
{
    // set the CTreeCtrl attributes
    m_dwDefaultStyle = WS_CHILD | WS_VISIBLE | WS_BORDER | TVS_HASLINES |
        TVS_SHOWSELALWAYS | TVS_LINESATROOT | TVS_HASBUTTONS;
}

CWzdTreeView::~CWzdTreeView()
{
}

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

    return CTreeView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CWzdTreeView drawing

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

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

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

```
// CWzdTreeView printing

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

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

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

/////////////////////////////////////////////////////////////////////////////
// CWzdTreeView diagnostics

#ifdef _DEBUG
void CWzdTreeView::AssertValid() const
{
    CView::AssertValid();
}

void CWzdTreeView::Dump( CDumpContext& dc ) const
{
    CView::Dump( dc );
}

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

/////////////////////////////////////////////////////////////////////////////
// CWzdTreeView message handlers

void CWzdTreeView::OnInitialUpdate()
{
    CTreeView::OnInitialUpdate();

    // set icons
    m_ImageList.Create( IDB_SMALL_BITMAP, 16, 1, RGB( 0,0,0 ) );
    GetTreeCtrl().SetImageList( &m_ImageList, TVSIL_NORMAL );

}
```

```
void CWzdTreeView::OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint )
{
    if ( lHint&LIST_VIEW_ONLY )
    {
        return;
    }

    HTREEITEM hitem;
    CWzdInfo *pInfo;
    switch ( lHint )
    {
        case TREE_VIEW_SELECT:
            pInfo = GetDocument() -> GetSelection();
            if ( ( hitem = FindTreeItem( GetTreeCtrl().GetChildItem(
                GetTreeCtrl().GetRootItem() ), pInfo -> m_nObjectID ) ) != NULL )
            {
                GetTreeCtrl().SelectItem(hitem);
            }
            break;

        case TREE_VIEW_UP_LEVEL:
            hitem = GetTreeCtrl().GetSelectedItem();
            hitem = GetTreeCtrl().GetParentItem( hitem );
            GetTreeCtrl().SelectItem(hitem);
            break;

        default:
            GetTreeCtrl().DeleteAllItems();
            AddBranch(TVI_ROOT, GetDocument() -> GetWzdList());
            break;
    }
}

// recurse until all CWzdInfo items are in a leaf
void CWzdTreeView::AddBranch( HTREEITEM hTreeItem,
    CList<CWzdInfo*,CWzdInfo*> *pList )
{
    for ( POSITION pos = pList -> GetHeadPosition(); pos; )
    {
        CWzdInfo *pInfo = pList -> GetNext( pos );
        if ( pInfo -> m_nCategory != CWzdInfo::FILE )
        {
            HTREEITEM hTreeItemx = AddLeaf( hTreeItem, pInfo );
            if ( pList -> GetCount() )
            {
                AddBranch( hTreeItemx, &pInfo -> m_list );
            }
        }
    }
}
```

```
// add leaf to tree
HTREEITEM CWzdTreeView::AddLeaf( HTREEITEM hTreeItem, CWzdInfo *pInfo )
{
   TV_INSERTSTRUCT tvstruct;
   tvstruct.hParent = hTreeItem;
   tvstruct.hInsertAfter = TVI_LAST;
   tvstruct.item.iImage = tvstruct.item.iSelectedImage = pInfo -> m_nObjectID;
   tvstruct.item.pszText = ( char * )LPCTSTR( pInfo -> m_sName );
   tvstruct.item.mask =
      TVIF_IMAGE |TVIF_SELECTEDIMAGE | TVIF_PARAM  | TVIF_TEXT;
   tvstruct.item.lParam = ( DWORD )pInfo;
   return GetTreeCtrl().InsertItem( &tvstruct );
}

// recurse until item found or items expended
HTREEITEM CWzdTreeView::FindTreeItem( HTREEITEM hTreeItem, long nObjectID )
{
   while ( hTreeItem != NULL )
   {
      CWzdInfo *pInfo = ( CWzdInfo * )GetTreeCtrl().GetItemData( hTreeItem );
      if ( pInfo -> m_nObjectID == nObjectID )
      {
         return hTreeItem;
      }
      else if ( GetTreeCtrl().ItemHasChildren( hTreeItem ) )
      {
         HTREEITEM hTreeItemx =
            FindTreeItem( GetTreeCtrl().GetChildItem( hTreeItem ),
            nObjectID );
         if ( hTreeItemx )
         {
            return hTreeItemx;
         }
      }
      else
      {
         hTreeItem = GetTreeCtrl().GetNextItem( hTreeItem, TVGN_NEXT );
      }
   }
   return NULL;
}

// new selection, change listview!
void CWzdTreeView::OnSelchanged( NMHDR* pNMHDR, LRESULT* pResult )
{
   NM_TREEVIEW* pNMTreeView = ( NM_TREEVIEW* )pNMHDR;

   GetDocument() -> SaveSelection( ( CWzdInfo * )pNMTreeView ->
      itemNew.lParam,LIST_VIEW_ONLY );

   *pResult = 0;
}
```

