第 7 章　对话框和对话条

对话框和对话条可以提示用户使用一些类型的控件，例如按钮和列表框。对话条是工具栏和对话框的混合物，对话条可以停靠到应用程序的某一边，并同时可以包含所有可用于对话框的控件。

MFC和Developer Studio将为用户自动创建对话框和类。在这一章中，我们将着眼于几种用手工修改对话框的方法，使得对话框对用户更为有用，本章的实例包括：

实例21　动态改变对话框的尺寸，单击一个按钮将使对话框扩大，以便允许在上面添加更多的控件。

实例22　自定义数据交换(Customizing Data Exchange)和验证(Validation)，将看到如何创建用户自己的可在对话框的类成员变量和控件之间进行交换的数据类型。

实例23　重载通用文件对话框，将看到如何定制文件对话框，这个对话框对于所有的应用程序都适用，并且可以免去令人头痛的编写代码工作。

实例24　重载通用颜色对话框，与前一个例子相同，只是在该例中看到的是颜色对话框。

实例25　获得目录名，将看到如何调用 API来获得目录名而不是整个文件的路径。

实例26　子对话框，将使用一个对话框，该对话框就像是另一个对话框中的普通控件。

实例27　子属性表，将使用一个属性表，该属性表就像是另一个对话框中的普通控件。

7.1　实例21：动态改变对话框的尺寸

1. 目标

为了在对话框上添加更多的控件将对话框放大，如图 7-1所示，单击More >>按扭可将对话框尺寸放大。

图7-1　动态改变对话框的尺寸

2. 策略

以通常方式创建一个具有 More >> 按钮的对话框资源和对话框类，但在对话框类的OnInitDialog()函数中用CWnd::MoveWindow()函数来将对话窗口缩为More >>按钮。这样，当单击More >>按钮时，将使用MoveWindow()函数来切换对话窗口的尺寸：从大到小或从小到

大。

3. 步骤

切换对话框的尺寸

按通常方式，用 Dialog Editor和ClassWizard创建对话框模板和对话框类，在窗口中央添加More >>按钮，并将其他控件放在它的一侧。

用ClassWizard重载OnInitDialog()函数，在OnInitDialog()函数中首先保存当前对话框的尺寸，该尺寸将是对话框的最大尺寸，然后定位到 More>>按钮代码并使窗口缩小为该按钮大小，代码如下：

```
BOOL CWzdDialog::OnInitDialog()
{
   CDialog::OnInitDialog();

   // save our full size
   GetWindowRect( &m_rectFull );
   m_rectHalf = m_rectFull;

   // calculate our half size based on bottom of "More" button
   CRect rect;
   m_ctrlMoreButton.GetWindowRect( &rect );
   m_rectHalf.bottom = rect.bottom+10;      // + 10 for cosmetics

   // toggle window size
   ToggleSize();

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

由于More >>按钮也将切换对话框窗口的尺寸，因此在下面要填充的 ToggleSize()函数中写入控制逻辑代码来缩小窗口的尺寸。

用ClassWizard为More >>按钮增加消息处理函数来调用 ToggleSize()函数:

```
void CWzdDialog::OnMoreButton()
{
   ToggleSize();
}
```

ToggleSize()函数将按钮上的名字改变为 <<Less，然后用CWnd::SetWindowPos()来改变窗口的尺寸，代码如下：

```
void CWzdDialog::ToggleSize()
{
   CRect rect;
   CString str;
   if ( m_bToggleSize )
   {
      str = "<< &Less";
      rect = m_rectFull;
   }
   else
```

```
    {
        str = "&More >>";
        rect = m_rectHalf;
    }
    SetWindowPos( NULL,0,0,rect.Width(),rect.Height(),
        SWP_NOZORDER|SWP_NOMOVE );
    m_ctrlMoreButton.SetWindowText( str );
    m_bToggleSize = !m_bToggleSize;
}
```

要查看该对话框类的全部程序代码，请参考本实例结尾处的程序清单 —— 对话框类。

4. 注意

如果用户愿意在More >>按钮和<<Less按钮上添加更多的饰物，那么用 Dialog Editor 设置其类型为BS_BITMAP，并且用CButton 中的SetBitmap()函数来改变位图。

5. 使用光盘时注意

运行随书附带光盘中的工程时，单击 Test按钮，然后再单击 Wzd菜单命令，打开对话框，然后再单击More >>按钮进一步打开对话框。

6. 程序清单 —— 对话框类

```
#if !defined( AFX_WZDDIALOG_H__1EC8499A_C589_11D1_9B5C_00AA003D8695__INCLUDED_ )
#define AFX_WZDDIALOG_H__1EC8499A_C589_11D1_9B5C_00AA003D8695__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CWzdDialog dialog

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

// Dialog Data
    // {{AFX_DATA( CWzdDialog )
    enum { IDD = IDD_WZD_DIALOG };
    CButton m_ctrlMoreButton;
    // }}AFX_DATA


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

// Implementation
```

```
protected:

    // Generated message map functions
    // {{AFX_MSG( CWzdDialog )
    virtual BOOL OnInitDialog();
    afx_msg void OnMoreButton();
    // }}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL m_bToggleSize;
    CRect m_rectFull;
    CRect m_rectHalf;
    void ToggleSize() ;
};