8. 程序清单 —— 列表视

```
// WzdListView.h : interface of the CWzdListView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined WZDLISTVIEW_H
#define WZDLISTVIEW_H

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

#include <afxcview.h>

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

// Attributes
public:
    CWzdDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    // {{AFX_VIRTUAL( CWzdListView )
public:
    virtual void OnDraw( CDC* pDC );    // overridden to draw this view
    virtual BOOL PreCreateWindow( CREATESTRUCT& cs );
    virtual void OnInitialUpdate();
protected:
    virtual BOOL OnPreparePrinting( CPrintInfo* pInfo );
    virtual void OnBeginPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnEndPrinting( CDC* pDC, CPrintInfo* pInfo );
    virtual void OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint );
    // }}AFX_VIRTUAL

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

protected:

// Generated message map functions
protected:
```

```
    // {{AFX_MSG( CWzdListView )
    afx_msg void OnLButtonDblClk( UINT nFlags, CPoint point );
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CImageList m_ImageList;
    CImageList m_ImageSmallList;
    void AddLine( CWzdInfo *pInfo );
};

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

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

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

#endif
    // !defined( AFX_WzdListView_H__CA9038F0_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_ )
// WzdListView.cpp : implementation of the CWzdListView class
//

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

#include "WzdDoc.h"
#include "WzdListView.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdListView

IMPLEMENT_DYNCREATE( CWzdListView, CListView )

BEGIN_MESSAGE_MAP( CWzdListView, CListView )
    // {{AFX_MSG_MAP( CWzdListView )
    ON_WM_LBUTTONDBLCLK()
    // }}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND( ID_FILE_PRINT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_DIRECT, CView::OnFilePrint )
    ON_COMMAND( ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview )
```

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWzdListView construction/destruction

CWzdListView::CWzdListView()
{
    // TODO: add construction code here

}

CWzdListView::~CWzdListView()
{
}

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

    return CListView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CWzdListView drawing

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

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

/////////////////////////////////////////////////////////////////////////////
// CWzdListView printing