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

#endif
    // !defined( AFX_WZDDIALOG_H__1EC8499A_C589_11D1_9B5C_00AA003D8695__INCLUDED_ )
// WzdDialog.cpp : implementation file
//

#include "stdafx.h"
#include "wzd.h"
#include "WzdDialog.h"

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

/////////////////////////////////////////////////////////////////////////////
// CWzdDialog dialog


CWzdDialog::CWzdDialog( CWnd* pParent /* = NULL*/ )
    : CDialog( CWzdDialog::IDD, pParent )
{
    // {{AFX_DATA_INIT( CWzdDialog )
    // NOTE: the ClassWizard will add member initialization here
    // }}AFX_DATA_INIT
    m_bToggleSize = FALSE;
}


void CWzdDialog::DoDataExchange( CDataExchange* pDX )
{
    CDialog::DoDataExchange( pDX );
    // {{AFX_DATA_MAP( CWzdDialog )
    DDX_Control( pDX, IDC_MORE_BUTTON, m_ctrlMoreButton );
    // }}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP( CWzdDialog, CDialog )
    // {{AFX_MSG_MAP( CWzdDialog )
    ON_BN_CLICKED( IDC_MORE_BUTTON, OnMoreButton )
    // }}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CWzdDialog message handlers

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

    // save our full size
    GetWindowRect( &m_rectFull );
    m_rectHalf = m_rectFull;

    // calculate our half size based on bottom of "More" button
    CRect rect;
    m_ctrlMoreButton.GetWindowRect( &rect );
    m_rectHalf.bottom = rect.bottom+10;          // + 10 for cosmetics

    // toggle window size
    ToggleSize();

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

void CWzdDialog::OnMoreButton()
{
    ToggleSize();
}

void CWzdDialog::ToggleSize()
{
    CRect rect;
    CString str;
    if ( m_bToggleSize )
    {
        str = "<< &Less";
        rect = m_rectFull;
    }
    else
    {
        str = "&More >>";
        rect = m_rectHalf;
    }
    SetWindowPos( NULL,0,0,rect.Width(),rect.Height(),SWP_NOZORDER|SWP_NOMOVE );
    m_ctrlMoreButton.SetWindowText( str );
    m_bToggleSize = !m_bToggleSize;
}
```

7.2 实例22：自定义数据交换并验证

1. 目标

创建一个新的数据交换和验证函数来支持在对话框控件和对话框类成员变量之间的动态数据交换。

2. 策略

正如本系列丛书先前书中所看到的，动态数据交换 (Dynamic data exchange)允许对话框控件和对话框类的成员变量之间自动交换数据，而动态数据验证 (Dynamic data validation)则可以自动检测用户输入数据中的语法错误。 MFC已经支持几种数据类型 (例如int和double数据类型)的数据交换和验证，但仍然可以通过提供用户自己的静态转换函数来支持自定义的数据类型(例如COctalString)，其中该静态函数使用适当的命名和调用约定 (如: DDX_something()或DDV_something())。在这个实例中，将对二进制数据串进行数据交换和验证，其中二进制数据串是由实例64中的自定义CString类所提供的。

3. 步骤

1) 创建自定义的数据交换函数

用下面的框架代码 (outline)创建静态例程，对于一个完整的实例，请参考实例结尾处的程序清单 —— 二进制数据交换和验证函数。

```
void AFXAPI DDX_Xxxx( CDataExchange* pDX, int nIDC, CXxxx& value )
{
   CWzdString str;
   HWND hWndCtrl = pDX -> PrepareEditCtrl( nIDC );

   // getting data from control
   if ( pDX -> m_bSaveAndValidate )
   {
      // hWndCtrl == window handle of the control
      int nLen = ::GetWindowTextLength( hWndCtrl );
      ::GetWindowText( hWndCtrl, str.GetBufferSetLength( nLen ),
         nLen+1 );
      str.ReleaseBuffer();

         // temporary str now has text data, now put into member variable

   }
     // putting data into control window
     else
   {
         // copy data from member variable into str

         // sets window text only if it's different
         AfxSetWindowText( hWndCtrl, str );
   }
}
```

2) 创建自定义的数据验证函数

用下面的框架代码 (outline)创建静态例程，对于一个完整的实例，请参考实例结尾处的程序清单 —— 二进制数据交换和验证函数。

```
void AFXAPI DDV_MaxChars( CDataExchange* pDX, CByteArray const& value,
    int nBytes )
{
    // if retrieving from control, make check of value
    if ( pDX -> m_bSaveAndValidate )
    {
        if ( error )
        {
            CString str;
            str.Format( "Maximum characters you can enter is %d!",
                nBytes );
            AfxMessageBox( str, MB_ICONEXCLAMATION );
            pDX -> Fail();
        }
    }
    // else if sending to control, setup any control function
    //     that will allow it to perform it's own check
    else if ( pDX -> m_hWndLastControl !=
        NULL && pDX -> m_bEditLastControl )
    {
        ///
    }
}
```

3) 实现新的数据交换和验证函数

在对话框类中包含对新的数据的定义。

```
#include  "WzdXchng.h"
```

插入对话框类中DoDataExchang()例程所需的函数，但是必须插在{{}}括号后面，这样才能使ClassWizard不发现它们：