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

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

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

```
}

/////////////////////////////////////////////////////////////////////////////
// CWzdListView diagnostics

#ifdef _DEBUG
void CWzdListView::AssertValid() const
{
    CView::AssertValid();
}

void CWzdListView::Dump( CDumpContext& dc ) const
{
    CView::Dump( dc );
}

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

/////////////////////////////////////////////////////////////////////////////
// CWzdListView message handlers

void CWzdListView::OnInitialUpdate()
{
    CListView::OnInitialUpdate();

    // attach image lists to list control
    m_ImageList.Create( IDB_NORMAL_BITMAP, 32, 1, RGB( 0,0,0 ) );
    GetListCtrl().SetImageList( &m_ImageList, LVSIL_NORMAL );
    m_ImageSmallList.Create( IDB_SMALL_BITMAP, 16, 1, RGB( 0,0,0 ) );
    GetListCtrl().SetImageList( &m_ImageSmallList, LVSIL_SMALL );

    // initialize list control
    CClientDC dc( this );
    TEXTMETRIC tm;
    dc.GetTextMetrics( &tm );
    GetListCtrl().ModifyStyle( 0,LVS_REPORT|LVS_SHOWSELALWAYS );
    GetListCtrl().SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE,0,
        LVS_EX_FULLROWSELECT );
    GetListCtrl().InsertColumn( 0,"Name", LVCFMT_LEFT,20*tm.tmAveCharWidth,0 );
    GetListCtrl().InsertColumn( 1,"Comment", LVCFMT_LEFT,
        30*tm.tmAveCharWidth,1 );

}

void CWzdListView::OnUpdate( CView* pSender, LPARAM lHint, CObject* pHint )
{
```

```
if ( lHint&TREE_VIEW_ONLY )
{
    return;
}

GetListCtrl().DeleteAllItems();

switch (lHint)
{
    case ICON_LIST_VIEW:
        GetListCtrl().ModifyStyle(LVS_LIST|LVS_SMALLICON|LVS_REPORT,
            LVS_ICON);
        break;

    case LIST_LIST_VIEW:
        GetListCtrl().ModifyStyle(LVS_ICON|LVS_SMALLICON|LVS_REPORT,
            LVS_LIST);
        break;

    case SMALL_ICON_LIST_VIEW:
        GetListCtrl().ModifyStyle(LVS_LIST|LVS_ICON|LVS_REPORT,
            LVS_SMALLICON);
        break;

    case REPORT_LIST_VIEW:
        GetListCtrl().ModifyStyle(LVS_LIST|LVS_SMALLICON|LVS_ICON,
            LVS_REPORT);
        break;
}

// list all folders in selection
POSITION pos;
CWzdInfo *pSelInfo = GetDocument() -> GetSelection();
for ( pos = pSelInfo -> m_list.GetHeadPosition(); pos; )
{
    CWzdInfo *pInfo = pSelInfo -> m_list.GetNext( pos );
    if ( pInfo -> m_nCategory == CWzdInfo::FOLDER )
    {
        AddLine( pInfo );
    }
}

// list all files in selection
for ( pos = pSelInfo -> m_list.GetHeadPosition(); pos; )
{
    CWzdInfo *pInfo = pSelInfo -> m_list.GetNext( pos );
    if ( pInfo -> m_nCategory == CWzdInfo::FILE )
    {
        AddLine( pInfo );
    }
}
```

```
}

void CWzdListView::AddLine( CWzdInfo *pInfo )
{
   LV_ITEM item;
   item.mask = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM;
   item.iItem = GetListCtrl().GetItemCount();
   item.iSubItem = 0;
   item.pszText = ( char * )LPCTSTR( pInfo -> m_sName );
   item.iImage = pInfo -> m_nCategory;
   item.lParam = ( DWORD )pInfo;
   int ndx = GetListCtrl().InsertItem( &item );

   // if report style add comment
   if ( GetListCtrl().GetStyle() & LVS_REPORT )
   {
      GetListCtrl().SetItemText( ndx, 1, pInfo -> m_sComment );
   }
}

void CWzdListView::OnLButtonDblClk( UINT nFlags, CPoint point )
{
   int ndx;
   if ( GetListCtrl().GetSelectedCount() == 1 &&
      ( ndx = GetListCtrl().GetNextItem( -1, LVIS_SELECTED ) ) != -1 )
   {
      GetDocument() ->
         SaveSelection( ( CWzdInfo * )GetListCtrl().GetItemData( ndx ),
         TREE_VIEW_SELECT );
   }

   CListView::OnLButtonDblClk( nFlags, point );
}
```