```
void CWzdDialog::DoDataExchange( CDataExchange* pDX )
{
    CDialog::DoDataExchange( pDX );
    // {{AFX_DATA_MAP( CWzdDialog )
    // }}AFX_DATA_MAP
    DDX_Text( pDX, IDC_WZD_EDIT, m_WzdArray );    <<<<
    DDV_MaxChars( pDX, m_WzdArray, 7 );          <<<<
}
```

确保数据验证函数在其检测到数据交换函数之后马上被调用，CdataExchange中m_hWndLastControl函数的成员变量由数据交换函数初始化，当然用户也可以在数据验证程序中重新初始化该变量，但是这样要浪费CPU的周期，是没有必要的。

4. 注意

没有必要创建一个单独的数据交换或数据验证函数，相反应该直接修改 DoDataExchange ()例程。在ClassWizard 的{{}}括号后面增加一些附加函数代码，使这些添加的函数代码与上面的函数类似，并且记住当 m_bSave And Validate为TRUE时，应当从控件转移到对话框的成员变量。

```
void CWzdDialog::DoDataExchange( CDataExchange* pDX )
{
```

```
    CDialog::DoDataExchange( pDX );
    // {{AFX_DATA_MAP( CWzdDialog )
    :   :   :
    // }}AFX_DATA_MAP
    if ( pDX -> m_bSaveAndValidate )
    {
        // get from control
    }
    else
    {
        // store to control
    }
}
```

当然这个方法的缺点是用户不能轻易地从新的逻辑代码转移到另一个对话框类。

5. 使用光盘时注意

运行随书附带光盘上的工程时，在WzdView.cpp中的OnTestWzd()函数中设置断点，单击Test按钮和Wzd然后输入一个数字到对话框中，观察它是如何自动转换成字节数组(byte array)的。

6. 程序清单 —— 二进制数据转换和验证函数

```
#ifndef WZDXCHNG_H
#define WZDXCHNG_H

void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, CByteArray& value );
void AFXAPI DDV_MaxChars( CDataExchange* pDX, CByteArray const& value,
    int nBytes );


#endif
// WzdXchng
//

#include "stdafx.h"
#include "WzdXchng.h"
#include "WzdString.h"
#include <afxpriv.h>

////////////////////////////////////////////////////////////////////////////
// CWzdXchng

void AFXAPI DDX_Text( CDataExchange* pDX, int nIDC, CByteArray& value )
{
    CWzdString str;
    HWND hWndCtrl = pDX -> PrepareEditCtrl( nIDC );
    if ( pDX -> m_bSaveAndValidate )
    {
        int nLen = ::GetWindowTextLength( hWndCtrl );
        ::GetWindowText( hWndCtrl, str.GetBufferSetLength( nLen ), nLen+1 );
        str.ReleaseBuffer();
        int size = str.GetLength();
        if ( size%2 )
        {
```

```
            AfxMessageBox( "Please enter even number of digits." );
            pDX -> Fail();                          // throws exception
        }
        size /= 2;
        value.SetSize( size );
        str.GetBinary( value.GetData(),size );
    }
    else
    {
        str.PutBinary( value.GetData(),value.GetSize() );
        AfxSetWindowText( hWndCtrl, str );          // sets window text only if
                                                    //     it's different
    }
}

void AFXAPI DDV_MaxChars( CDataExchange* pDX, CByteArray const& value,
    int nBytes )
{
    if ( pDX -> m_bSaveAndValidate && value.GetSize() > nBytes )
    {
        CString str;
        str.Format( "Maximum characters you can enter is %d!", nBytes );
        AfxMessageBox( str, MB_ICONEXCLAMATION );
        str.Empty();          // will not return after exception
        pDX -> Fail();
    }
    else if ( pDX -> m_hWndLastControl != NULL && pDX -> m_bEditLastControl )
    {
        // set edit box not to allow more then these bytes
        // note that this function depends on being called
        //     right after the DDX function
        ::SendMessage( pDX -> m_hWndLastControl, EM_LIMITTEXT, nBytes*2, 0 );
    }
}
```

7.3　实例23：重载通用文件对话框

1. 目标

在通用文件对话框上增加一些自己的特色，如图 7-2所示。

2. 策略

从Common File Dialog类中派生出用户自己的新类 CfileDialog，而且为对话框增加一些控件。同时使用其构造函数中的参数以及设置其OPENFILENAME结构来定制该对话框。

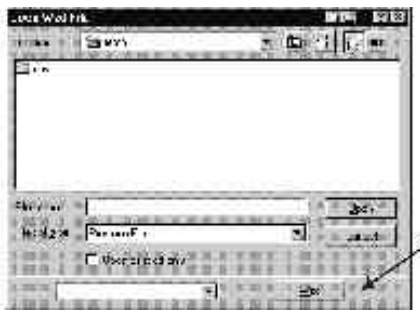

图7-2　自定义文件对话框

3. 步骤

1) 创建新的通用文件对话框类

注意 如果用户不希望为文件对话框增加自己的控件，便不需要创建新的类，因此可以略过这一部分。

使用ClassWizard从CFfileDialog中派生新的类。

用Dialog Editor创建一个小的对话框模板，模板中仅包括用户希望添加到通用文件对话框上的控件。该通用对话框将自动创建对话框中可见的其余标准控件，因此用户不必担心它们，只需创建新的控件即可。模板应该具有一个不带框架的子窗口风格。

用ClassWizard为新的控件添加消息处理函数，为了访问由文件对话框自动添加的控件，可以用GetDlgItem()函数和下面任一个ID号，ID与控件匹配关系在以后的Windows版本中可能会改变:

```
psh1,psh2,...,psh12        // push buttons
chx1-12                    // checkboxes
rad1-12                    // radio buttons
grp1-12                    // group boxes
stc1-12                    // statics
lst1-12                    // listboxes/views
cmb1-12                    // comboboxes
edt1-12                    // edit boxes
scr1-12                    // scrollbars
```

2) 初始化通用文件对话框类

为使用CFileDialog对话框类或自己的派生类，首先构造通用文件对话框类如下：

```
CFileDialog dlg(
   TRUE,                              // TRUE = create a File Open dialog,
                                      // FALSE = create a File Save As
                                      //    dialog
   _T(".log"),                        // default filename extension
   " ",                               // initial filename in edit box
                                      //    functionality flags
   OFN_ALLOWMULTISELECT|              // allow multiple files
                                      //    to be selected
   // OFN_CREATEPROMPT |              // if File Save As, prompts user
                                      //    if they want to create
                                      //    non-existant file
   // OFN_OVERWRITEPROMPT |           // if File Save As—prompts user to
                                      //    ask if they want to overwrite
                                      //    an existing file
   // OFN_ENABLESIZING |              // if Windows NT 5.0 or Win 98,
                                      //    causes box to be resizable
                                      //    by user
   // OFN_EXTENSIONDIFFERENT|         // allows user to enter a filename
                                      //    with a different extension
                                      //    from the default
   // OFN_FILEMUSTEXIST               // file must exist
   // OFN_NOLONGNAMES |               // causes dialog to use short
                                      //    filenames (8.3)
   // OFN_PATHMUSTEXIST |             // user can only type valid paths
```

```
                                        //    and filenames
    // OFN_NOVALIDATE |                 // the returned filname can have
                                        //    invalid characters
                                        //    appearence flags
    // OFN_HIDEREADONLY |               // hides read-only check box
    // OFN_NONETWORKBUTTON |            // hides Network button
    // OFN_READONLY |                   // initially check Read Only
                                        //    check box
    // OFN_SHOWHELP |                   // Help button appears—when clicked
                                        //    the hook procedure gets a
                                        //    CDN_HELP message
    // custom template flags
    OFN_ENABLETEMPLATE |                // you will be supplying your own
                                        //    custom dialog box template
    0,
    "Accounting Files ( *.log;*.txt )|*.log;*.txt|All Files
       ( *.* )|*.*||",                  // file filter
       NULL);                           // parent window
```

如果要在对话框中添加自己的控件，那么也必须定义文件对话框打开时的对话框模板，代码如下：

```
dlg.m_ofn.lpTemplateName=MAKEINTRESOURCE(IDD_WZD_FILEOPEN);
```

当构造上面的类时，必须设置OFN_ENABLETEMPLATE 标志。

用下面的代码使通用文件对话框在开始时使用一个特定目录：

```
// set an initial directory
char lpszInitDir[] = {"c:\\temp"};
dlg.m_ofn.lpstrInitialDir = lpszInitDir;
```

用下面的代码将文件对话框命名为Open或Save File As:

```
//set the dialog's title
char lpszTitle[] = {"Open Wzd File"};
dlg.m_ofn.lpstrTitle = lpszTitle;
```

为设置一个在文件对话框关闭后仍然保留的文件过滤器，首先设置一个如下所示的静态字符串：

```
static char lpstrCustomFilter[255] = {"Previous Filter\0*.log\0"};
```

然后用它的地址初始化文件对话框，如下所示：

```
// retain the customer's last file filter selection
dlg.m_ofn.lpstrCustomFilter = lpstrCustomFilter;
dlg.m_ofn.nMaxCustFilter = 255;
```

3) 创建通用文件对话框和检索值

用CfileDialog ::DoModal() 函数打开一个文件对话框，然后检查用户是否已经单击 IDOK，如果是，则从对话框中获得该值：

```
if ( dlg.DoModal() == IDOK )
{
```

检索用户选择的文件名，这可以使用下面任何一个函数：

```
CString path = dlg.GetPathName();           // ex: c:\temp\temp.tmp
```

```
CString file = dlg.GetFileName();          // ex: temp.tmp
CString title = dlg.GetFileTitle();        // ex: temp
CString ext = dlg.GetFileExt();            // ex: tmp
```

可以用下面的代码确定用户选择了哪一个文件过滤器：

```
int nFilterIndex=dlg.m_ofn.nFilterIndex;
```

该过滤器作为索引返回到各种可能的文件过滤器列表中。

为了获得只读(Read Only)复选框的状态，可以用下面的代码：

```
BOOL bReadOnly =dlg.GetReadOnlyPref();
```

如果用户选择了多个文件(在上面设置了OFN_ALLOWMULTISELECT)，那么可以用下面的代码翻动它们：