12.4 实例44：创建简单的ODBC数据库编辑器

1. 目标

创建一个简单的ODBC数据库编辑器，它允许浏览遵循 ODBC格式的整个数据库并编辑其

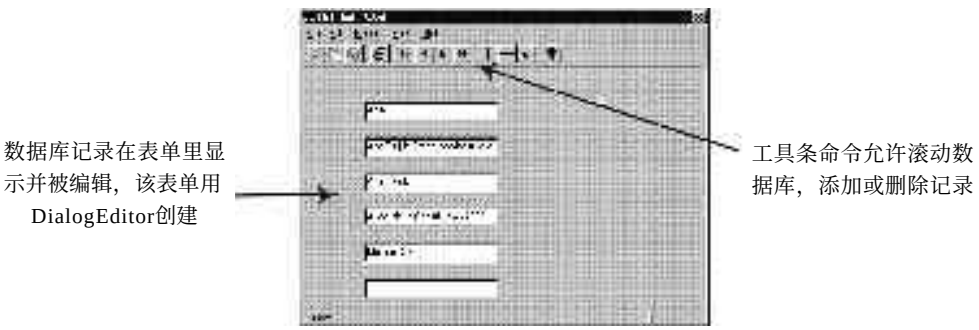

图12-4 简单ODBC编辑器

记录，如图12-4所示。

2. 策略

本例中的大多数工作都将由AppWizard、ClassWizard和对话框编辑器来完成。AppWizard将使用CRecordView类创建一个SDI应用程序。该视类由CFormView派生而来并为其视使用一个对话框。然后将使用对话框编辑器为这个视加入字段，ClassWizard将这些字段赋值到数据库中的列。视类和其他AppWizard由添加的命令提供了所有必需的功能以浏览和编辑这些数据库的字段。但是，还将添加3个附加的命令用于添加、删除或清空数据库记录。

3. 步骤

1) 用AppWizard创建一个ODBC数据库编辑器

使用AppWizard创建一个SDI应用程序。在第2步为应用程序加入Database view without file support，然后单击Data Source按钮，选择ODBC。使用下拉列表框，在其中选择该应用程序将要编辑的数据库。如果AppWizard成功地打开这个数据库，它将提醒用户该应用程序将要编辑数据库中的哪一个表。对此为应用程序选择缺省项即可。

注意 如果选择了Database view with file support，AppWizard将允许访问数据库并在同一应用程序中串行化文件文档。该选项还允许用户使用“数据库视(database view)”选项创建MDI应用程序。

AppWizard将为应用程序创建两个对话框模板，其中一个用于视 (另一个是普遍存在的“关于(About)”对话框模板)。使用对话框编辑器将字段加入到这些将在数据库表中代表列的模板中。

使用ClassWizard并选择视类。单击Member Variables标签，会发现刚才加入到这个视对话框中的字段的控件标识号在此被列出。选择第一个控件标识号，然后单击 Add Variable按钮，打开Add Member Variable对话框。在该对话框的Member variable name组合框中，单击它的下拉按钮以显示在这个数据库表中所有列的列表。选择其中的一列便将这个存放记录的列绑定到该控件标识号所代表的控件字段。对每一个控件字段重复此过程，为它指定存放记录的列。

为使视的大小与对话框大小相匹配，使用 ClassWizard重载视类的OnInitialUpdate()函数，使其如下所示：

```
void CWzdView::OnInitialUpdate()
{
   m_pSet = &GetDocument() -> m_wzdSet;
   CRecordView::OnInitialUpdate();

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

   // get rid of those pesky scroll bars
   SetScrollSizes( MM_TEXT, CSize( 20,20 ) );
}
```

2) 为该编辑器加入一个Add命令

使用菜单编辑器和工具栏编辑器，为菜单和工具栏添加 Add命令。

使用ClassWizard，在视类中对这个命令进行如下处理：

```
void CWzdView::OnRecordAdd()
```

```
{
    m_pSet -> AddNew();
    UpdateData( FALSE );
}
```

注意视类中的m_pSet成员变量是一个指针，它指向这个编辑器所存取的记录集。

3) 为该编辑器加入一个Clear命令

使用菜单编辑器和工具栏编辑器，为菜单和工具栏添加 Clear命令。

使用ClassWizard，在视类中对这个命令进行如下处理：

```
void CWzdView::OnRecordCanceledit()
{
    m_pSet -> CancelUpdate();
    m_pSet -> Move( 0 );
    UpdateData( FALSE );
}
```

4) 为该编辑器加入一个Delete命令

使用菜单编辑器和工具栏编辑器，为菜单和工具栏添加 Delete命令。

使用ClassWizard，在视类中对这个命令进行如下处理：

```
void CWzdView::OnRecordDelete()
{
    try
    {
        m_pSet -> Delete();
    }

    catch( CDBException* e )
    {
        AfxMessageBox( e -> m_strError );
        e -> Delete();
    }

    // Move off the record we just deleted
    m_pSet -> MoveNext();

    // If we moved off the end of file, move back to last record
    if ( m_pSet -> IsEOF() )
        m_pSet -> MoveLast();

    // If the recordset is now empty, clear the record we just deleted
    if ( m_pSet -> IsBOF() )
        m_pSet -> SetFieldNull( NULL );
        UpdateData( FALSE );
}
```

4. 注意

■ 为使用编辑器编辑一个字段，首先需要对相应的字段进行修改，然后从该记录进行向上或向下的滚动。在下一个记录显示之前，上一个记录已经被修改。 Clear命令将取消对当前记录所进行的任何修改，这样它代表的记录便没有改变。

■ 使用AppWizard和ClassWizard提供的所有自动功能的代价是所得的最终结果在某种程度上很不灵活。如果不选择 Database View with file support，则将不得不创建一个 SDI应用程序，并且不得不使用窗体视而不是其他类型的视。如果为数据库支持选择 Header files only则将失去所有这些自动功能，但应用程序将会变得更加灵活。

5. 使用光盘时注意

当运行随书附送光盘上的工程时，将注意到可以浏览一个 ODBC数据库。

12.5 实例45：创建简单的DAO数据库编辑器

1. 目标

创建一个简单的DAO数据库编辑器，允许浏览整个 DAO数据库并编辑其记录，如实例 44所示。

2. 策略

本例中的策略与上一个实例相同。仅有的不同之处是在处理 Add、Clear和Delete三个编辑器命令时会有差异。对于DAO数据库，对这三个命令的处理与通常差别很大。

3. 步骤

1) 创建一个DAO数据库编辑器

按上一个实例进行相同的操作以创建应用程序，并在视中加入字段—除了一点不同：定义的是DAO数据库文件而不是ODBC数据库。

2) 为该编辑器加入一个Add命令

使用菜单编辑器和工具栏编辑器，为菜单和工具栏添加 Add命令。

使用ClassWizard，在视类中对这个命令进行如下处理：

```
void CWzdView::OnRecordAdd()
{
   m_bAddingRecord = TRUE;   // keep track of what mode we're in
   m_pSet -> AddNew();
   UpdateData( FALSE );
}
```

注意视类中的m_pSet成员函数是一个指针，它指向这个编辑器所存取的记录集。

在创建Clear命令之前，需要在视类中添加一个重载函数以帮助用户确定是否仍然处于记录添加模式中。当在 ODBC数据库编辑器中清空一个记录时，可以使用 CancelUpdate()取消所进行的修改。但是在添加新记录的中途，对 DAO数据库试图进行同样的操作时， MFC DAO类将弹出一个错误信息 (exception)。因此，在Add命令中设置一个标志 (flag)以指示当前处于记录添加模式，然后在视类中的 OnMove()函数中清除该标志，因为 OnMove()函数将退出当前的记录并保存用户所进行的修改。