```
    for (POSITION pos = dlg.GetStartPosition();pos;)
    {
        CString pathx = dlg.GetNextPathName(pos);
            // ex: c:\temp\temp.tmp
    }
}
```

注意，当选择多个文件时，必须分析文件名以获得文件的标题和扩展名等等。

4. 注意

这个实例演示了一种修改文件浏览器风格的 File Open 对话框的方法，用这种风格，用户不但可以打开需要的文件，而且可以执行与 Windows Explorer一样的文件维护功能，包括创建新的目录，删除旧的文件等。另一个旧的 File Open对话框版本也可以用，旧风格的优点在于每个对话框控件都比较容易使用，因为用户接触到编辑实际的对话框模板，这个模板称为FILEOPEN.DLG，可以在\VC\INCLUDE文件夹找到。如果用户愿意使用旧版本的 File Open对话框，可参考实例 24，该实例说明了如何使用 \VC\INCLUDE 中的模板重载通用文件对话框。当然，用户也可以用 FILEOPEN.DLG，不管怎样，必须在使用下面代码构造文件对话框类时设置OFN_EXPLORER标志，因为在构造CFileDialog类时这个标志将自动设置。

```
dlg.m_ofn.flags&=~OFN_EXPLORER;
```

另外，尽量不要使用 CFileDialog提供的Help，用户应该通过添加和运行 Help来得到更好的使用效果。

5. 使用光盘时注意

运行随书附带光盘上的工程时，单击 Test按钮和Wzd打开通用文件对话框，注意到这个对话框除了在对话框底部增加了两个新控件以外，与在 Windows中遇到的大多数文件对话框都类似。

7.4 实例24：重载通用颜色对话框

1. 目标

在通用颜色对话框上增加一些自己的特色，如图 7-3所示。

2. 策略

将从Common Dialog类中派生出新类CColorDialog，而且为对话框增加一些控件。同时使用其构造函数中的参数以及设置其 CHOOSECOLOR结构来定制该对话框。

3. 步骤

1) 创建新的通用颜色对话框类

注意 如果用户不希望为颜色对话框增加自己的控件，可以不需要创建新的类，因此可以略过这一部分。

使用ClassWizard从CColorDialog中派生出新的类。

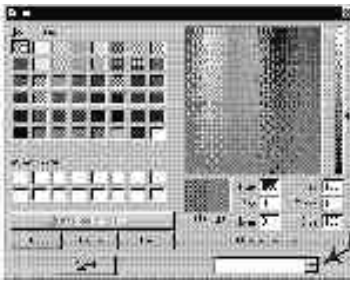

图7-3 定制颜色对话框

这一步必须剪切标准颜色对话框模板并粘贴到应用程序的源代码中，该标准模板可以在 \VC\INCKUDE目录下的COLOR.DLG中找到，用文本编辑器打开它并把它粘贴到 .rc文件的对话框部分中。

同时必须将 COLORDLG.H的资源包含文件添加到自己的源文件中，这可以通过单击Developer Studio中的View和Resource Includes菜单项来打开Resource Includes对话框，然后进入COLORDLG.H来实现。

这一步使用Dialog Editor来编辑对话框，可以移动、删除和添加这些控件 —— 只是不能改变现有控件的ID号，颜色对话框将会寻找它们的 ID号。

2) 初始化通用颜色对话框类

为使用CColorDialog对话框类或自己的派生类，首先构造通用颜色对话框类如下：

```
CWzdColorDialog dlg(
   rgb,                          // initial color selection
   // CC_FULLOPEN|               // intially opens both sides
                                 //    of color dialog
   // CC_PREVENTFULLOPEN|        // prevent user from opening customize
                                 //    side of color dialog
   CC_SHOWHELP|                  // creates help button
   // CC_SOLIDCOLOR|             // display only non-dithered colors
   CC_ENABLETEMPLATE|            // use a custom dialog template
   0,
   NULL                          // parent window
);
```

如果使用自己的对话框模板，那么同时必须在 ColorDialog的CHOOSECOLOR结构中定义该模板：

```
dlg.m_cc.lpTemplateName=MAKEINTRESOURCE(IDD_WZN_CHOOSECOLOR);
```

当构造上面的类时，必须设置 CC_ENABLETEMPLATE 标志。

为在CColorDialog的调用之间保存用户所定制的颜色，必须提供指向 16个COLORREF值的静态数组的指针，代码如下：

```
COLORREF lpCustColors[16];
   :  :  :
   dlg.m_cc.lpCustColors = lpCustColors;
```

3) 创建通用颜色对话框

用CColorDialog ::DoModal()函数打开一个通用颜色对话框，然后检查用户是否已经单击IDOK，如果单击，那么从对话框中获取其值：