使用 ClassWizard重载视类的 OnMove()函数，其中应当清除在 Add命令中设置的m_bAddingRecord标志:

```
BOOL CWzdView::OnMove( UINT nIDMoveCommand )
{
   m_bAddingRecord = FALSE;

   return CDaoRecordView::OnMove( nIDMoveCommand );
}
```

3) 为该编辑器加入一个Clear命令

使用菜单编辑器和工具栏编辑器，为菜单和工具栏添加 Clear命令。

使用ClassWizard，在视类中对这个命令进行如下处理：

```
void CWzdView::OnRecordCanceledit()
{
    if ( m_bAddingRecord )
        m_pSet -> CancelUpdate();
    m_pSet -> Move( 0 );
    m_bAddingRecord = FALSE;
    UpdateData( FALSE );

}
```

4) 为该编辑器加入一个Delete命令

使用菜单编辑器和工具栏编辑器，为菜单和工具栏添加 Delete命令。

使用ClassWizard，在视类中对这个命令进行如下处理：

```
void CWzdView::OnRecordDelete()
{
    try
    {
        m_pSet -> Delete();
    }

    catch( CDaoException* e )
    {
        AfxMessageBox( e -> m_pErrorInfo -> m_strDescription );
        e -> Delete();
    }

    // Move off the record we just deleted
    m_pSet -> MoveNext();

    // If we moved off the end of file, move back to last record
    if ( m_pSet -> IsEOF() )
        m_pSet -> MoveLast();

    // If the recordset is now empty, clear the record we just deleted
    if ( m_pSet -> IsBOF() )
        m_pSet -> SetFieldNull( NULL );
    UpdateData( FALSE );
}
```

4. 注意

■ 为使用编辑器编辑一个字段，首先需要对相应的字段进行修改，然后从该记录进行向上或向下的滚动。当下一个记录显示之前，上一个记录已经被修改。 Clear命令将取消对当前记录所进行的任何修改，这样它代表的记录便没有改变。

■ 如果希望从一个非窗体 (nonform) 视应用程序直接访问 DAO数据库，那么在使用AppWizard创建应用程序的时候，应当指定Header files only。

5. 使用光盘时注意

当执行随书附送光盘上的工程时，将注意到可以浏览一个 DAO数据库。

12.6 实例46：创建简单的向导

1. 目标

如图12-5所示，使用Windows中大多数向导的规范创建一个简单的向导。

2. 策略

本例将使用 CPropertySheet中的向导模式来创建向导。此外，虽然向导可以是另一个应用程序的一部分，但是本例将使向导成为一个独立的应用程序。为此将创建一个对话框应用程序，然后用属性表来代替有模式对话框。

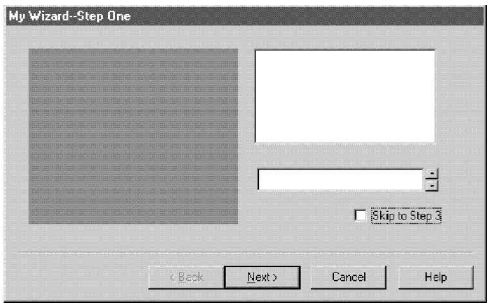

图12-5 简单的向导

3. 步骤

1) 创建工程

使用AppWizard来创建一个对话框应用程序，然后删除由它生成的对话框模板和类文件。

2) 创建向导页面

使用对话框编辑器，在向导中创建页面。除了用户能连续滚动它们以外，向导页面和属性页面是一样的。用户赋予每个对话框模板什么属性并不重要— CProperty类将重载它。每一个对话框模板的标题将变成在向导中看到的页标题。按照惯例，应当在每个标题后面附加“第一步”、“第二步”等字样。

使用ClassWizard为每一个对话框页面创建一个属性页类，该类从 CPropertyPage派生。在每一个类中可以使用 ClassWizard重载四个成员函数： CPropertyPage::OnsetActive()、OnWizardBack()、OnWizardNext()和OnWizardFinish()。

使用ClassWizard重载每一个属性页的OnSetActive()成员函数。在其中设置哪一个向导按钮对于该页是可见的或是可用的，如下所示：

```
BOOL CPage1::OnSetActive()
{
   ( ( CPropertySheet* )GetOwner() ) -> SetWizardButtons(
      PSWIZB_NEXT |                    // the “Next” button is visible
      PSWIZB_BACK |                    // the “Back” button is visible
      PSWIZB_FINISH |                  // the “Finish” button is visible
      PSWIZB_DISABLEDFINISH );         // disabled “Finish” button
                                       //     is visible

   return CPropertyPage::OnSetActive();
}
```

Back、Next和Finish按钮由CPropertySheet类自动处理。如果需要进行控制，则使用ClassWizard重载OnWizardBack()、OnWizardNext()和OnWizardFinish()函数。在每一个函数中可以调用UpdateData(TRUE)函数来检索该页的任何变化。 OnWizardBack()或OnWizardNext ()

函数的返回值决定了向导下一次将显示哪一页。返回缺省值为零则将显示连续的上一页或下一页。然而如果返回的是向导中其他属性页的对话框模板的标识号，向导接下来将显示该页。

通过重载OnWizardFinish()而后选择是否调用基类的CPropertyPage :: OnWizardFinish()函数可以有条件地终止向导。若不调用基类函数将允许向导继续进行。

在下面的例子中，如果用户单击Page 3 Next复选框，则接下来将显示IDD_PAGE3页:

```
LRESULT CPage1::OnWizardNext()
{
   UpdateData( TRUE );
   if ( m_bPage3Check )
   {
      return IDD_PAGE3;
   }
   else
   {
      return CPropertyPage::OnWizardNext();
   }
}
```

注意 CPropertyPage::OnWizardNext()和CPropertyPage::OnWizardBack()函数都只返回零(0)，这将使向导分别转向上一页或下一页。

3) 创建向导

在应用程序类中嵌入向导的属性页类：

```
CPageOne m_Page1;
CPageTwo m_Page2;
:   :   :
```

在应用程序类的InitInstance()函数中，在堆栈上创建CPropertySheet类的一个示例，然后按照希望显示给用户的次序，依次将属性页加入其中。此时还可以初始化属性页类的成员变量，然后设置一个全能的向导风格并调用CPropertySheet::DoModal()函数：

```
CWzdSheet wzd( " " );
m_pMainWnd = &wzd;
AddPage( &m_Page1 );
AddPage( &m_Page2 );
AddPage( &m_Page3 );
SetWizardMode();
if ( wzd.DoModal() == ID_WIZFINISH )
{
   // process results
}
```

如果DoModal()函数返回ID_WIZFINISH值，既可以在在最后一页的 OnWizardFinish()函数中处理向导结果，也可以直接在应用程序类的 DoModal()函数中处理向导的结果，后者可以很容易地访问所有页的成员变量，然而此时向导已关闭且不能返回。

4. 注意

■ 属性表的向导模式目前除在标题栏中显示标题之外，不能显示其他任何内容，换句话

说，就是没有关闭按钮和最小化按钮。如果已经在标题栏中加入了这些东西，请参照实例 27将向导嵌入到对话框应用程序的模板中。为去掉向导的标题栏，还可能需要从 CPorpertySheet中派生一个类并重载WM_NCPAINT消息以阻止它到达属性表。

■ 另外一种为向导的标题栏加入关闭按钮的方法是自己绘制一个关闭按钮，这可以参考实例29。但此处不必按照该例所介绍的方式来创建自己的按钮位图，而是使用CDC::DrawFrameControl()函数并指定合适的位图类型和状态参数来绘制。