```
if ( dlg.DoModal() == IDOK )
{
   rgb = dlg.GetColor();
}
```

4. 注意

■ 除了以上介绍的文件对话框和颜色对话框之外，还有用于字体设置、查找和替换文本、打开文件、页面设置和文件打印的通用对话框，打开文件的通用对话框将在最后的实例中演示。

■ 颜色对话框通常用于选择散射(diffused)颜色，但使用CC_SOLIDCOLOR风格来创建颜色对话框时，可用它来选择非散射颜色。不幸的是，在一个只有256种颜色的系统上颜色的选择可能变得非常少。为在一个只有256种颜色的系统提供给用户更广泛的非散射颜色选择范围，请参考实例31。

5. 使用光盘时注意

运行随书附带光盘上的工程时，单击Test按钮和Wzd打开通用颜色对话框，用户将会注意到这个对话框除了在对话框底部增加了两个新控件以外，与在Windows中遇到的大多数颜色对话框都类似。

7.5 实例25：获得目录名

1. 目标

创建一个可显示系统驱动器的子目录结构的树形控件来显示系统磁盘的目录，如图7-4所示。

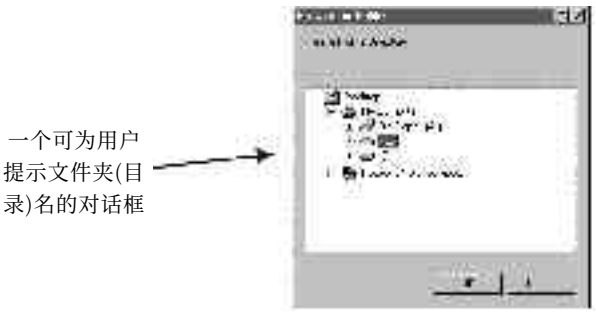

图7-4 目录提示对话框

2. 策略

用户可能认为通用文件对话框可以处理所有的文件提示，但某一天可能忽然意识到自己应该具有一个只用于提示文件目录的应用程序，同时也意识到没有捷径来使用通用文件对话框为用户提示目录信息。虽然可以采用使控件不可见以及使编辑控件在用户单击目录时返回这种方式，但是这会弄得一团糟，而且采用这种方式的通用文件对话框，当用于提示目录信息时，给用户看起来的外观和用起来的感觉与其他微软应用程序不匹配，那么该用什么呢？答案：使用MFC文档中的一个小小的::SHBrowse-ForFolder() API调用。

由于::SHBrowseForFolder()需要一些特殊的处理，本例将在自己的Directory类中封装其功能。

3. 步骤

1) 创建目录类

用ClassWizard创建新的类，且不从任何类中派生。本例要使用的API不需要从任何MFC类中继承功能。

向该类中添加GetDirectory()函数，::SHBrowseForFolder()在创建BROWSEINFO结构时将

启动该函数。

```
CString CWzdDirDlg::GetDirectory( CWnd *pParent,LPCSTR lpszRoot,
   LPCSTR lpszTitle )
{
   CString str;
   BROWSEINFO bi;
   bi.hwndOwner = pParent -> m_hWnd;          // owner of created dialog box
   bi.pidlRoot = 0;                           // unused
   bi.pszDisplayName = 0;                     // buffer to receive name
                                              //    displayed by folder
                                              //    (not a valid path)
   bi.lpszTitle = lpszTitle;                  // title is "Browse for
                                              //    Folder", this is
                                              //    an instruction
   bi.lpfn = BrowseCallbackProc;              // callback routine called
                                              //    when dialog has been
                                              //    initialized
   bi.lParam = 0;                             // passed to callback routine
   bi.ulFlags =
      BIF_RETURNONLYFSDIRS |                  // only allow user to select
                                              //    a directory
      BIF_STATUSTEXT |                        // create status text field
                                              //    we will be writing to
                                              //    in callback
      // BIF_BROWSEFORCOMPUTER|               // only allow user to select
                                              //    a computer
      // BIF_BROWSEFORPRINTER |               // only allow user to select
                                              //    a printer
      // BIF_BROWSEINCLUDEFILES|              // displays files too which
                                              //    user can pick
      // BIF_DONTGOBELOWDOMAIN|               // when user is exploring the
                                              //    "Entire Network" they
                                              //    are not allowed into
                                              //    any domain
      0;
   m_sRootDir = lpszRoot;                     // save for callback routine
```

下一步，可以调用 :SHBrowseForFolder()提示用户输入目录名，随后可以使用 ::SHGet PathFromIDList()获取该目录名:

```
   LPITEMIDLIST lpItemId = ::SHBrowseForFolder( &bi );
   if ( lpItemId )
   {
      LPTSTR szBuf = str.GetBuffer( MAX_PATH );
      ::SHGetPathFromIDList( lpItemId, szBuf );
      ::GlobalFree( lpItemId );
      str.ReleaseBuffer();
   }

   return str;
}
```

在第1步中，我们指定了一个回调例程。只有在需要进一步更改 ::SHBrowse ForFolder()行为的时候才需要使用回调例程。如果指定 NULL，则说明不需要回调例程。但是，为了在打开提示时实现一个初始目录，则需要使用一个回调例程！所以，加入下例回调函数以修改提示的初始目录:

```
int CALLBACK BrowseCallbackProc( HWND hwnd,UINT msg,LPARAM lp,
   LPARAM pData )
{
   TCHAR buf[MAX_PATH];

   switch( msg )
   {
      // when dialog is first initialized, change directory
      //     to one chosen above
      case BFFM_INITIALIZED:
         strcpy( buf,CWzdDirDlg::m_sRootDir );
         ::SendMessage( hwnd,BFFM_SETSELECTION,TRUE,( LPARAM) buf );
         break;
```

如果选择上面的BIF_STATUSTEXT风格，则用户每次选择不同的文件夹时都可以在这个回调程序中设置其状态:

```
      case BFFM_SELCHANGED:
         if ( ::SHGetPathFromIDList( ( LPITEMIDLIST ) lp ,buf ) )
            SendMessage( hwnd,BFFM_SETSTATUSTEXT,0,( LPARAM )buf );
         break;
   }
   return 0;
}
```

该类的完整程序代码可参考本实例结尾处的程序清单 —— 目录类。

2) 使用目录类

为用户提示目录现在是本实例中的目录类调用其 GetDirectory()函数所完成的一件轻而易举的事情:

```
// get current working directory
char buf[MAX_PATH];
_getcwd( buf,MAX_PATH );

CWzdDirDlg dlg;
CString dir = dlg.GetDirectory(
   this,      // parent window
   buf,       // root directory
   "Title"    // optional title
   );
```

4. 注意

用户可能已经注意到了 BROWSEINFO结构中的pidlRoot成员变量，也许会认为它是用来指定一个初始的目录，而事实上它除了使用户花费几个小时试图使其工作以外好像别无用处。

5. 使用光盘时注意

运行随书附带光盘上的工程时，单击 Test按钮然后单击Wzd菜单命令打开目录，此时将提

示已经打开的当前工作目录。

6. 程序清单 —— 目录类

```
#if !defined( AFX_WZDDRDLG_H__D52656D3_8A25_11D2_9C53_00AA003D8695__INCLUDED_ )
#define AFX_WZDDRDLG_H__D52656D3_8A25_11D2_9C53_00AA003D8695__INCLUDED_

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

class CWzdDirDlg
{
public:
    CWzdDirDlg();
    virtual ~CWzdDirDlg();

    CString GetDirectory( CWnd *pParent = NULL,
        LPCSTR lpszRoot = "c:\\", LPCSTR lpszTitle = "Please Pick a Directory" );

    static CString m_sRootDir;

};
int CALLBACK BrowseCallbackProc( HWND hwnd,UINT uMsg,LPARAM lp, LPARAM pData );

#endif
    // !defined( AFX_WZDDRDLG_H__D52656D3_8A25_11D2_9C53_00AA003D8695__INCLUDED_ )
// WzdDrDlg.cpp: implementation of the CWzdDirDlg class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "WzdDrDlg.h"
#include "Shlobj.h"

CString CWzdDirDlg::m_sRootDir;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CWzdDirDlg::CWzdDirDlg()
{

}

CWzdDirDlg::~CWzdDirDlg()
{

}

CString CWzdDirDlg::GetDirectory( CWnd *pParent,LPCSTR lpszRoot,LPCSTR lpszTitle )
{
    CString str;
```

```
    BROWSEINFO bi;
    bi.hwndOwner = pParent -> m_hWnd;          // owner of created dialog box
    bi.pidlRoot = 0;                           // unused
    bi.pszDisplayName = 0;                     // buffer to receive name displayed by
                                               //    folder (not a valid path)
    bi.lpszTitle = lpszTitle;                  // title is “Browse for Folder”,
                                               //    this is an instruction
    bi.lpfn = BrowseCallbackProc;              // callback routine called when dialog
                                               //    has been initialized
    bi.lParam = 0;                             // passed to callback routine
    bi.ulFlags =
       BIF_RETURNONLYFSDIRS |                  // only allow user to select
                                               //    a directory
       BIF_STATUSTEXT |                        // create status text field we will be
                                               //    writing to in callback
       // BIF_BROWSEFORCOMPUTER|                // only allow user to select a computer
       // BIF_BROWSEFORPRINTER |                // only allow user to select a printer
       // BIF_BROWSEINCLUDEFILES|               // displays files too which user
                                               //    can pick
       // BIF_DONTGOBELOWDOMAIN|                // when user is exploring the “Entire
                                               //    Network” they are not allowed
                                               //    into any domain
       0;
    m_sRootDir = lpszRoot;

    LPITEMIDLIST lpItemId = ::SHBrowseForFolder( &bi );
    if ( lpItemId )
    {
       LPTSTR szBuf = str.GetBuffer( MAX_PATH );
       ::SHGetPathFromIDList( lpItemId, szBuf );
       ::GlobalFree( lpItemId );
       str.ReleaseBuffer();
    }

    return str;
}

int CALLBACK BrowseCallbackProc( HWND hwnd,UINT msg,LPARAM lp, LPARAM pData )
{
    TCHAR buf[MAX_PATH];

    switch(msg)
    {
       // when dialog is first initialized, change directory to one chosen above
       case BFFM_INITIALIZED:
          strcpy( buf,CWzdDirDlg::m_sRootDir );
          ::SendMessage( hwnd,BFFM_SETSELECTION,TRUE,( LPARAM )buf );
          break;

       // if you picked BIF_STATUSTEXT above, you can fill status here
       case BFFM_SELCHANGED:
       if ( ::SHGetPathFromIDList( ( LPITEMIDLIST ) lp ,buf ) )
```