■ 为什么属性表的向导模式不能在它的标题栏中包含任何控件？毕竟 AppWizard的标题栏中具有一个关闭按钮和一个上下文帮助按钮。说出来原因有点令人吃惊， AppWizard不能使用属性表的向导模式。事实上 AppWizard根本不使用属性表。AppWizard实际上是一个具有单个主对话框模板的对话框应用程序。因此，它具有所有对话框应用程序都自动具备的标题按钮。它是如何生成一步一步地换页这样的错觉的呢？请参考实例 26。

■ Developer Studio中有一个向导，它可创建自己定制的 AppWizard。在MFC文档中可找到十分详细的例子。

5. 使用光盘时注意

当执行随书附送光盘上的工程时，将注意到该程序的表现与一个标准的向导类似。

6. 程序清单 —— 属性页类

```
#if !defined( AFX_PAGE1_H__A846F4D3_ECA0_11D1_A18D_DCB3C85EBD34__INCLUDED_ )
#define AFX_PAGE1_H__A846F4D3_ECA0_11D1_A18D_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif    // _MSC_VER >= 1000
// Page1.h : header file
//


/////////////////////////////////////////////////////////////////////////////
// CPage1 dialog

class CPage1 : public CPropertyPage
{
   DECLARE_DYNCREATE( CPage1 )

// Construction
public:
   CPage1();
   ~CPage1();

// Dialog Data
   // {{AFX_DATA( CPage1 )
   enum { IDD = IDD_PAGE1 };
   BOOL m_bPage3Check;
   // }}AFX_DATA


// Overrides
```

```
    // ClassWizard generate virtual function overrides
    // {{AFX_VIRTUAL( CPage1 )
public:
    virtual LRESULT OnWizardBack();
    virtual LRESULT OnWizardNext();
    virtual BOOL OnWizardFinish();
    virtual BOOL OnSetActive();
protected:
    virtual void DoDataExchange( CDataExchange* pDX );    // DDX/DDV support
    // }}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    // {{AFX_MSG( CPage1 )
    virtual BOOL OnInitDialog();
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CWzdSheet *m_pSheet;
};

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

#endif
    // !defined( AFX_PAGE1_H__A846F4D3_ECA0_11D1_A18D_DCB3C85EBD34__INCLUDED_ )
// Page1.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdSheet.h"
#include "Page1.h"

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

/////////////////////////////////////////////////////////////////////////////
// CPage1 property page

IMPLEMENT_DYNCREATE( CPage1, CPropertyPage )

CPage1::CPage1() : CPropertyPage( CPage1::IDD )
{
    // {{AFX_DATA_INIT( CPage1 )
    m_bPage3Check = FALSE;
```

```
    // }}AFX_DATA_INIT
}

CPage1::~CPage1()
{
}

void CPage1::DoDataExchange( CDataExchange* pDX )
{
CPropertyPage::DoDataExchange( pDX );
    // {{AFX_DATA_MAP( CPage1 )
    DDX_Check( pDX, IDC_PAGE3_CHECK, m_bPage3Check );
    // }}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP( CPage1, CPropertyPage )
    // {{AFX_MSG_MAP( CPage1 )
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPage1 message handlers

BOOL CPage1::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    m_pSheet = ( CWzdSheet* )GetOwner();

    return TRUE;            // return TRUE unless you set the focus to a control
                            //     EXCEPTION: OCX Property Pages should return FALSE
}

BOOL CPage1::OnSetActive()
{
    m_pSheet -> SetWizardButtons( PSWIZB_NEXT );
        // PSWIZB_BACK,PSWIZB_FINISH,PSWIZB_DISABLEDFINISH

    return CPropertyPage::OnSetActive();
}

LRESULT CPage1::OnWizardBack()
{

    return CPropertyPage::OnWizardBack();
}

LRESULT CPage1::OnWizardNext()
{
    UpdateData( TRUE );
    if ( m_bPage3Check )
```

```
    {
        return IDD_PAGE3;
    }
    else
    {
        return CPropertyPage::OnWizardNext();
    }
}

BOOL CPage1::OnWizardFinish()
{

    return CPropertyPage::OnWizardFinish();
}
```