```
            SendMessage( hwnd,BFFM_SETSTATUSTEXT,0,( LPARAM )buf );
        break;
    }
    return 0;
}
```

7.6 实例26：子对话框

1. 目标

使用一个就像是另一个对话框中通用控件一样的对话框，如图 7-5所示。

图7-5 另一个对话框中的子对话框

2. 策略

考虑到对话框自身的典型用途是作为一个弹出窗口，很难想到它也可以用做一个父窗口甚至另一个对话框的子窗口，因此本例只是创建一个作为另一个对话框的子窗口的对话框。

3. 步骤

为一个弹出的对话框添加一个子对话框

用对话框编辑器为每一个子对话框创建一个对话框模板，设置它们的属性为 Child，而且无边界。

使用ClassWizard为这些模板创建一个对话框类。

在父对话框类中嵌入每一个子对话框类：

```
CWzdDlg1 m_wzd1;
CWzdDlg2 m_wzd2;
  :  :
```

用Dialog Editor 为父对话框模板增加一个具有一定尺寸的静态控件，其位置和子对话框所希望出现的位置相同，这将作为一个占位符 (Placeholder)。

用ClassWizard为父对话框增加一个WM_INITDIALOG 消息处理函数，并获得这个静态控件的矩形区域：

```
CRect rect;
m_ctrlStatic.GetWindowRect( &rect );
ScreenToClient( &rect );
```

接下来在同一个消息处理函数中，创建子对话框并且将它们全部添加在静态控件上，确保只有其中一个可见：

```
m_dlg1.Create( IDD_DIALOG2,this );
m_dlg1.ShowWindow( SW_SHOW );
```

```
m_dlg1.MoveWindow( &rect );

m_dlg2.Create( IDD_DIALOG3,this );
m_dlg2.ShowWindow( SW_HIDE );
m_dlg2.MoveWindow( &rect );
```

根据其他一些控件，例如按钮和标签控件，来改变可见的子对话框。

在子对话框自己的类中处理来自这些子对话框的命令。

4. 注意

■ 可以用标签控件来控制哪个对话框可见，然而可以使用 CPropertySheet 和 CPropertyPage 类来获得更好的效果，这些类提供给用户按照这种思路来创建的其他功能。事实上，如果这个对话框上不是所有的控件都根据条件而改变，那么用户应该只用这种方法。

■ AppWizard 实际上就是用这种方法来显示它的页码而不是使用具有向导选项的属性表。

5. 使用光盘时注意

运行随书附带光盘上的工程时，单击 Test按钮然后单击Wzd菜单命令打开一个对话框，然后单击Button 1或Button 2按钮来显示特定的控件系列。

7.7 实例27：子属性表

1. 目标

使用一个就像是另一个对话框中通用控件一样的属性表，如图 7-6所示。

2. 策略

尽管Dilog Editor允许为一个对话框模板添加标签控件，然而它创建的控件除了告诉用户什么标签被按下以外，将不会具有任何其他的功能。这时就只有用户自己显示一个子对话框，如前一个实例所示。但本例并不按照这种方法，而是使用 Property Sheet控件并尽量避免使用其提供的功能。本例将使用CPropertySheet来创建该Property Sheet，创建时采用无模式风格然后将它添加到对话框上。

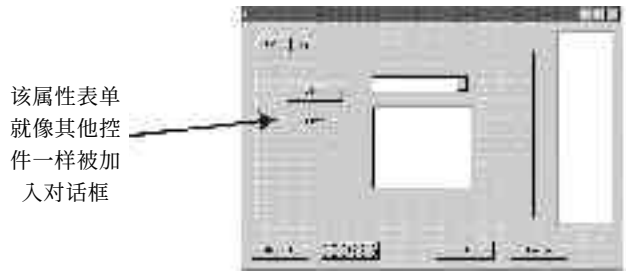

图7-6 属性表用做对话框控件

3. 步骤

为对话框增加属性表

用DialogEditor为每一个属性表页创建对话框模板，设置每一个模板为 Child属性并且没有边界。

用ClassWizard为每一个模板创建属性页类并且都从 CPropertyPage 中派生。

在对话框类中嵌入一个CPropertySheet类。

用Dialog Editor为对话框模板增加一个带尺寸的静态控件，放在属性表所希望出现的位置。

用ClassWizard为对话框增加一个WM_INITDIALOG 消息处理函数，并获得这个静态控件的尺寸和位置：

```
CRect rect;
m_ctrlStatic.GetWindowRect( &rect );
ScreenToClient( &rect );
```

在这个OnInitDialog函数中，为每一个属性表创建并添加页面：

```
m_pFirstPage = new CFirstPage;
m_pSecondPage = new CSecondPage;
m_sheet.AddPage( m_pFirstPage );
m_sheet.AddPage( m_pSecondPage );
```

最后，创建属性表并且把它添加在静态控件上：

```
m_sheet.Create( this,WS_VISIBLE|WS_CHILD );
m_sheet.MoveWindow( &rect );
```

4. 注意

用户也可以用这种方法在一个对话框中嵌入一个属性表向导，由于属性表向导的无模式窗体的标题栏中没有关闭或最小化按钮，因此可以用一个简单的方法来增加这些按扭：创建一个带关闭和最小化按扭的对话框，并且将一个具有 Wizard风格的属性表控件嵌入其上。

5. 使用光盘时注意

运行随书附带光盘上的工程时，单击 Test按钮然后单击Wzd菜单命令打开一个对话框，这个对话框中包含有其他一些控件,其中就有属性表控件。